UNIVERSIDAD DEL BÍO-BÍO
Facultad de Educación y Humanidades
Departamento de Ciencias de la Educación

Profesor Guía:
Dr. Pedro Salcedo Lagos

Cátedra:
Proyecto de Intervención e Innovación Pedagógica

UNA PROPUESTA DIDÁCTICA PARA LA ENSEÑANZA DE LA PROGRAMACIÓN ORIENTADA A OBJETOS QUE EMPLEA APRENDIZAJE BASADO EN PROBLEMAS EN ENTORNOS VIRTUALES

Milton Agustín Ramírez Klapp

Informe de Tesis para optar al grado de Magíster en Pedagogía para la Educación Superior

Chillán, Julio de 2013

Índice General

Índice de Tablas

Agradecimientos

Para abrir este trabajo, deseo expresar mis agradecimientos a:

Mi familia, por su tremendo respaldo y apoyo durante todo este proceso.

Mi prometida, por estar siempre «*en las buenas y en las malas*», aconsejarme en todo momento, prepararme las ricas tortillas que comía los días sábado, calentarme la comida en la mañana mientras me preparaba para ir a clases.

Mi amigo Gupi, que aunque no haya estado físicamente en esta última etapa, su recuerdo me acompañó en cada momento.

Los compañeros del MAPES, por la excelente convivencia que tuvimos.

Los profesores del MAPES, por sus conocimientos y dedicación.

Mi profesor guía Pedro Salcedo Lagos por su generosidad, tiempo, dedicación y sabios consejos.

Muchas gracias a todos.

Con afecto,

Milton Agustín Ramírez Klapp

Resumen

El trabajo que a continuación se expone, corresponde a la última etapa del desarrollo del Programa de Magíster en Pedagogía en Educación que imparte la Universidad del Bío-Bío, que busca contribuir a la optimización de la praxis docente en el ámbito de la enseñanza de la programación orientada a objetos para estudiantes de Ingeniería Civil Informática en la Universidad San Sebastián.

Esta propuesta tiene su génesis en el proyecto de investigación cualitativa que tiene como norte recopilar los elementos característicos que tiene la acción docente en esta materia, junto con develar aquellos antecedentes de corte explicativo que emanan tanto de las concepciones educativas personales de la dirección de carrera y docentes, como de la cultura institucional de este plantel de educación superior.

La primera parte corresponde a un resumen de lo que fue ese proceso, que abarca desde la formulación del problema de investigación como tal, hasta los resultados que se originaron, pasando por el marco teórico que lo sustenta como objeto de estudio, junto al diseño metodológico que permitió ponerlo en ejecución durante el primer semestre académico del año 2012.

La segunda parte corresponde al planteamiento de lo que es la propuesta de innovación pedagógica desde aquellos ámbitos donde se aprecian falencias educativas, que se pretenden subsanar con el proyecto que acá se expone. Se define lo que se quiere lograr, cuáles son los objetos generales y específicos en que se desglosa el trabajo, junto con el soporte teórico que avala lo sugerido.

Además, en esta parte se ofrece un estudio de factibilidad de realización de este proyecto, que tiene como objetivo justificar el por qué esta propuesta se puede llevar a la práctica. Finaliza este trabajo con las conclusiones, en las cuales el autor ofrece algunas reflexiones de lo que ha sido este trabajo: desde el sentido innovador de lo propuesto, hasta lo que ha sido la globalidad de este proceso.

PARTE

I

Origen

Capítulo 1

Antecedentes de la Investigación

Introducción

La programación de computadores es una habilidad transversal dentro de cualquier área de la Ingeniería, cuyo desarrollo por parte del estudiante requiere de la maduración del pensamiento algorítmico, que se relaciona con la capacidad de determinar cuáles son los pasos que permiten generar una solución para algún problema determinado.

Dentro de ella es posible encontrar una variedad de paradigmas que han ido surgiendo con el correr del tiempo. En este sentido, la orientación a objetos es el que más se demanda en la actualidad por parte del sector productivo, y que además es el que la mayoría de los estudiantes deben conocer y manejar con soltura.

La evidencia empírica que existe para comunicar el extracto de la investigación que se presenta en este capítulo, revela que los docentes a cargo de enseñar programación a nivel superior no siempre cuentan con alguna base de conocimientos sólidos en pedagogía que les permita implementar alguna metodología que facilite la comprensión de esta rama de las Ciencias de la Computación a los estudiantes.

La investigación que se ha desarrollado permite develar dos aspectos. El primero de ellos corresponde a determinar las características principales que tiene la enseñanza de esta disciplina, que se enfoca desde la perspectiva de la planificación, organización y desarrollo de la misma. El segundo apunta a develar aquellos antecedentes explicativos del primer aspecto, en función de las concepciones educativas de la institución y de las creencias pedagógicas personales de los estamentos involucrados en este estudio.

La investigación se ha desarrollado bajo el enfoque metodológico del paradigma hermenéutico, ya que la aproximación a la realidad del objeto de análisis se hará mediante una interpretación del fenómeno de la enseñanza de la programación orientada a objetos, donde la entrevista en profundidad, focus

group, observación metodológica de aula y la revisión documental se constituyeron en los instrumentos que permitieron recoger la información de campo, para luego analizarla de acuerdo a las directrices de la triangulación hermenéutica.[1]

Posterior a eso, se comentarán los resultados principales que se obtuvieron después de haber aplicado los instrumentos de recolección de información cualitativa a los sujetos que conformaron los distintos estamentos de la investigación: director de carrera, docente de aula, y estudiante.

1.1 Planteamiento problemático

1.1.1 Antecedentes

Una de las principales deficiencias que se pueden observar del trabajo con estudiantes de tercer y cuarto año de Ingeniería Civil Informática en la Universidad San Sebastián es la dificultad que exhiben para diseñar e implementar soluciones algorítmicas para problemas de la vida real, ya que las inquietudes que formulan para la realización de tareas que involucran la habilidad de programación apuntan a que no comprenden con claridad cómo estructurar un ordenamiento secuencial de instrucciones para que una máquina pueda generar los resultados esperados.

Esto se puede ver como una consecuencia directa en la cual los estudiantes creen verse enfrentados a una manera de razonar distinta de la que estaban acostumbrados cuando tuvieron los primeros cursos matemáticos de su carrera, como «*Álgebra*» o «*Cálculo*», donde el planteamiento de problemas matemáticos no requería que su resolución se transportara al ámbito de una respuesta computacional. En este último caso, se manejan ciertos códigos que requieren de un tiempo de maduración, y que claramente los estudiantes no tienen

[1] Cisterna, F. (2005). *Categorización y triangulación como procesos de validación del conocimiento en investigación cualitativa*. Revista Theoria: Ciencia, Arte y Humanidades, Universidad del Bío-Bío, Chillán.

completamente asimilados, como el hecho de hacer una formulación abstracta de algún problema que requiera del uso de estructuras algebraicas discretas como grafos o arreglos.

1.1.2 Problema como tal

El problema que se ha detectado a partir de la evidencia empírica que se maneja, es que los profesionales que imparten docencia en alguna institución de educación superior no siempre traen las competencias pedagógicas que demanda esta actividad, lo que en cierta medida explica el por qué los alumnos presentan grandes falencias al momento de tener que plantear una solución algorítmica para algún problema específico.

Es por ello que la investigación que se ha realizado apunta a desarrollar un estudio de las características que configuran la praxis docente en el aula en el ámbito pedagógico de la didáctica, aplicada a la asignatura «*Diseño y Construcción de Software*», que se dicta en el primer semestre del segundo año de la carrera Ingeniería Civil Informática de la Universidad San Sebastián, para estudiar las principales falencias en que incurren los docentes de estos cursos en el marco del desarrollo de la enseñanza en el área de programación basada en el paradigma de la orientación a objetos.

Es por tanto que el ámbito temático sobre el cual se ha estructurado la presente investigación corresponde al de la didáctica de la programación orientada a objetos, que se enmarca dentro del contexto de las Ciencias de la Computación. Específicamente, se trabajará con la asignatura *«Diseño y Construcción de Software»*, que es un curso que busca profundizar en el desarrollo de las habilidades para construir programas computacionales desde la perspectiva de la orientación a objeto por medio del conocimiento y de la operacionalización de los conceptos claves que se revisan en esta área, a través del lenguaje Java.

1.1.3 Justificación

En primer lugar, es importante destacar que la mayoría de los docentes que han hecho clases en asignaturas del área algorítmica no cuentan con una preparación formal en Educación. Por tanto, al no tener estudios mínimos en esta disciplina, varios profesionales que se dediquen a esta actividad no diseñarán estrategias metodológicas de enseñanza que estén orientadas a hacer una adecuada labor de transposición didáctica para que la adquisición de los conocimientos en sus estudiantes se lleve a cabo de manera efectiva. Por lo mismo, la investigación que se ha realizado tiene como foco principal estudiar el ámbito pedagógico de la didáctica, porque interesa conocer y comprender cómo se lleva la acción docente al interior del aula.

La importancia de hacer un estudio orientado hacia la didáctica de la algorítmica se justifica en parte por la necesidad de contar con resultados claros que permitan tomar las decisiones correctas en pro de mejorar la calidad de la enseñanza en esta área. Además, indagar sobre las características del ejercicio didáctico en esta asignatura que se imparte en el segundo año de la carrera, permitió generar un diagnóstico que será la base sobre la cual se ha construido la propuesta de intervención pedagógica que se levanta a contar de la segunda parte de este trabajo. Otra arista sobre la cual se puede explicar la importancia del estudio, es que «*Diseño y Construcción de Software*», es uno de los ramos claves que permiten cimentar las bases del conocimiento algorítmico en los estudiantes.

La capacidad de desarrollar nociones de pensamiento algorítmico va en directo beneficio de la preparación académica del estudiante de Ingeniería Civil Informática de la Universidad San Sebastián. Como se verá, la influencia que este dominio ejerce es transversal en la formación de ellos, ya que sus principios y normas aparecen de una u otra manera en cada asignatura que compone el programa de estudios.

Sin pretender entrar en mayores honduras teóricas, es bueno dejar establecido de antemano que todo lo que se ve en las asignaturas propias de la especialización en Ingeniería Civil Informática tiene que ver con los objetos discretos, es decir, aquellos que tienen una naturaleza finita, que se pueden contar. Y es así, puesto que el instrumento que sirve para implementar los proyectos de Ingeniería es el computador, el cual es de naturaleza finita. Ralph Grimaldi identifica algunas preguntas clásicas que se hacen a diario en esta área de la carrera: *«¿Hay alguna conexión entre dos computadores de una red?*»; o bien, dada una tecnología de cableado: *«¿cuál es el diseño de red más económico para cierta empresa?*», «*¿cómo puede ordenarse una lista de números enteros (o de tareas de una cadena) en forma creciente?», «¿cuántas palabras clave válidas hay para acceder a un sistema?», o «¿cómo se puede codificar de forma adecuada y segura un mensaje?».* En suma, de acuerdo a lo expuesto por el autor, esta rama de las Ciencias de la Computación contribuye al desarrollo de ciertas competencias fundamentales para un ingeniero: capacidad de formalizar, de razonar rigurosamente y de representar de manera abstracta algunos conceptos e ideas que surgen a partir de especificaciones de problemas reales.[2]

Todo lo anterior se complementa con el hecho que en la página web de la Universidad San Sebastián se señala que el egresado de Ingeniería en Informática «*será capaz de acreditar una continua autogestión del conocimiento, lo que le permitirá diseñar, construir y administrar proyectos de innovación tecnológica y de optimización de procesos informáticos con autonomía y creatividad*». Una primera mirada a esta información permite vislumbrar un aspecto en que la rama algorítmica desempeña un rol clave, por cuanto la construcción optimizada de los sistemas de información que requieren los distintos tipos de empresa están condicionados por la eficiencia con que se hayan diseñado las rutinas y subrutinas algorítmicas sobre las cuales se articulan los proyectos de programación, aparte de los elementos propios del modelado de las situaciones de la vida real que se complementan con la formación que tengan en Ingeniería de Software.

[2] Grimaldi, R. (1997). *Matemáticas Discreta y Combinatoria: Una introducción con aplicaciones.* México. Editorial Addison-Wesley Iberoamérica.

Existe una amplia gama de campos de la Ingeniería Informática en los cuales las Ciencias de la Computación ejercen influencia en cuanto a los sustentos matemáticos que los afirman, y cuyos resultados encuentran una aplicación directa en el terreno de los algoritmos y programación. Dentro de los que están considerados en el plan de estudios que contempla la Universidad San Sebastián para sus estudiantes de Ingeniería Civil Informática están: Teoría de Grafos, Teoría de Autómatas, Teoría de la Computación, Análisis de Algoritmos, Estructuras de Datos, Lenguajes de Programación, Bases de Datos e Inteligencia Artificial. Todos ellos se estudian en diferentes asignaturas, tales como: «*Electivos de Especialidad I y II», «Programación Orientada a Objetos I y II», «Arquitectura de Computadores y Sistemas Operativos», «Transmisión y Comunicación de Datos», «Estructura de Datos», «Base de Datos», «Inteligencia de Negocios»*, entre otros.

En el ámbito algorítmico se ve todo lo relacionado con el uso de dialectos artificiales que tienen como propósito crear aplicaciones computacionales para algún propósito específico, usando para ello ciertas técnicas articuladas a través de la lógica de la programación. Esto le permite al alumno desarrollar líneas de pensamiento sistémico, lógico y analítico a través de la resolución de problemas complejos.

Por otra parte, esta es una de las principales fuentes de empleo que tienen los Ingenieros en Informática, por lo que la adquisición de una base teórica sólida en esta materia es crucial. Las estructuras de datos van de la mano con los lenguajes de programación, porque corresponden a una serie de técnicas que tienen como objetivo organizar un conjunto de datos determinado para facilitar su manipulación, definiendo según Behrouz Forouzan una interrelación entre los mismos que se implementan directamente en un algoritmo. La importancia que tiene para el estudiante aprender el correcto manejo de estas herramientas radica en que están íntimamente relacionados con la calidad de las aplicaciones

computacionales que desarrollen, en términos de espacio en memoria principal y secundaria, y del tiempo de ejecución de los mismos.[3]

Las bases de datos corresponden a estructuras de datos dinámicas que permite almacenar grandes volúmenes de información en diferentes tablas para su manipulación posterior a partir de la indexación que se hace sobre cada registro de ellas. La influencia que ejercen las Ciencias de la Computación en este campo viene dada por el modelamiento conceptual de ella, ya que permite determinar la lógica que subyace a las relaciones definidas entre los diferentes actores que forman parte de la definición del problema, y porque además define de manera formal todas las operaciones que se pueden hacer a partir de ellas, como por ejemplo: inserción, eliminación, actualización, selección, proyección, juntura, unión, intersección y diferencia entre distintos registros. Todas ellas vienen configuradas dentro del álgebra relacional, donde la Lógica de Primer Orden es el lenguaje que se emplea para hacer las consultas. Posteriormente, en la vida laboral del Ingeniero Civil Informático, estos conocimientos tendrá que aplicarlos de manera rigurosa en pro de mejorar la calidad de los procesos internos de la institución donde esté trabajando. Pero no sólo deberá saber plantear algún modelo conceptual para articular la lógica de una base de datos, sino que tendrá que ser capaz de hacer la codificación apropiada para operacionalizar mejor algún sistema de información.

La Teoría de Grafos juega un papel importante en la fundamentación matemática de las Ciencias de la Computación, ya que a partir de ahí se pueden modelar y resolver problemas clásicos que surgen en el quehacer de un Ingeniero Informático, como por ejemplo la configuración de una red de información, determinar el camino óptimo para ir de un lugar a otro dentro de una grilla, o también para hacer los modelos conceptuales que permiten concebir algún sistema de información soportado por un Sistema Gestor de Base de Datos (SGBD). Otros problemas que se pueden transportar al ámbito de la vida diaria y laboral que se abordan mediante el uso de estas estructuras discretas son:

[3] Forouzan, B. (2003). Introducción a la Ciencia de la Computación: de la manipulación de datos a la Teoría de la Computación. Cupertino, California. Editorial Thompson.

asignación de recursos, determinar la planaridad de un grafo, hacer abstracciones sobre redes de computadores mediante grafos condensados, o decidir si acaso admite un «*ciclo hamiltoniano*»[4], que es un problema que tiene como aplicación directa determinar la mejor ruta que debiera seguir una persona para minimizar el costo total de desplazamiento a través de una red interconectada de ciudades. Todos estos conceptos se traducen en expresiones algorítmicas que requieren ser implementadas por computador para apoyar la resolución de problemas de la vida real.

La Teoría de Autómatas es aquella que estudia las máquinas abstractas vistas como una estructura algebraica, y los problemas que éstas son capaces de resolver. Arno Formella[5] dice que este campo está relacionado con la teoría de los lenguajes formales, ya que los autómatas se clasifican de acuerdo a la clase de lenguajes formales que son capaces de reconocer, de acuerdo a la jerarquía de Chomsky. Además, los autómatas finitos y las expresiones regulares aparecen de una u otra forma de manera reiterada en muchas áreas de la Ingeniería en Informática como el «*metalenguaje para describir las expresiones que son procesadas por una fase dentro de un compilador*». La Teoría de la Computación, en tanto, explora los límites de los computadores modernos para solucionar problemas de la vida cotidiana empleando algoritmos. Michael Sipser establece que el campo de Análisis de Algoritmos provee las técnicas para determinar cuándo una solución computacional es eficiente en cuanto al espacio requerido y al tiempo que demande su ejecución, y es ahí donde se hace necesario que el estudiante cuente con las habilidades necesarias para hacer demostraciones que permitan concluir la eficiencia o no eficiencia de un algoritmo propuesto.[6]

La Inteligencia Artificial es una de las áreas donde las aplicaciones algorítmicas tienen plena cabida. Stuart Russel es uno de los autores más connotados en este ámbito, y de acuerdo a su visión se trata de un área que «*estudia la automatización del comportamiento inteligente*». Es posible que con

[4] En términos simples, si acaso es posible recorrer una red, pasando por todos sus vértices exactamente una sola vez, salvo en el caso del vértice inicial y terminal que deben ser iguales.
[5] Formella, A. (2010). *Teoría de Autómatas y Lenguajes Formales*. Vigo. Universidad de Vigo.
[6] Sipser, M. (2005). *Introduction to the Theory of Computation*. Massachusetts. Editorial MIT Press.

las técnicas que se exploran dentro de estas asignaturas los estudiantes puedan ser capaces de crear sistemas basados en agentes inteligentes que permitan, por ejemplo: apoyar la gestión de los médicos para hacer diagnósticos, supervisar la condición de los pacientes, administrar tratamientos y preparar estudios estadísticos.[7] En el ámbito doméstico, los estudiantes pueden implementar plataformas que permitan brindar asesoría acerca de dietas, compras, supervisión y gestión de consumo energético y seguridad del hogar. Por supuesto, las técnicas provistas en este sector se pueden transportar a programas diseñados para implementar sistemas de agentes inteligentes, como robots.

En Educación, se pueden formular proyectos tendientes a mejorar la calidad de la enseñanza por medio de Sistemas Hipermediales Adaptativos — SHA —, que combinan elementos hipermediales[8], estilos de aprendizaje y adaptabilidad a la realidad de cada alumno. Para alcanzar esta última característica, se emplean técnicas propias del campo de la Inteligencia Artificial — como por ejemplo Planificadores, Redes Neuronales y Redes Bayesianas —, que actuando en conjunto permitan diagnosticar los niveles de conocimientos y estructurar de forma automática la planificación didáctica y las evaluaciones, en función del perfil del estudiante, de acuerdo a lo que ha estado haciendo Pedro Salcedo en el campo de los SHA. En cualquiera de las áreas que revisamos en este apartado, el uso de algoritmos y programación es fundamental, porque está orientado principalmente a lo que será el ejercicio profesional que tenga el estudiante que la institución está formando.[9]

---

[7] Russell, S. J. y Norvig, P. (2004).*Inteligencia Artificial: Un enfoque moderno.* Madrid. Editorial Pearson Educación.

[8] Es el término con que se designa al conjunto de métodos o procedimientos para escribir, diseñar, o componer contenidos que tengan texto, video, audio, mapas u otros medios, y que además tenga la posibilidad de interactuar con los usuarios. El término hipermedia toma su nombre de la suma de hipertexto y multimedia, una red hipertextual en la que se incluye no sólo texto, sino también otros medios: imágenes, audio, video, etc. (multimedia).

[9] Salcedo, P. (2004). *Un curso de lenguajes formales e introducción a la teoría de compiladores que usa una plataforma basada en el conocimiento para educación a distancia.* Concepción, Chile. Revista de Ingeniería Informática.

1.1.4 Preguntas de investigación

Como se comentó en la introducción, la investigación que se ha realizado es de corte descriptivo-explicativa, y por lo mismo se han definido preguntas que van en esa línea.

Pregunta de carácter descriptivo: ¿Cuáles son los principales aspectos que permiten caracterizar la praxis pedagógica desde el ámbito de la didáctica en los docentes de la carrera de Ingeniería Civil Informática de la Universidad San Sebastián que imparten «*Diseño y Construcción de Software*»?

Pregunta de corte explicativo: ¿Qué elementos se pueden revelar como antecedentes explicativos de la praxis pedagógica ejecutada por los docentes que imparten «*Diseño y Construcción de Software*» en relación a las acciones didácticas que ejecutan al interior del aula?

1.1.5 Objetivos de la investigación

Los objetivos generales que se han formulado para el estudio realizado fueron los siguientes:

Objetivo general número 1. Describir cuáles son los elementos principales que caracterizan la praxis pedagógica en el ámbito de la didáctica de los docentes de Ingeniería Civil Informática que conducen la asignatura del área de lenguajes de programación «*Diseño y Construcción de Software*» para los estudiantes segundo año de la carrera de Ingeniería Civil Informática de la Universidad San Sebastián. Esto conlleva los siguientes objetivos específicos:

> **Objetivo específico 1.1.** Describir la praxis pedagógica de los docentes que imparten la asignatura «*Diseño y Construcción de Software*» desde aquellos aspectos relacionados con las acciones de planificación de la enseñanza en este curso.

Objetivo específico 1.2. Describir las características del ejercicio que los docentes a cargo de dictar «*Diseño y Construcción de Software*» con respecto a las acciones que emprenden en cuanto al desarrollo de la enseñanza en estas asignaturas.

Objetivo general número 2. Evidenciar antecedentes explicativos de la praxis pedagógica de los docentes que imparten la asignatura antes mencionada, de acuerdo al desarrollo de las acciones didácticas. Los objetivos específicos que se definieron a partir de esto, son:

Objetivo específico 2.1. Describir antecedentes explicativos de la praxis pedagógica de docentes que imparten «*Diseño y Construcción de Software*» en el desarrollo de sus acciones didácticas desde el ámbito de la cultura institucional de la Universidad San Sebastián.

Objetivo específico 2.2. Describir aquellos antecedentes explicativos ligados a la praxis pedagógica de los docentes de la asignatura «*Diseño y Construcción de Software*» en el desarrollo de sus acciones didácticas desde el punto de vista de sus creencias o concepciones pedagógicas personales.

1.1.6 Categorías y subcategorías apriorísticas

Categoría A: Planificación y organización de la enseñanza

Corresponde a las actividades sistemáticas que llevan a cabo los docentes para definir la secuencia de pasos que conduzcan a la enseñanza a la meta final —lograr aprendizaje de los alumnos —, como una forma de concretar las directrices curriculares que indica la Universidad San Sebastián. Esto incluye además el modo en que se organiza el desarrollo de dichas actividades en el espacio y tiempo real del trabajo de aula en la asignatura «*Diseño y Construcción de Software*», destinadas a estudiantes de segundo año de Ingeniería Civil

Informática de la Universidad San Sebastián. Las subcategorías que se desprenden son las que se explican a continuación.

Subcategoría A.1: Planificación curricular. La planificación curricular corresponde al conjunto de acciones de tipo prescriptivo que realizan los docentes para un semestre o un año, que tiene como meta organizar los contenidos de la enseñanza y garantizar el total cumplimiento de la cobertura curricular definidas en el programa de estudio de la asignatura «*Diseño y Construcción de Software*», donde se debe velar por el cumplimiento de cinco principios fundamentales: (1) *principio de secuencia vertical*, que establece que los contenidos tienen que ir aumentando en dificultad a medida que se revisan; (2) *principio de continuidad*, que significa que la secuencia de las unidades didácticas que componen alguna materia deban seguir un ordenamiento lógico; (3) *principio de integración*, que apela al diálogo interdisciplinar con los contenidos presentes en otras asignaturas; (4) *principio de congruencia horizontal*, cuyo objetivo es asegurar que los contenidos aparezcan debidamente desglosados en los objetivos, y que exista una coherencia respecto a los recursos didácticos y metodología que se vaya a emplear, para que se genere una globalidad unificada; y (5) *principio de comunicabilidad*, que busca que los aprendizajes esperados y los objetivos de enseñanza y aprendizaje sean claramente conocidos por los alumnos.[10]

Subcategoría A.2: Planificación didáctica. La planificación didáctica también corresponde a una secuencia de acciones que desarrolla el docente, con la diferencia que ésta se hace clase a clase, que incluye elementos como: aprendizajes esperados y las actividades de inicio, desarrollo y cierre de la clase. Al inicio se hace una activación de los aprendizajes previos, luego se presentan los aprendizajes esperados para la sesión, y se termina con una síntesis que permita retroalimentar, hacer evaluación formativa y reforzar en líneas generales lo que ya se ha visto.[11]

[10] Cisterna, F. (2007). *Currículum Educacional. Concepciones teóricas y desarrollo en el aula.* Chillán. Ediciones Universidad del Bío-Bío. Páginas 18 a 22.
[11] Ibíd.

Categoría B: Desarrollo de la enseñanza

Se refiere al conjunto de acciones que emprenden los docentes al interior del aula para ejecutar las acciones que vienen definidas en las planificaciones curricular y didáctica, que incluyen: estrategias metodológicas de enseñanza, interacción pedagógica, recursos didácticos, organización y uso del espacio y tiempo de la clase, y clima de aula. Las subcategorías que se desprenden son las que se explican a continuación.

Subcategoría B.1: Estrategias metodológicas de enseñanza. Corresponde a todas las actividades que el docente realiza para revisar los contenidos disciplinarios de «*Diseño y Construcción de Software*», a fin de realizar una efectiva labor de transposición didáctica en el aula, que se define como el proceso orientado a transformar un saber sabio en un saber enseñable y aprehensible de manera significativa en los estudiantes.[12]

Subcategoría B.2: Interacción pedagógica. Se refiere a la relación que se establece al interior del espacio áulico entre el docente y los alumnos, que se expresa en el modo en el cual el profesor se dirija a los estudiantes, los haga participar, les sugiera preguntas para que avancen en sus procesos de aprendizaje o les brinde algún tipo de retroalimentación, que se refleja por ejemplo en la calidad de las explicaciones que brinda, o en el tipo de preguntas que el docente haga durante la clase para despertar ciertos aprendizajes en sus alumnos.

Subcategoría B.3: Recursos didácticos. Corresponden a los materiales que los docentes que dictan «*Diseño y Construcción de Software*» incorporan y utilizan de manera efectiva con la finalidad de facilitar los procesos que conduzcan a aprendizajes significativos y la consecución de habilidades de diseño e implementación de soluciones algorítmicas por parte de los estudiantes.[13]

[12] Ibíd.
[13] Ibíd.

Subcategoría B.4: Organización del tiempo y del espacio. Se refiere a la forma en cómo el docente efectivamente emplea el tiempo que tiene para estar activo durante la clase, el cual abarca eventos como la hora de llegada del docente y de los estudiantes a la sala, pasando por el inicio, desarrollo y cierre de la sesión. La organización del espacio corresponde a la forma en cómo el profesor distribuye a los estudiantes al interior del aula para que ellos puedan aprovechar de mejor manera la clase.

Subcategoría B.5. Clima de aula. Está relacionado con la garantía de mantener un ambiente propicio para el aprendizaje de los estudiantes, que se manifiesta en las siguientes maneras: manejo del grupo curso por parte del docente para recrear un ambiente que propicie relaciones de aceptación, equidad, confianza, solidaridad y respeto; generación de altas expectativas en relación a las posibilidades de aprendizaje de los alumnos; establecimiento de normas de convivencia que sean propias de una clase; y en la intensidad y ritmo que le imprima a la clase.[14]

Categoría C: Cultura institucional

Está relacionada con la cultura institucional de la Universidad San Sebastián, que se expresa en su Proyecto Educativo Institucional — PEI — y que conforma un referente esencial de tipo normativo que orienta el trabajo y el desempeño de los profesores que componen su equipo docente, tanto para los de planta como para los categorizados como adjuntos. De esta manera, el PEI es un instrumento que orienta todos los procesos que ocurren en un establecimiento educacional, clarifica a los actores las metas de mejoramiento, da sentido y racionalidad a la gestión para el mediano o largo plazo, permite la toma de decisiones pedagógicas y curriculares, articula los proyectos y acciones innovadoras en torno al aprendizaje y la formación de los alumnos, y ordena las grandes tareas en torno a

[14] Ideas extraídas del segundo dominio definido en el *Marco para la buena enseñanza*, redactado por el Ministerio de Educación de Chile (2003).

objetivos compartidos.[15] Las subcategorías que se desprenden son las que se explican a continuación.

Subcategoría C.1: Cultura institucional en materias de tipo general. Se refiere a las directrices que se desprenden directamente de la cultura institucional de la Universidad San Sebastián en asuntos pedagógicos de interés general, como por ejemplo: su orientación filosófica, religiosa o la relación que exista entre los miembros de la comunidad educativa. En conjunto determinan directrices importantes para la orientación del trabajo docente en el aula, con especial énfasis en el curso «*Diseño y Construcción de Software*».[16]

Subcategoría C.2: Cultura institucional en materias específicas de tipo pedagógicas. Se refiere a la cultura institucional que hace explícita en materias de tipo específicamente pedagógicas y que tienen que ver con directrices establecidas en la Universidad San Sebastián, como el tipo de planificación didáctica que debe efectuarse, los objetivos educacionales que se hayan trazado como prioritarios, el trabajo colaborativo que se propicia entre el cuerpo docente, la importancia, variedad y disponibilidad de los recursos didácticos que empleen al interior del aula, entre otros.

Categoría D. Creencias pedagógicas personales

Se relaciona con las creencias personales que poseen los docentes que están a cargo del curso «*Diseño y Construcción de Software*» sobre la función y rol de la educación en general y sobre el papel que el docente desempeña en su calidad de agente generador de aprendizajes en los estudiantes.

Subcategoría D.1: Creencias pedagógicas personales de tipo general en cuanto al rol de la educación en general. Se refiere a las creencias personales expresadas por los docentes que dictan «*Diseño y Construcción de*

[15] Villarroel, S. (2002). *Proyecto Educativo Institucional: Marco legal y Estructura básica.* Gobierno de Chile. Ministerio de Educación.
[16] Cisterna, F. (2011). *Investigación Educacional: Paradigmas, Planteamiento Problemático, Marco Teórico, Procedimientos de Recolección y Análisis de la Información*. Chillán. Ediciones Universidad del Bío-Bío.

Software» en cuanto al rol que la educación en general posee para la sociedad y para las personas.

Subcategoría D.2: Creencias pedagógicas personales en cuanto al rol específico del docente como agente pedagógico. Se refiere a las creencias expresadas por los docentes asignados a «*Diseño y Construcción de Software*» en cuanto al rol que debe desempeñar como sujeto mediador de los aprendizajes de los estudiantes y que constituyen los referentes personales que orientan de manera fundamental su trabajo de aula.

1.2 Marco teórico de la investigación

Una de las acepciones que trae el Diccionario de Real Academia de la Lengua Española con respecto a la palabra problema es la siguiente: «*planteamiento de una situación cuya respuesta desconocida debe obtenerse a través de métodos científicos*». Con miras a lograr esa respuesta, un problema se puede definir como una situación en la cual se trata de alcanzar una meta y para lograrlo se deben hallar y utilizar unos medios y unas estrategias. Juan Carlos López identifica algunos elementos que tienen en común: (1) estado inicial; (2) meta, que apunta a lo que se quiere lograr; (3) conjunto de recursos y reglas que determinan lo que está permitido hacer y/o utilizar; y (4) un dominio, el estado actual de conocimientos, habilidades y energía de quien va a resolverlo. Casi todos los problemas requieren en quien los resuelva que sea capaz de dividirlo en submetas que, cuando sean dominadas, lleven a alcanzar el objetivo deseado.[17]

Cada ámbito disciplinario determina ciertas estrategias específicas para resolver problemas que sean propios de su campo de estudio. Por ejemplo: resolver problemas matemáticos implica utilizar estrategias propias de esta ciencia, como uso de sistemas de ecuaciones, ecuaciones diferenciales, métodos

[17] López, J. C. (2009). *Algoritmos y Programación: Guía para Docentes.* Quito. Fundación Gabriel Piedrahita Uribe.

numéricos, etc. Sin embargo, algunos psicólogos opinan que es posible utilizar con éxito estrategias generales que sean útiles para resolver problemas en muchas áreas. A través del tiempo, la humanidad ha utilizado diversas estrategias generales para resolver problemas, entre los cuales podemos destacar algunos como el de ensayo y error y los algoritmos. El primero consiste en actuar hasta que algo funcione, aunque puede demandar bastante tiempo hasta alcanzar la solución deseada, o puede que incluso jamás se pueda llegar. Su uso es apropiado cuando el conjunto solución abarca pocos elementos y es posible chequearlos todos, pudiendo empezar el proceso de iteración desde aquella solución que ofrezca la mayor probabilidad de poder resolver el problema. En cambio, los algoritmos se orientan a aplicar adecuadamente una serie de pasos detallados que aseguren una solución correcta del mismo.

Luego de analizar detalladamente el problema hasta entenderlo completamente, se procede a diseñar un algoritmo que lo resuelva por medio de pasos sucesivos y organizados en secuencia lógica. La idea de algoritmo establece que se trata de un conjunto de procedimientos y reglas que se deben seguir en un orden determinado para poder brindarle una solución.[18]

En el ámbito de la computación, los algoritmos son una herramienta que permite describir claramente un conjunto finito de instrucciones, ordenadas secuencialmente y libres de ambigüedad, que debe llevar a cabo un computador para lograr un resultado que sea de utilidad para el usuario final.[19] En términos generales, un algoritmo debe ser: (1) realizable — que debe terminar después de una cantidad finita de pasos —; (2) comprensible — ser claro lo que hace, de forma que quien ejecute los pasos descritos en el cuerpo del mismo tenga claridad absoluta sobre cómo y cuándo hacerlo —; y (3) y preciso, es decir, que ante los

[18] La misión de un Ingeniero Civil Informático no es sólo ser eficaz en cuanto al planteamiento del algoritmo que orientará la resolución de un problema, sino que además debe ser hecha de manera eficiente tanto en el tiempo que demande su ejecución como en el espacio físico que requiera para almacenar los valores temporales que arrojen sus procesos internos. Durante la asignatura que se estudia, este punto no se ve en profundidad, ya que es necesario que los estudiantes adquieran mayor familiaridad con las ideas del razonamiento algorítmico antes de aplicar alguna estrategia que les permita mejorar sus implementaciones, y profundizar aún más en matemática discreta, lo que recién se empieza a estudiar en el curso «*Especialidad I*».

[19] López, J. C. (2009). *Algoritmos y Programación*… Página 21.

mismos datos de entrada, genere siempre la misma salida, acorde con la definición de determinismo que proporciona Rosa Guerequeta.[20]

Por otro lado, uno de los productos exigidos a los profesionales del área Informática es la generación de un software o aplicación que sirva para resolver algún problema. Pues bien, para llegar a esa solución, lo que se hace es partir haciendo el diseño de la solución — expresado en un algoritmo en pseudocódigo o en diagrama de flujo — que luego se debe implementar. La herramienta que permite hacer la mediación entre el diseño de la solución y el producto final es lo que se denomina «*lenguaje de programación*». Se trata de un idioma artificial que se ha creado para traducir las ideas contenidas en un algoritmo al lenguaje propio de la máquina donde se va a desarrollar el software. El «*código fuente*» es un conjunto de líneas de texto que corresponde al conjunto de instrucciones que debe seguir la máquina para poder ejecutar el programa, constituyéndose de esta manera en el documento oficial que describe por completo su funcionamiento. En el contexto de este trabajo, se entenderá que la palabra «*programación*» alude al proceso mediante el cual se escribe, prueba y depuran las instrucciones contenidas en el código fuente.

1.2.1 Enseñanza de la programación

Siendo la programación una tarea que implica un proceso mental — generalmente complejo y creativo —, el desafío de enseñarla no es menor, puesto que aprender a crear aplicaciones computacionales es un objetivo clave en la gran mayoría de los cursos introductorios de carreras del área de la Informática, como es «Diseño y Construcción de Software».

Desde el punto de vista educativo, la programación de computadores posibilita no sólo activar una amplia variedad de estilos de aprendizaje, sino de desarrollar el pensamiento algorítmico, que se refiere al desarrollo y uso de

[20] Guerequeta, R. (2002). *Técnicas de diseño de algoritmos.* Málaga. Universidad de Málaga.

algoritmos que puedan ayudar a resolver un tipo específico de problema o a realizar un tipo específico de tarea.

Esto supone otros desafíos intermedios a los estudiantes, como son: decidir sobre la naturaleza del problema, para llegar a la mejor estrategia de resolución, seleccionar alguna representación que sea apropiada para alcanzar la respuesta; y realizar de manera permanente el ejercicio de metacognición para que ellos reflexionen sobre sus propios pensamientos y en los esquemas que están empleando para encontrar la solución. López también indica que la creatividad es otra habilidad de pensamiento que también se puede ayudar a desarrollar en cursos como los que se van a tomar en cuenta para esta investigación.

En el último tiempo, la creatividad forma parte de las prioridades de los sistemas educativos en varios países, junto a otras habilidades de pensamiento de orden superior. Una de las razones que permite explicar el por qué este tema se ha convertido en algo tan importante, es por el alto impacto que tiene en la generación de riqueza por parte de las empresas de la Sociedad del Conocimiento, ya que los reconocimientos que recaen en los profesionales que trabajan en ellas los miden en base a la inteligencia y a la creatividad con que desarrollen sus actividades. López plantea que uno de los grandes desafíos que recae sobre los sistemas educativos — no sólo a nivel de educación superior — es lograr que se generen las estrategias adecuadas para que los estudiantes se desarrollen como pensadores creativos, donde el computador y los lenguajes de programación pasen a ser los instrumentos que le permitan estructurar trabajos que den cuenta de esta habilidad.[21]

Jorge Villalobos complementa lo que López describe al introducir una lista de los principales factores que contribuyen a explicar la dificultad que le surge a los docentes tener que enseñar a programar, que las resume en cuatro puntos: (1)

[21] Ibíd. Página 22.

aprendizaje por imitación; (2) aproximación de abajo hacia arriba; (3) desequilibrio de ejes temáticos; y (4) manejo de la motivación y de la frustración.[22] [23]

Aprendizaje por imitación

Un curso de enseñanza de la programación está orientado a la generación de habilidades, más que a la simple exposición de un conjunto de conceptos. Es más, la cantidad de teoría que podemos encontrar detrás de la programación se sorprende del poco número de conceptos distintos y de su simplicidad. El problema radica en cómo despertar en los estudiantes la habilidad para manipular a conveniencia esas herramientas, y generar productos de software de calidad.

El esquema tradicional de clases donde el profesor revisa los conceptos más importantes, responde a las preguntas que puedan surgir de dichas definiciones, y luego comienza a hacer ejercicios en el pizarrón se basa en la idea de que el estudiante sea capaz de detectar patrones en los problemas planteados y los logre asociar con las técnicas que usa el profesor cuando desarrolla un ejemplo en el aula, y que luego el estudiante sea capaz de hacer la generalización adecuada en su cabeza, para poder aplicar esa asociación patrón- técnica para resolver otros problemas. Este enfoque de «*aprendizaje por imitación*» se basa en la hipótesis de que el alumno sea capaz de generar las habilidades necesarias para tratar de imitar lo que el profesor hace.

Dentro de los inconvenientes que esta forma de aprender tiene en el alumno es que no se genera el vocabulario necesario para hablar de la manera de construir un programa, ya que el binomio patrón-técnica sólo se manejan a un nivel intuitivo y jamás se verifica su validez, y sólo se genera vocabulario para hablar de las estructuras sintácticas de los lenguajes de programación, y esto dificulta en gran medida el apoyo que el profesor puede darle al estudiante en caso de que tenga dificultades de construcción de software más profundas.

[22] Estas razones son independientes del paradigma de programación que se desee enseñar.
[23] Villalobos, J. (2006). *El reto de diseñar un primer curso de programación de computadores.* Bogotá: Departamento de Ingeniería de Sistemas Bogotá, Universidad de los Andes.

Vale decir, que ver la forma en cómo el docente resuelve un problema o conformarse con leer el desarrollo de un ejercicio en un libro de texto no garantiza la adecuada generación de habilidades en el estudiante. Por otra parte, al «*no existir una receta mágica para enseñar a diseñar un programa*», deja dando vueltas la sensación que el proceso de aprendizaje deba quedar en manos de la inspiración y del ingenio del alumno, lo que puede generar situaciones de ansiedad en ellos al no saber si ese «*chispazo*» que se gatilló en sus cerebros sea suficiente para resolver los problemas que el profesor le pueda plantear en alguna evaluación.

Aproximación de abajo hacia arriba

La aproximación de abajo hacia arriba es una manera bastante común de organizar los contenidos de los cursos de programación, y la asignatura «*Diseño y Construcción de Software*» no escapa a ello. Se inicia la enseñanza a través del análisis de los problemas a resolver, y continúa con la introducción de los términos de base, para luego dar paso a las estructuras de control, ciclos iterativos, funciones, pasaje de parámetros, vectores, matrices y estructuras de datos avanzadas como pilas, filas, árboles, etc., lo que va en concordancia con los principios de secuencialidad vertical y de continuidad que debe satisfacer una planificación curricular. Esto provoca que al final del curso el estudiante sea capaz de construir un programa completo, y pocas veces se alcance a estudiar el tema de las interfaces o de pruebas.

Entonces, desde la perspectiva del alumno se puede decir — según Villalobos — que él no tiene una visión clara de la razón por la cual se introduce un concepto, porque no le ve una necesidad real, dificultando su proceso de aprendizaje. Además, muchos de los temas que se ven resultan artificiales debido al tamaño de los problemas sobre los cuales se puede trabajar, generando la idea que hay etapas dentro del proceso de programación que son una auténtica pérdida de tiempo, cuando en realidad no lo sea, como en el caso de creer que el diseño de la solución tarde más tiempo que en su implementación y prueba. Por último, en un plazo de no más de dos meses de estudio se debe cubrir una

extensa cantidad de contenidos disciplinarios antes de que el alumno caiga en la cuenta que se puede hacer algo interesante con lo que ha estado aprendiendo, que vaya más allá de problemas aparentemente sin contexto alguno, como contar la cantidad de elementos que tiene un vector o insertar un elemento al final de una lista doblemente encadenada. Eso puede generar que el estudiante caiga en la desesperación que aún no vaya a ser capaz de desarrollar aplicaciones comerciales.

Desequilibrio de ejes temáticos.

El desequilibrio de los ejes temáticos está asociado porque en la tarea de la enseñanza de la programación aparecen vinculados una serie de dominios de conocimiento, como por ejemplo: modelamiento de una realidad, especificación de un problema, lenguajes de programación, uso de herramientas computacionales, algorítmica, metodología de programación, proceso de construcción de un programa (análisis, diseño, pruebas), etc. Un curso de programación puede estar estructurado siguiendo uno o varios de estos ejes temáticos. Los cursos típicos de programación están guiados por el eje asociado con el lenguaje, y los demás sólo aparecen puntualmente en algunas partes del curso, lo que hace que la asignatura se base en un recorrido de las estructuras sintácticas del lenguaje.

Todos los demás aspectos que hacen parte de la labor de programar, se ven de manera lateral, o sencillamente se ignoran. Esta manera de estructurar el curso tiene como consecuencia que al estudiante no se le da una visión real de lo que es programar. De acuerdo a la visión de Villalobos, el estudiante le confiere más importancia a los elementos del lenguaje de programación que al proceso de construir un programa, lo que se puede traducir que en las sesiones de clase se generen preguntas en torno a particularidades de alguna clase de Java que a la manera en cómo se especifique un problema que vaya a ser resulto mediante la óptica de la orientación a objetos. El estudiante puede llegar a menospreciar algunos de los elementos que el profesor intenta introducir de manera lateral al curso, y eso dificulta el proceso de enseñanza porque termina menospreciando las habilidades que haya generado en los demás ejes.

Manejo de la motivación y de la frustración.

El último factor que permite explicar la dificultad para enseñar a programar tiene que ver con la motivación y la frustración. La motivación se ve afectada por una parte en la idea que si no ven algún beneficio concreto e inmediato que les puedan reportar los contenidos vistos en la asignatura, les haga perder paulatinamente el interés por la carrera, considerando que estos cursos, al ser de los dos primeros años, pueden condicionar la permanencia de un alumno en la universidad. Por otro lado, el nivel de dificultad que pueda tener la asignatura puede llegar a nublar sus expectativas al darse cuenta que la realidad contrasta bastante con la idea que el proceso podía resultar más sencillo.

Los problemas de frustración están asociados con la parte práctica de la programación porque, a pesar de entender la teoría, a pesar de poder escribir los algoritmos que el profesor pide en clase y a pesar de todo el tiempo que el estudiante dedique a escribir un programa, el éxito da la impresión de depender de un factor extra que nadie controla, que muchas veces se puede atribuir a la «*suerte*», ya que es imposible de predecir cuánto es lo que uno pueda llegar a demorarse en resolver un problema mediante una aplicación computacional. Esto genera una brecha entre lo que el alumno ve en la clase y lo que encuentra en el monitor de su computador al momento de estar programando.

1.3 Metodología

El paradigma científico sobre el cual se estructura la investigación es el hermenéutico, ya que la aproximación a la realidad se realizó a través de una interpretación del fenómeno de la enseñanza de la programación orientada a objeto en la Universidad San Sebastián, aplicada a la asignatura «*Diseño y Construcción de Software*». En ningún caso se buscará hacer una predicción o una generalización, sino más bien una comprensión en profundidad del objeto de estudio, dado el carácter naturalista y fenomenológico que posee.

Además, la investigación realizada es de tipo cualitativa, de finalidad básica, con alcance temporal longitudinal-prospectiva, de profundidad descriptiva-explicativa, de fuentes primarias, naturaleza empírica, y de tipo evaluativa en cuanto a los estudios a los que dé lugar. Su finalidad es de tipo básica, ya que busca una mejor comprensión de los fenómenos que generan los hechos que configuran las actividades de preparación y desarrollo de la enseñanza en «*Diseño y Construcción de Software*».[24]

En cuanto a su profundidad, se puede comentar es de tipo descriptivo-explicativa, ya que se busca lograr una mejor comprensión de las actividades que desarrollan los docentes que imparten «*Diseño y Construcción de Software*» en el ejercicio de su quehacer didáctico tanto dentro del aula — expresado en la parte de desarrollo de la enseñanza — como fuera de ella — por medio de la planificación de la misma —, así como también poder hacer una descripción de aquellos elementos que permitan develar el por qué ejercen de cierta manera su praxis pedagógica, mediante una indagación que abarca tanto a la propia Universidad San Sebastián como a los docentes, con el fin de brindar una visión holista del fenómeno que se quiere estudiar.

El alcance temporal es de tipo longitudinal, ya que las observaciones se han hecho en diferentes periodos de tiempo para notar las diferencias que puedan haber en el ámbito del cómo los docentes que están a cargo de impartir «*Diseño y Construcción de Software*» conducen sus respectivas cátedras, y determinar en cada instante elementos que contribuyan a nutrir con información relevante cada uno de los aspectos que se han considerado a raíz de la categoría «*B. Desarrollo de la enseñanza*». Como se están estudiando fenómenos que ocurren en el presente, y que tienen directa influencia con los hechos que vayan a ocurrir a futuro, esta investigación es de tipo longitudinal-prospectiva.

Según las fuentes utilizadas, este estudio es de tipo primario, ya que todos los datos que sean relevantes para su desarrollo se obtienen de primera mano,

[24] De acuerdo al modo en que se puede caracterizar específicamente una investigación cualitativa, según lo revisado en Cisterna, F. (2007). *Manual de Metodología de la Investigación Cualitativa…* Páginas 42-43.

puesto que el autor se encargó de la recolección de toda la información que sea pertinente. Con respecto a su naturaleza, es de tipo empírica porque se va a trabajar en el entorno natural en el cual ocurren los hechos, sin que exista una intención manifiesta por parte del investigador de hacer algún tipo de manipulación en el mismo.

Sobre los estudios a los que dé lugar, es de corte evaluativo, porque apunta a hacer una apreciación y un juicio en cuanto a la forma en que los docentes que dictan «*Diseño y Construcción de Software*» diseñan y ejecutan la enseñanza de esta asignatura que introduce a los estudiantes en el universo de la programación orientada a objetos.

1.3.1 Unidad y sujetos de estudio

La unidad de estudio en la que se va a trabajar es la Universidad San Sebastián, específicamente en la Facultad de Ingeniería y Tecnología, campus Concepción. Los estamentos que se consideraron para esta oportunidad son:

Director de la carrera Ingeniería Civil Informática. Es Ingeniero Civil Informático de la Universidad del Bío-Bío, y cuenta con grado de magíster en Ciencias de la Computación, otorgado por la Universidad de Concepción. Tiene alrededor de 12 años de experiencia en docencia de nivel superior, y su línea de intereses docentes abarca el área de bases de datos e ingeniería del software. En este estudio fue conocido como sujeto 1.

Docentes de aula que imparten «*Diseño y Construcción de Software*». Son tres profesionales cuyo denominador común es ser ingenieros civiles informáticos titulados de las universidades de Concepción y del Bío-Bío, y que tienen experiencia académica que varían entre 3 y 15 años. Además, se desempeñan como desarrolladores de aplicaciones computacionales cuando no hacen clases. Corresponden a los sujetos 2, 3 y 4.

Estudiantes. Ellos pertenecen a cada una de las especialidades de Ingeniería Civil a las cuales se les imparte la asignatura: Informática, Biotecnología, Civil e Industrial, y que serán reconocidos como los sujetos 5 al 18.

1.3.2 Instrumentos para recopilar información

La investigación cualitativa provee de un amplio repertorio de instrumentos que nos van a permitir recabar la información que sea necesaria para el enriquecimiento del estudio que se va a efectuar. En este sentido, las herramientas que se escogieron para esta oportunidad son las siguientes:

Entrevistas en profundidad. Este instrumento es uno de los que más ayuda a profundizar en cuanto a la recopilación de los datos que se van a precisar, porque permite obtener descripciones del mundo vivido por las personas entrevistadas, con el fin de lograr interpretaciones lo más fidedignas posibles de los significados que se le puedan atribuir a los fenómenos descritos.

Además, en una entrevista se pueden encontrar elementos propios del lenguaje corporal de la persona que será entrevistada — como gestos faciales, inflexiones de voz que denoten un cierto énfasis a algunos elementos propios del objeto de estudio, o cambios en su postura — para tener una mejor aproximación al fenómeno de estudio.

El tipo de entrevista fue de tipo semi estructurada, porque las preguntas ya vendrán definidas de antemano, pero cuya forma — no así su fondo — pudieran ser modificables durante su ejecución para hacerla más comprensible al entrevistado, permitiendo un cierto grado de flexibilidad al momento de su aplicación.

Observación metodológica. Se elige esta fuente de información porque le permite al investigador observar *in situ* el desarrollo del fenómeno que se quiere investigar, que en este caso se pueden observar los elementos propios del

desarrollo de la enseñanza de la programación orientada a objeto en el aula, que es el escenario natural en el cual estos hechos toman lugar.

De acuerdo con Francisco Cisterna, este instrumento de recogida de información es importante de considerar, porque no siempre el relato que pueda proporcionar el entrevistado pueda coincidir con lo que efectivamente ocurra en el espacio áulico. En este sentido, de acuerdo a la pertenencia del observador, ésta es de tipo endógena, porque quien suscribe esta investigación sí pertenece a la comunidad que observa, que en este caso corresponde al cuerpo docente de la Universidad San Sebastián.[25]

En cuanto al rol del observador, será una observación no-participante, donde el papel a cumplir se limitó a tomar notas y registros de lo que sucedía al interior del espacio áulico para cada una de las subcategorías que se desprenden de la categoría «*B. Desarrollo de la enseñanza*».

Grupos de discusión. Atendiendo a que el lenguaje es una herramienta poderosa para el intercambio de ideas y pensamientos, se pensó en incorporar un grupo de discusión conformado por seis estudiantes de la asignatura para conocer mejor sus impresiones sobre cómo sienten y asimilan lo que cada profesor ejecuta durante las sesiones de cátedra, permitiendo con esto la contrastación de opiniones entre los estamentos participantes y también de comparar las respuestas que hayan proporcionado en las entrevistas individuales.

Esta situación se potencia con el efecto de sinergia que se da en este tipo de instancias, ya que el efecto grupal que se genera propicia un aumento en el caudal de información por parte de los estudiantes seleccionados. En cuanto a la forma en que participen los miembros del grupo, se adoptó la idea de hacerlo cerrado, para que todos tengan la posibilidad de participar y que la palabra circule de la forma más ordenada posible. [26]

[25] Cisterna, F. (2007). *Manual de Metodología de la Investigación Cualitativa…*
[26] Ibíd.

Revisión documental. Esta herramienta es de gran importancia para la investigación que se quiere desarrollar, porque permite recolectar información relevante, fidedigna, y por sobre todo, imparcial para poder verificar el fenómeno que se está estudiando.

En este caso, se consultaron las planificaciones curriculares y didácticas de «*Diseño y Construcción de Software*», y para acceder a documentos propios de la Universidad en cuanto a sus directrices de tipo general y en asuntos educativos propiamente tales.

1.4 Resultados de la Investigación

1.4.1 Resultados de las entrevistas

La entrevista aplicada al estamento director de carrera revela que en cuanto a la planificación y organización de la enseñanza los docentes tienen la posibilidad de elegir la manera en cómo hacer la planificación curricular de la asignatura, para que ésta sea común a las tres secciones que existen en la actualidad, y que se pueda mantener estable en el tiempo. De acuerdo a la información proporcionada por el sujeto, la progresión en cuanto al nivel de dificultad que deban tomar los contenidos disciplinarios debía ser congruente con el esquema tradicional de desarrollo de pequeñas aplicaciones computacionales que proponen diferentes autores consultados por los propios docentes: análisis, diseño e implementación.

Se infiere además que el Director ha sugerido que los docentes traten de hacer que los estudiantes resuelvan problemas sencillos relacionados con asignaturas que se estén dictando en paralelo con «*Diseño y Construcción de Software*», como una forma de lograr que ellos comprendan que los problemas que se pueden resolver en el papel también se pueden transportar al ámbito computacional. Se busca lograr una estandarización en cuanto al nivel de dificultad que se le vaya a dar a la asignatura, ya que este es un curso común a

todas las especialidades de Ingeniería. Se deduce de sus propias palabras que no existe algún mecanismo interno que regule la profundidad con la que se revisen los contenidos: sólo puede decir que la cobertura curricular se da a partir de lo que se consulta en los exámenes finales. Con respecto a la planificación didáctica de la asignatura, él considera que cada docente es libre de escoger la metodología y recursos didácticos de enseñanza que considere más apropiada, en función de su propia capacidad pedagógica, y de la experiencia que tenga al mando de un curso. El documento empleado para hacer la planificación didáctica es el syllabus.

En relación a la cultura institucional, se deduce que él está al tanto de lo que se está haciendo a nivel de Universidad, debido a la seguridad que mostró cuando comentaba sobre cuáles eran los proyectos principales que en el área docente se estaban implementando en esta casa de estudios. Por una parte está KAYROS, que apunta a adquirir un conocimiento del estudiante de primer año, como una forma de lograr que su paso por la institución sea provechoso en términos académicos y humanos, y en otra vereda está CREAR, que se propone hacer una intervención curricular en la Universidad San Sebastián tendiente a irradiar en cada una de las carreras que se imparte el sello ético y filosófico que caracteriza a la institución, de tal manera que el resultado final apunte a un pleno desarrollo intelectual y a una integración de los individuos en el marco de la comunidad académica sebastiana.

El entrevistado le ha otorgado una especial importancia a que esta integración se puede lograr con medidas concretas como una serie de cursos de nivelación que vienen contemplados con CREAR, de tal manera que los estudiantes estén en las mismas condiciones académicas para enfrentar con éxito el desafío que conlleva conseguir su titulación profesional. Recalca además que para la institución es importante mantener un buen nivel de relaciones humanas dentro de la institución, como una forma de desarrollar a plenitud todas las metas propuestas. Siguiendo sus palabras, la Universidad considera que es positivo que los docentes cuenten con algún tipo de preparación en temas pedagógicos, sobre todo para aquellos que estén a cargo de impartir asignaturas para estudiantes de

primer año: el proyecto KAYROS contempla que a futuro los profesionales docentes puedan recibir algún tipo de certificación, como una forma de apoyar de mejor manera la preparación académica de los estudiantes novatos.

En cuanto a las creencias pedagógicas personales, la Educación es vista como una herramienta que le permite al estudiante desarrollar sus potencialidades tanto en lo cognitivo como en lo humano, y que por lo mismo asume el reto de ser docente a nivel superior como un gran desafío en su carrera profesional debido a la enorme carga de responsabilidad social que conlleva el quehacer docente. Con claridad señala que la experiencia que una persona tenga como docente es tanto o más importante a que tenga conocimientos formales en materias pedagógicas, porque deja entrever que existe una componente de improvisación que se puede desarrollar de tan buena manera que permitiría pasar por alto las carencias en cuanto a saberes oficiales del quehacer académico. Esto se reafirma con que es deseable pero no exigible que los profesionales que se dedican a hacer clase tengan algún certificado que los acredite como tales.

En la entrevista aplicada al estamento docente de aula, se puede decir que en cuanto a la planificación y organización de la enseñanza, un rasgo común que ellos evidencian es que tienen experiencia como desarrolladores de aplicaciones computacionales, y que han podido contextualizarla en una asignatura donde los estudiantes deben aprender los primeros pasos para llegar a construir este tipo de soluciones. Considerando además que la mayoría de los colegas tiene experiencia en educación superior, pudieron combinar estos elementos para converger en una planificación curricular que tiene como propósito introducir al estudiante en una asignatura cuyo objetivo es permitirle analizar problemas de mediana complejidad para resolverlos mediante la lógica de la programación. Para eso, acordaron seguir la estructura clásica que se emplea para abordar proyectos de software, que contempla las etapas de análisis, diseño e implementación. Los sujetos desconocían las definiciones técnicas que definen a cada uno de los cinco principios fundamentales que caracterizan a la planificación curricular, pero cuando se les explicó en qué consistía cada de una de ellas, brindaron

argumentaciones que permiten evidenciar que tienen arraigado la importancia que conlleva el acto de planificar.

En cuanto a lo que hacen clase a clase, los docentes estructuran sus clases a nivel de semana, y se plantean metas de contenidos disciplinarios que debieran cubrir durante las tres sesiones que tienen de lunes a viernes. Hacen esto, debido a que hay contenidos básicos — como elementos de hardware o estructuras simples de programación — que se trataron en el curso que es pre requisito que los estudiantes no tienen bien asimilados, o que simplemente han olvidado por completo, y que tienen que volver a revisar para que las clases sean provechosas.

Con respecto al desarrollo de la enseñanza, se puede señalar que en cuanto a las estrategias metodológicas de enseñanza, los docentes han optado por clases expositivas, y por otras de tipo taller, que son eminentemente prácticas. Como no poseen formación en materias pedagógicas, ellos han optado por la estrategia de enseñar del mismo modo como observaban que lo hacían sus profesores de universidad, añadiendo sus características personales. Prefieren acompañar sus explicaciones con analogías de lo que sucede en el mundo real, porque comprenden que es una asignatura complicada para los estudiantes, donde tienen que desarrollar aún más el pensamiento abstracto, y por lo mismo estos símiles que se elaboran a partir de la vida diaria creen que les podría servir para darse a entender mejor. Los recursos didácticos que prevalecen son: (1) pizarra y plumón, que emplean para hacer anotaciones auxiliares para complementar las explicaciones principales de las diapositivas; (2) proyector con diapositivas hechas en PowerPoint, que contienen el material usado como apunte oficial de la asignatura, y que cada uno elabora por separado; (3) NetBeans, que se emplea para mostrar en directo cómo se implementan los algoritmos descritos en los apuntes y para que los estudiantes identifiquen cuáles son las complicaciones adicionales que surgen, y que a lo mejor no salen descritas en las diapositivas, porque el abanico de posibilidades es muy amplio; y (4) el portal de estudiantes que ha implementado la Universidad San Sebastián que sirve como medio de comunicación para que en este caso los profesores suban el material y

esté siempre a disposición de los alumnos cada vez que ellos lo requieran. Según lo que han manifestado en conjunto, en cuanto a la distribución del tiempo, comentan que parten con un repaso que no sobrepasa los 10 minutos, y que en general no alcanzan a hacer actividad de cierre porque el tiempo se agota rápidamente; y en cuanto al espacio, dejan que los estudiantes se agrupen como estimen conveniente, quedando en evidencia que desconocen la existencia de una forma de organizar al curso dentro de la sala para que la clase sea más provechosa. Finalmente, en cuanto al clima de aula, ellos se muestran abiertos para comentarles a los estudiantes que los únicos perjudicados son ellos cuando insisten en conversar demasiado o llegar atrasados, que fueron las situaciones disciplinarias más comentadas por los docentes de aula.

Sobre sus creencias pedagógicas personales, la mayoría de los docentes entrevistados piensa que la Educación es uno de los ejes que permite forjar el crecimiento integral de los individuos de una sociedad, para ayudar a construir un país desarrollado. Bajo ese contexto, ellos están alineados en torno a la idea que todo profesional que se dedique a la labor docente debiera tener instrucción formal en materias de índole pedagógica, debido al tremendo impacto social que conlleva formar a las futuras generaciones, desde el punto de vista del conocimiento y de los valores humanos y éticos que deben proyectar en el ejercicio de su profesión.

La entrevista aplicada al estamento estudiante permite derivar en primer lugar que ven que sus profesores realizan clases de corte expositivo que combinan con talleres prácticos, donde la tónica viene dada por el trabajo de carácter grupal. Con respecto a esto, sólo una profesora realiza las clases prácticas en alguno de los laboratorios de computación con los que cuenta la Facultad de Ingeniería y Tecnología, lo que significa una amplia aprobación por parte de ellos, ya que las clases resultaron ser más provechosas cuando la ejercitación se hace en este ambiente, en comparación a los otros docentes que sólo acompañan la teoría con resolución de problemas dentro del mismo ámbito de la sala de clases, lo que no acarrea resultados significativos para los estudiantes, ya que por diferentes motivos no siempre es posible que ellos lleven

el computador portátil que la Universidad les ha facilitado desde primer año, principalmente por problemas de infraestructura de las salas de clase, que claramente no están acondicionadas para hacer cursos de esta naturaleza.

Con respecto a la interacción pedagógica, la mayoría coincide en sentirse a gusto con el hecho que el profesor los mencione por su nombre, y por la buena disposición que en general han demostrado para atender consultas que puedan surgir durante o después de las clases. Además, se cree que los desafíos que se dan en clase son abordables, porque están relacionados con conceptos que se han repasado en reiteradas oportunidades tanto en esta asignatura como en cursos paralelos o del año anterior. Los recursos didácticos pizarrón y NetBeans son los que más han comentado los estudiantes de las tres secciones consultadas, siendo este último el más importante de acuerdo a la opinión de los entrevistados porque les permite ver cómo se desarrollan las aplicaciones computacionales solicitadas por sus profesores.

En cuanto a la organización del tiempo, los estudiantes consideran que en promedio los profesores dejan espacio para atender preguntas y para que ellos puedan madurar parcialmente lo que se les está enseñando; valoran además la actividad de repaso porque les permite estar mejor conectados con lo que se quiere revisar en la clase del día. Sin embargo, cuando uno de los docentes fuera viendo los contenidos a un ritmo mayor al soportado por los alumnos, bastaba con que ellos lo manifestaran para que el sujeto en cuestión bajara la marcha. Una de las docentes tiene a su cargo la mayoría de las asignaturas de programación, y sabe la importancia que tiene estar desarrollando esta habilidad con el tiempo; por lo mismo, eso puede servir de antecedente explicativo para aclarar por qué tuvo que enseñar prácticamente de nuevo todo lo que se vio en «*Tecnologías de la Información*», porque ha pasado un semestre académico completo hasta que los estudiantes retomaran el ritmo. En cuanto a la configuración espacial del aula, lo que dan a entender los alumnos es que el profesor introduce cambios sólo cuando hay actividades de carácter grupal, donde ellos tienen la posibilidad de sentarse con quien estimen conveniente. Finalmente, casi todos coinciden en que el clima

de aula es el apropiado para desarrollar una clase de la manera más grata posible, donde ellos se sientan acogidos y con la confianza suficiente para plantear inquietudes sin sentir que puedan ser objeto de burlas o de descalificaciones. Hubo quejas desde un par de alumnos que señalan un comportamiento de su profesora que tiene como objetivo dejar en evidencia a aquellas personas que sólo conversan o llegan atrasadas, lo que generaba una sensación de apatía generalizada en contra de la docente, que traía como rebote un desgano hacia lo que ella transmite en la clase.

1.4.2 Resultados del focus group

La caracterización grupal que se puede levantar en torno al conjunto de actividades que emprenden los docentes al interior del aula para revisar los contenidos disciplinarios de «*Diseño y Construcción de Software*» contempla lo siguiente: en primer lugar, la estrategia metodológica de enseñanza empleada por los profesores apunta a hacer clases de corte expositivo que se apoyan fundamentalmente en el uso de medios didácticos como pizarrón, diapositivas hechas en PowerPoint y NetBeans que emplean para acercar de mejor manera la lógica de la programación a los estudiantes, tratando en lo posible de hacer que los alumnos constantemente estén practicando lo que indique la teoría.

Una de las dificultades que han tenido los estudiantes es que ellos pasaron un semestre completo sin haber tenido contacto con asignaturas donde ellos tuvieran que haber hecho algún programa, lo que en cierta medida les ha traído problemas para aprovechar mejor las clases de esta asignatura que presupone la maduración de esta clase de conocimientos. La otra tiene que ver con la comunicación existente entre profesor y alumno ayudante, en el sentido que el relato de algunos estudiantes sugiere que no existe una sincronización en cuanto al nivel de dificultad que se le imprime tanto a la clase teórica como práctica, lo que genera inconsistencia en cuanto a las tareas que tengan que desarrollar. Por

lo mismo, muchos contenidos disciplinarios se están repitiendo para hacer que los cursos queden nivelados con respecto a la exigencia de la asignatura, lo que significa que para los alumnos más aventajados sientan que exista una especie de vacío pedagógico que los profesores pueden cubrir a través de actividades dirigidas especialmente a ellos.

Los recursos didácticos empleados son: proyector, diapositivas, pizarrón y NetBeans, coincidiendo en que los dos últimos corresponden a los más importantes desde el punto de vista de sus procesos de aprendizaje, ya que en la pizarra se puede hacer un seguimiento de lo que se está programando en el compilador, y se pueden adjuntar explicaciones que enriquecen todo el caudal de conocimiento que se desea impartir. El tono de voz de uno de los docentes involucrados fue señalado como uno de los puntos negros en este ítem, ya que es muy bajo y cuesta seguir con claridad lo que quiere expresar. Con respecto a la organización del tiempo, queda la sensación que los objetivos de aprendizaje no siempre son declarados al inicio de cada clase, lo que significa que los alumnos no saben con claridad lo que se espera que puedan desarrollar al finalizar la sesión.

Durante el desarrollo de la misma, ellos son libres de sentarse en aquella ubicación que mejor satisfaga sus necesidades educativas, y en algunas oportunidades los profesores les piden que se sienten de a dos o tres personas para desarrollar alguna actividad. Sobre el final, no existe una conclusión clara sobre la adquisición de los objetivos de aprendizaje, y ellos prefieren retirarse del aula apenas llega la hora. Al interior del aula, se propicia un clima de respeto que viene condimentado con una dosis de humor que ayuda a desarrollar la clase de manera más efectiva. Los profesores hacen valer de manera respetuosa el cumplimiento de las normas de convivencia mínimas que debieran prevalecer en cualquier ámbito educativo: mantener un nivel de ruido en la sala que sea tolerable, concentrarse en la clase y no desviar la atención hacia sitios de Internet que no aportan mucho a la buena ejecución de la sesión.

1.4.3 Resultados de las observaciones metodológicas de aula

A partir de todo lo que se ha recogido en el trabajo de campo en cuanto a las observaciones metodológicas de aula, se puede concluir que los profesores recurren al formato de clase expositiva que combinan con sesiones de práctica donde se levanta un taller al interior de la sala de clase. Los tres docentes de aula siguieron este patrón de comportamiento, salvo uno de ellos que eligió hacer las sesiones prácticas al interior de un laboratorio de computación especialmente equipado para la ocasión en lugar de permanecer en el mismo espacio áulico donde se desarrolla el grueso de la teoría. A partir de lo observado tanto del comportamiento del profesor como de los propios estudiantes, el laboratorio resulta ser un lugar idóneo para impartir clases de programación, ya que cuenta con la infraestructura necesaria para ejecutar este tipo de cursos, al revés de lo que ocurre con una sala convencional donde: (1) los alumnos se sienten incómodos para ubicar en la silla el computador portátil y sus cuadernos para tomar notas rápidas; (2) no siempre está la disponibilidad de enchufes para cargar la batería de los notebooks; y (3) la conexión a Internet no es estable, por lo que no siempre los estudiantes pueden descargar algún material adicional que complemente su trabajo de clase.

Con respecto a la interacción pedagógica, los docentes se dirigen de manera respetuosa a los estudiantes por su nombre, y muestran una clara disposición para atender sus inquietudes. Lo que varía es el modo en cómo ellos trabajan: una docente recalca que se acuerda del nombre de las personas que destacan por alguna situación disciplinaria negativa — como llegar atrasado o conversar mucho en clase —, lo que genera un distanciamiento del curso con respecto a ella porque resulta claro que esa actitud no cae bien entre los estudiantes, lo que no significa que su modo de responder sea grosero, sino que esa mala forma de recordar a las personas genera anticuerpos que a la larga perjudican la labor de enseñanza. En otra vereda, está la profesora que utiliza el laboratorio de computación, que se aprende los nombres de sus alumnos, y que

les brinda una atención más personalizada hasta que les ayuda a solucionar in situ los problemas de aprendizaje que tengan hasta el momento, ya que se sentaba al lado de un alumno y le reiteraba las explicaciones brindadas con anterioridad.

Existe coincidencia en que los recursos didácticos que se emplean son: (1) portal docente-estudiantil de la Universidad San Sebastián, que se emplea como recurso virtual para publicar programas de asignatura, material didáctico y evaluaciones de cada curso; (2) pizarra, que la ocupan para hacer anotaciones auxiliares que complementan las explicaciones del profesor, cuyo uso no siempre fue óptimo porque en general los docentes tienden a salpicar este recurso didáctico de anotaciones vagas que no aportan mucho a la comprensión de los alumnos, haciendo que lo escrito se vea con bastante desorden; (3) PowerPoint, que es el software utilizado para hacer los apuntes de clase; y (4) NetBeans, que es el entorno de trabajo que permite la construcción de aplicaciones computacionales con Java, y que los profesores usaban para mostrar — en algunos casos — paso a paso alguna solución propuesta. Los estudiantes eran libres para organizarse dentro de la sala, siempre y cuando el orden y el nivel de ruido se mantuvieran dentro de los límites tolerables por el profesor.

Con respecto a la distribución de los tiempos, los profesores hacían una actividad de inicio donde indicaban de manera verbal lo que se espera para la clase, pero los objetivos no siempre se alcanzaban a cumplir, puesto que el cierre no siempre se hizo, ya que el tiempo de la sesión había terminado, y se veían obligados a dar indicaciones a los pocos alumnos que iban quedando en la sala con respecto a lo que se iba a ver en la clase siguiente. Finalmente, el clima de aula se percibió bueno, donde dos de los tres docentes propiciaban un ambiente que se prestaba para que los estudiantes plantearan sus inquietudes, mientras que uno de ellos no lograba tal efecto, debido a la indiferencia que se sentía del curso hacia ella.

1.4.4 Resultados de la revisión documental

Del texto «*Proyecto educativo de la Universidad San Sebastián*» se puede decir que, en cuanto a la cultura institucional, el documento define cómo se estructura y realiza el plan de enseñanza en la institución, el cual define los planes de estudio, el enfoque pedagógico y la formación integral en los estudiantes. En primer lugar, la institución construye el currículum universitario en función del perfil de egreso de sus estudiantes, donde describe el conjunto de aprendizajes esperados que cada uno de ellos debiera tener al momento de egresar, cuya estructura comprende tres grandes áreas formativas que son transversales a todas las carreras que ofrece la Universidad San Sebastián:

1. Formación Disciplinaria Básica, que abarca los cursos que se ofrecen en los primeros años de la carrera, donde los estudiantes pueden conocer los fundamentos conceptuales y teóricos de ellas, como en el caso de «*Diseño y Construcción de Software*» que es común para las distintas especialidades de Ingeniería.

2. Formación Disciplinaria Profesional, que son las asignaturas relacionadas directamente al ejercicio profesional de cada carrera. Se dictan de preferencia a partir del tercer año del plan de estudios.

3. Formación Integral, que comprende cursos de carácter obligatorio cuyo objetivo es formar a los estudiantes en concordancia con el sello de la Universidad San Sebastián.

De acuerdo a la lectura realizada, se busca que los alumnos tengan un tránsito fluido a través de la malla curricular de sus carreras, que se ve impulsado por factores como: (1) red mínima y simplificada de requisitos entre los diferentes cursos de la malla; (2) tránsito expedito entre el estado de egreso al de titulación del estudiante; (3) salidas intermedias en los planes de estudio con eventual certificación; (4) sistemas de créditos transferibles; (5) articulación entre los programas de pregrado y postgrado; (6) estructuración del currículo con ciclos

básicos y ciclos profesionales; y (7) temporada académica recuperativa de verano (TAV).

Junto a lo anterior, los planes de estudios contemplan una secuencia de tres ciclos que buscan facilitar el tránsito estudiantil con salidas intermedias, los que se enuncian a continuación:

1. Inicial, que desemboca en un Bachillerato.

2. Profesional, que conduce a la Licenciatura y luego al Título Profesional.

3. Postgrado, que incluye cursos conducentes al grado de Magíster.

En segundo lugar, el plan de enseñanza se concreta en la interacción que se produce en el espacio educativo pertinente, donde el profesor adquiere un rol que posibilite el aprendizaje de sus estudiantes, quienes deberán desarrollar capacidades de investigación y de autoestudio que les permitan mantenerse actualizados a lo largo de su vida profesional, donde la calidad de los docentes y la infraestructura física y tecnológica con que cuente la Universidad sean algunos de los factores que permitan explicar el éxito académico de los educandos. El estudiante es visto como un ser humano irrepetible de naturaleza espiritual, y que ingresa a este plantel de educación superior con carencias cognitivas y formativas que deben ser compensadas con la preparación académica que reciba, las que se determinan a través de un diagnóstico de entrada que le permitirá a la Universidad tener una visión ampliada de sus conocimientos.

El otro actor involucrado en el acto educativo es el docente, a quien la Universidad lo considera como una parte fundamental de todo el proceso de enseñanza-aprendizaje, quien deberá evidenciar conocimientos profundos en su campo disciplinar y habilidades pedagógicas y cualidades humanas que vayan en concordancia con los valores universitarios.

Y en tercer lugar, la formación integral del estudiante es aquella que busca potenciar sus capacidades personales, de acuerdo a los lineamientos que establece este documento, lo que se materializa en la diferencia que exista entre

las habilidades desarrolladas en lo cognitivo y valórico al momento de titularse con aquellas que tenía cuando ingresó a la carrera.

En el documento *«Preguntas y respuestas de nuestra Universidad»* se puede decir que no hubo registros en cuanto a aspectos relacionados con la planificación y organización de la enseñanza. Sin embargo, en materias de cultura institucional se desprende que los asuntos de carácter general apuntan hacia la misión, visión, valores, y la vinculación con el medio que hace la Universidad. La misión de la Universidad San Sebastián consiste en educar y formar profesionales bajo un modelo educativo que acoja la diversidad sociocultural de los alumnos que la Universidad matrícula, centrando el proceso docente en el aprendizaje. La visión institucional apunta a alcanzar: (1) el prestigio académico, (2) la formación integral de sus estudiantes, y (3) el compromiso con la responsabilidad social. Los valores que guían el quehacer institucional y que orientan a la comunidad académica son: (1) búsqueda de la verdad; (2) vocación por el trabajo bien hecho; (3) honestidad; (4) responsabilidad; (5) solidaridad; (6) alegría; y (7) superación. Por otro lado, la vinculación con el medio es entendida como una acción académica recíproca que se establece de manera constante con la sociedad, donde los estudiantes y docentes pasan a ser los grandes ejecutores de este ámbito. Algunas de las actividades que se incluyen dentro de este ítem son: salud y desarrollo social, programas de preparación y apoyo para la PSU, extensión académica, recuperación de espacios medioambientales, entre otros.

Con respecto a materias pedagógicas propiamente tales, la Universidad ha estado haciendo durante cinco años un proceso de caracterización de sus estudiantes, siendo el Centro de Rendimiento y Apoyo a los Estudiantes CREAR-USS el organismo interno de la institución el que se encargue de la aplicación de instrumentos como: LASSI, Thurstone y cuestionarios socio-demográficos, los cuales permitirán establecer estrategias de seguimiento que permitan mejorar las acciones pedagógicas con los estudiantes.

La Universidad San Sebastián actualiza los perfiles de egreso de sus estudiantes en base a levantamientos por competencias que respondan a los

requerimientos del mundo laboral, cautelando que estos vayan en armonía con las características locales del medio donde el alumno se vaya a desempeñar, considerando que la Universidad tiene sedes en distintos puntos del país, y que cada una de ellas está inmersa en una realidad distinta con respecto a las otras. Por otro lado, se considera la opinión de distintos grupos de interés que ayudan a revisar o actualizar los distintos planes de estudios, con el fin que estos mantengan una consistencia aproximada de dos años. Algunos de los estamentos que participan de este proceso son: (a) empleadores; (b) ejecutivos y profesionales de reconocido prestigio de las áreas disciplinarias afines de la carrera; (c) académicos de la Escuela; (d) especialistas del área; (e) departamentos que presten servicio al programa (si los hubiere); (f) egresados de la carrera; y (g) estudiantes en práctica.

El documento *«Syllabus de Diseño y Construcción de Software»* señala que en cuanto a la planificación y organización de la enseñanza apunta a las actividades que deba realizar el docente clase a clase, por lo que constituye más bien una planificación didáctica de la misma, donde se enumeran los objetivos cognitivos, procedimentales y actitudinales que se quieren alcanzar con la asignatura «*Diseño y Construcción de Software*». A continuación se presentan los eventos evaluativos que se desarrollarán durante el curso, donde la información más relevante corresponde a la fecha de aplicación de los mismos. En cuanto al desarrollo de las unidades de aprendizaje, se mantienen la clase expositiva y las sesiones tipo taller como el modelo a seguir por el profesor, sin que se aprecie una variedad mayor en cuanto a estrategias de aprendizaje, o que incorpore algún recurso didáctico innovador que sirva de apoyo a la enseñanza de la programación orientada a objetos que sea distinto de entornos de trabajo con Java como NetBeans o Eclipse. Tampoco existe alguna orientación explícita que dé cuenta de los recursos didácticos recomendados para hacer las clases o de la distribución temporal que debieran tener las actividades de inicio, desarrollo y cierre de cada una de ellas. No se registraron anotaciones específicas que develen antecedentes en cuanto a la cultura institucional.

El documento *«Programa de curso de Diseño y Construcción de Software»* indica que, con respecto al conjunto de acciones prescriptivas que realizan los docentes de aula para organizar los contenidos disciplinarios a enseñar, se puede decir lo siguiente en relación a cada uno de los principios fundamentales que debe satisfacer la planificación curricular:

Secuencialidad vertical. Resulta difícil indicar si la dificultad de los contenidos disciplinarios aumenta a medida que progrese la revisión de las unidades de aprendizaje de la asignatura, porque el ordenamiento que tienen no permite deducir si este criterio se satisface o no.

Continuidad. De acuerdo a cómo está presentado el documento, se puede decir que este principio no se respeta, ya que el orden presentado en los temas no sigue la secuencia natural de trabajo que se usa para estos casos.

Integración. No se habla de manera explícita en cuanto a un diálogo de saberes disciplinares con otras asignaturas de donde se puedan extraer ideas para desarrollar los ejercicios vistos en clase. A lo más aparece una tibia referencia a que el pre requisito de esta asignatura se llama «*Tecnologías de la Información*», pero no se especifica cuáles son los conocimientos que serán claves para que el alumno tenga un tránsito más fluido por el curso.

Congruencia horizontal. Si bien es cierto que existe una división de la asignatura en unidades de aprendizaje, y que cada una de ellas tiene una serie de objetivos junto a un listado de contenidos, no se aprecia que exista una vinculación clara entre los primeros con los segundos, o algo que indique cuáles son los saberes disciplinares que están relacionados con la consecución de determinado aprendizaje esperado, ni tampoco alguna recomendación explícita en cuanto al uso de los recursos didácticos que se consideren pertinentes de utilizar para obtener el máximo de beneficio pedagógico de la asignatura.

Comunicabilidad. El aseguramiento de este principio va a depender en la medida que este documento pueda estar disponible de manera pública para los estudiantes, ya sea por medios escritos o virtuales.

En relación a la lectura realizada a todos los documentos accesados, se puede decir en relación a la planificación y organización de la enseñanza que en el —Programa de Cursoǁ fue posible encontrar la estructura que tiene la planificación curricular del curso «*Diseño y Construcción de Software*», la cual se analizó desde el punto de vista del cumplimiento de los cinco principios básicos que ésta debiera satisfacer. Al momento de iniciar la lectura del mismo, fue posible detectar que la secuencia de las unidades de aprendizaje no permite implicar que exista un aumento progresivo de la dificultad de los mismos, ya que éstos presentan un ordenamiento poco usual. Además, no queda claro cuáles son los contenidos relevantes que se heredan de «*Tecnologías de la Información*» que el estudiante debiera traer asimilados para sacar el máximo de beneficio educativo de la asignatura. Y por otro lado, no queda en evidencia alguna vinculación entre los aprendizajes esperados por unidad didáctica con los contenidos que le dan vida a las mismas.

Por lo tanto, el cumplimiento de los principios de secuencialidad vertical, continuidad, integración y de congruencia horizontal no se desprende de manera directa de la lectura de este documento. En cuanto a la comunicabilidad, todo va a depender si los objetivos de enseñanza y de aprendizaje son conocidos de manera clara y directa por los estudiantes. Con respecto a la planificación didáctica, en el documento «*Syllabus*», los objetivos de aprendizajes en las dimensiones cognitiva, procedimental y actitudinal aparecen desglosados con claridad al inicio de esta programación, donde la estrategia metodológica sugerida apunta a la clase de carácter expositiva con retroalimentación de los alumnos, sin que se haya detectado otra más que esa. No hay referencias explícitas en cuanto al uso de recursos didácticos, o de la distribución temporal de las actividades de inicio, desarrollo y cierre que componen cada una de las sesiones del curso.

Sobre materias de cultura institucional, en «*Preguntas y Respuestas de nuestra Universidad*», las culturas institucionales en materias de tipo general guardan relación con la misión, visión, valores, y la vinculación con el medio que hace la Universidad. La Universidad San Sebastián tiene como misión educar bajo

un ambiente formativo que respete la diversidad sociocultural de los alumnos que matricula, y centra su proceso docente en el aprendizaje, donde el prestigio académico, la formación integral de los estudiantes, y el compromiso con la responsabilidad social los tres ejes sobre los cuales se articula la visión institucional, mientras que sus valores apuntan a lo siguiente: (1) búsqueda de la verdad; (2) vocación por el trabajo bien hecho; (3) honestidad; (4) responsabilidad; (5) solidaridad; (6) alegría; y (7) superación. La vinculación con el medio se entiende como una relación bidireccional entre la academia sebastiana con la sociedad, donde los estudiantes y docentes pasan a ser los principales exponentes.

En cuanto a materias estrictamente de corte pedagógico, el proyecto educativo institucional de la Universidad San Sebastián define cómo se estructura y realiza el plan de enseñanza en la institución, en cuanto a la formación pedagógica e integral de sus estudiantes. El perfil de egreso es el resultado de la convergencia de las tres áreas formativas que ha delineado la Universidad: (1) formación disciplinar básica, que es transversal para los primeros dos años de la carrera de Ingeniería al menos; (2) formación disciplinaria profesional, que se dictan a contar del tercer años; y (3) formación integral, que busca irradiar al estudiante del sello institucional. Por otra parte, la Universidad espera que los estudiantes tengan un avance fluido en el recorrido de las mallas curriculares de sus carreras, y por lo mismo genera una serie de salidas intermedias que se expresan en tres ciclos de estudios: (1) inicial, que culmina en un Bachillerato; (2) profesional, que conduce a la Licenciatura y luego al Título Profesional; y (3) postgrado, que termina en el grado de Magister, si es que la carrera lo ofrece.

Según lo que se lee del documento «*Preguntas y Respuestas de nuestra Universidad*», la actualización de los perfiles de egreso de las carreras se hace en base a las competencias demandadas por el mundo laboral, cautelando que éstos vayan en armonía con las características locales del medio donde el alumno se vaya a desempeñar, y que puedan tener una consistencia de dos años aproximados. Algunos de los estamentos que participan de este proceso son:(a)

empleadores; (b) ejecutivos y profesionales de reconocido prestigio de las áreas disciplinarias afines de la carrera; (c) académicos de la Escuela; (d) especialistas del área; (e) departamentos que presten servicio al programa (si los hubiere); (f) egresados de la carrera; y (g) estudiantes en práctica.

En «*Proyecto Educativo de la Universidad San Sebastián*» se explicita la visión que tiene en cuanto a los roles del profesor y del alumno. El primero es visto como un agente pedagógico que es capaz de posibilitar el aprendizaje de sus estudiantes, el cual deberá potenciar en ellos la capacidad de investigar y de mantener actualizados sus conocimientos durante su vida profesional, mientras que el segundo es concebido como ser un humano irrepetible de naturaleza espiritual, y que ingresa a la institución con carencias cognitivas y formativas que deben ser compensadas con la preparación académica que reciba, las que se determinan a través de un diagnóstico de entrada que le permitirá a la Universidad tener una visión ampliada de sus conocimientos.

Esta caracterización aparece explicada con mayor detención en el documento «*Preguntas y Respuestas de nuestra Universidad*», y ahí se relata que el Centro de Rendimiento y Apoyo a los Estudiantes CREAR-Universidad San Sebastián hace cinco años ha estado haciendo este proceso, a través de la aplicación de instrumentos como: LASSI, Thurstone y cuestionarios socio-demográficos, los cuales permitirán establecer estrategias de seguimiento que permitan mejorar las acciones pedagógicas con los estudiantes.

Conclusiones de la investigación

En cuanto a la planificación de la enseñanza, se detectaron los siguientes problemas, que afectan tanto a la de corte curricular como a la didáctica:

1. La secuencia de contenidos que finalmente se trataron clase a clase[27] aparecen invertidos con respecto a lo que se indica en la planificación curricular. A pesar de esto, la organización de los contenidos disciplinarios se rige por lo

[27] Análisis de problemas, diseño de la solución con UML e implementación con Java.

indicado en la literatura consultada para el marco teórico de esta investigación, la que no tuvo reparos por parte de la dirección de carrera, ya que a los profesores se les da libertad para que puedan hacer los cambios que estimen convenientes, con la condición que las tres secciones mantengan una homogeneización del orden en el cual se revisen las materias disciplinarias, porque las pruebas solemnes que se apliquen deben ser comunes para todos.

2. El material bibliográfico que se recomienda en la planificación didáctica difiere por completo del que se expresa en la planificación curricular.[28]

Otro punto que es interesante comentar a partir de lo derivado de la respuesta descriptiva tiene relación con los cinco principios que debe satisfacer toda planificación curricular. Resulta paradójico que a partir del programa de la asignatura no quede en evidencia el cumplimiento de cada uno de ellos, siendo que este documento debiera ser el elemento que oriente el quehacer docente. Sin embargo, las entrevistas aplicadas a los profesores de aula revelan que ellos están en conocimiento de estos principios[29],y que su modo de trabajar va acorde con ellos, donde la puesta en marcha del principio de integración fue una de las más promovidas por la dirección de carrera, como una forma de hacer que los académicos abordaran algunos problemas clásicos de otras asignaturas que tienen en paralelo — o que hayan sido cursadas — para que los estudiantes pudieran brindarle una solución computacional.

Los profesores coinciden en cuanto a la importancia que tiene planificar clase a clase, pero prefieren establecer un calendario semanal de las actividades que van a desarrollar, porque comprenden que los estudiantes no han pulido la habilidad para analizar, diseñar e implementar programas computacionales. Por lo mismo, tratan de resolver algunos problemas con los alumnos, y ver «*in situ*»

[28] En la planificación curricular aparecen como textos guías el de Jorge Villalobos «*Fundamentos de programación*», y «*Programación con Java*» de Thomas Wu, quedando relegado a la categoría de bibliografía complementaria el libro de los hermanos Deitel «*Cómo programar en Java*», el cual aparece como recomendado en la planificación didáctica.

[29] A pesar de no conocer con exactitud el nombre de los principios de la planificación curricular, los académicos pudieron responder con claridad a cada pregunta relacionadas con estos elementos, después que se les indicara la definición conceptual de cada uno de ellos. Las respuestas proporcionadas permiten establecer que comprenden la importancia de ellos, y que de manera tácita eran considerados en sus procesos de planificación.

cuáles son las dificultades que ellos vayan experimentando. Aprovechando la experiencia que ellos tienen como desarrolladores de aplicaciones computacionales, pueden incorporar algunas variantes a los ejercicios propuestos para hacerlos más desafiantes para los estudiantes, con el fin de reforzar sus aprendizajes en la materia.

Con respecto al desarrollo de la enseñanza, cabe señalar que — a partir de los recogido en entrevistas individuales con los profesores, alumnos, focus group y observación de aula — la estrategia metodológica que han empleado los docentes se basa en el formato de clase expositiva, donde abundan las sesiones prácticas en las cuales los estudiantes se agrupan de a dos o tres personas[30] para resolver algunos problemas propuestos por los profesores, donde los conceptos claves que se revisaron abarcaron la estructuración de un algoritmo tradicional: variables, constantes, operadores lógicos y aritméticos, ciclos iterativos y estructuras de control, más conceptos propios del paradigma orientado a objetos como objeto, clase, encapsulamiento, herencia y polimorfismo, donde se refirma que no existe una amplia variedad de conceptos en esta área, sino que el gran desafío que se abre es enseñar a combinar estas ideas para generar programas computacionales.

Algo que no se percibió demasiado a nivel observacional[31] fue que los profesores recalcaran la importancia que tiene la programación como herramienta válida para solucionar problemas que pudieran darse en el ejercicio de la especialidad de cada uno de los educandos.

Los recursos didácticos que se emplean con mayor frecuencia son NetBeans y la pizarra acrílica, siendo la combinación de estos elementos uno de los puntos que mejor opinión entre los estudiantes consultados — tanto de manera individual como grupal —, ya que en este último se pueden hacer anotaciones complementarias a lo que se está programando en el primero, lo que contribuye a

[30] Los alumnos eligen con quién quieren compartir grupos de trabajo; los docentes no intervienen en este asunto.
[31] Pero sí se reconoce la importancia que tiene a través de las entrevistas individuales.

clarificar mejor las ideas expuestas por los profesores, bajo esta modalidad por imitación que emplean para enseñar.

Al inicio de la clase, se hace un recordatorio de lo hecho anteriormente, y rara vez se enuncian los objetivos de aprendizaje. Además, la distribución de los tiempos que destinen a las tareas de clase no era la apropiada para alcanzar a terminar las actividades que se enunciaban, quedando pendiente su resolución para la clase siguiente. El ambiente que se genera al interior de la sala de clase se percibe bueno, donde los profesores generan un ambiente grato de estudio que invita a que el alumno pueda plantear sus inquietudes sin que pueda sentir temor de ser ridiculizado por el docente o por sus pares.

Con respecto a la respuesta de corte explicativo, las buenas relaciones interpersonales profesor-profesor, director-profesor, profesor-alumno, director-alumno tiene que ver con la sana convivencia que promulga la Universidad, y que se difunde a través de sus documentos institucionales, donde también se alienta a respetar la diversidad sociocultural de los alumnos que matricula, a los cuales desde primer año se les hace un estudio — dirigido por Centro de Rendimiento y Apoyo a los Estudiantes CREAR-USS — que tiene como finalidad hacer una caracterización de ellos en términos de los conocimientos que traen desde la enseñanza media.

En cuanto a la planificación y organización de la enseñanza, los principales problemas que se detectaron fueron los siguientes:

1. Los principios que caracterizan a la planificación curricular se están cumpliendo en la práctica, pero falta que queden registros por escrito de cómo se plasman para organizar de mejor manera la enseñanza de los contenidos de la asignatura. Con respecto a los alumnos, es necesario incluir sus capacidades, limitaciones, intereses, las que debieran emerger a partir del estudio de caracterización que guía el Centro de Rendimiento y Apoyo a los Estudiantes.

2. La inconsistencia que se genera entre las planificaciones curricular y didáctica dan cuenta que no se aplican protocolos en la institución que permitan resguardar la congruencia que debiera haber entre ambos tipos de documentos.

3. Una de las situaciones más comunes relatadas por los alumnos es que han debido pasar un semestre académico completo sin haber tenido asignaturas de programación, generando con esto una falta de continuidad temporal entre «*Tecnologías de Información*» y «*Diseño y Construcción de Software*», y que incluso los profesores sienten que deben volver a repasar materias que se suponen vistas y asimiladas por los estudiantes.

En relación al desarrollo de la enseñanza se puede indicar lo siguiente:

1. Según el relato colectivo de los alumnos, la estrategia metodológica que emplean los profesores no contribuye a generar una enseñanza efectiva de la programación orientada a objetos, ya que no todos son capaces de impregnarse con las ideas que transmiten los profesores, lo que reafirma los problemas que relata Villalobos en el marco teórico.

2. En la interacción pedagógica, sería recomendable acompañar las analogías que hacen los profesores — que utilizan para mejorar sus explicaciones — con casos puntuales que permitan contextualizar mejor a los alumnos con sus respectivas especialidad, como una forma de demostrar de manera empírica cómo la programación puede transformarse en una herramienta válida para solucionar problemas de cualquier índole.

3. Sobre el espacio de sala de clase, algo interesante que planteó un estudiante en una entrevista es la incomodidad que presenta el espacio de aula para ejecutar las sesiones de práctica, porque no están debidamente acondicionadas para este tipo de actividades: no hay suficientes enchufes para conectar los computadores portátiles; el tamaño de las sillas no permite tener cuadernos y notebooks al mismo tiempo; y la conectividad a Internet no siempre es constante. En cuanto a los recursos didácticos que se emplean, fue posible apreciar que en las salas donde se hizo observación de aula, se produce una

superposición del telón con el pizarrón, lo que obligaba a que el docente tuviera que estarlo acomodando de manera constante para poder escribir alguna anotación complementaria. Sin embargo, fue posible constata que los laboratorios de computación subsanan los problemas de conectividad y de tamaño de los puestos de trabajo, pero debieran refaccionarse para evitar el solapamiento de estos dos recurso didácticos.

4. El uso de pizarra acrílica ayuda a clarificar conceptos e ideas, pero falta acompañar esta buena intención con un mayor orden en cuanto al tratamiento de los temas que se estén revisando, ya que existe una tendencia natural a que este espacio didáctico se vea desordenado e incoherente. Asimismo, es recomendable mejorar la caligrafía, con el fin de dar a conocer de mejor manera el mensaje que el docente desee transmitir.

5. La voz también es uno de los instrumentos fundamentales que ocupan los profesores para transmitir conceptos e ideas, y por lo mismo tiene que ser ejercitada para lograr mejores resultados en cuando a la divulgación verbal de los contenidos disciplinarios.

Con respecto a la cultura institucional, lo que se considera más relevante desde el punto de vista investigativo es que la Universidad fomente aún más la preparación que debieran tener los docentes en materias pedagógicas, ya que eso les permitiría formalizar conocimientos que les permitan por ejemplo: conocer de una mayor variedad de estrategias metodológicas que puedan aplicar para aumentar la eficacia de sus clases; cómo emplear de mejor manera los recursos didácticos, y también para que adquieran un mayor protagonismo en cuanto a la irradiación de los valores institucionales hacia los estudiantes, tal como se indicó en la respuesta a la pregunta explicativa que guía la investigación.

El desafío de enseñar a programar es una tarea pedagógica no menor, considerando que es una habilidad que requiere de una serie de procesos mentales, donde la creatividad pasa a ser una de las protagonistas principales, junto al desarrollo del pensamiento algorítmico que el estudiante debe conjugar

con ejercicios de meta cognición que los guíen hacia una reflexión en torno a los planteamientos y esquemas mentales que estén empleando para determinar la solución de algún problema.

De acuerdo a López[32], es difícil enseñar a los alumnos a que desarrollen la creatividad para concebir programas computacionales, porque según Villalobos[33] no existe algún método universal para hacerlo, pero sí pueden existir algunas técnicas que les permitan relacionar los pocos conceptos que existen en esta área para tratar de combinarlos, y así generar un producto de software. Como se ha comentado, los docentes de aula utilizan el esquema de aprendizaje por imitación, cuya hipótesis de trabajo se basa en que el estudiante sea capaz de generar esta habilidad con tan sólo imitar lo que el profesor hace — ya sea en la pizarra acrílica o en NetBeans, lo que no ha traído buenos resultados, de acuerdo al testimonio que se recoge del relato de los alumnos.

Por lo tanto, el primer desafío que se puede enunciar para la docencia en la especialidad corresponde a una revisión — y posiblemente reestructuración — de esta estrategia metodológica y de la manera en cómo se articulan «*Tecnologías de Información*» con «*Diseño y Construcción de Software*», pero no sólo en cuanto al tiempo pedagógico de diferencia que exista entre ellas[34], sino también en relación al tratamiento de los contenidos disciplinarios que se hagan en la primera para ir en beneficio de la segunda, como una forma de conseguir un tratamiento didáctico unificado de ambas asignaturas que permitan fortalecer la base cognitiva y procedimental de los estudiantes en cuanto a la programación bajo el paradigma de la orientación a objetos.

Otro desafío pedagógico que se desprende a partir de lo investigado y de lo interpretado con el marco teórico, es que los profesores refuercen en el estudiante la importancia que tienen las etapas de análisis y diseño de la solución, para que

[32] López, J. C. (2009). *Algoritmos y Programación: Guía…*
[33] Villalobos, J. (2006). *El reto de diseñar un primer curso de programación de computadores.* Bogotá. Departamento de Ingeniería de Sistemas Bogotá, Universidad de los Andes.
[34] Que obligaría a una modificación de la malla curricular de las especialidades de Ingeniería de primer y segundo año.

no las vean como tareas aisladas[35], o como una pérdida de tiempo, según se desprende de los resultados del focus group. La importancia de esto radica en que ellos — como futuros ingenieros — deben seguir una metodología de trabajo para desarrollar software, y comprender que la tríada análisis, diseño e implementación van de la mano siguiendo ese orden, y que una etapa depende inmediatamente de la anterior.[36]

La programación es una habilidad tan difícil de adquirir, que obliga al maestro a tener que conocer ciertas técnicas que le ayuden a disminuir los niveles de frustración y de ansiedad que se generen en el estudiante cuando intente implementar por su cuenta algún programa y no le resulte, precisamente por haber sido educado bajo esta estrategia metodológica de enseñanza basada en imitación.

[35] Lo que Villalobos denomina «*aproximación de abajo hacia arriba*».
[36] Lo que justifica la secuencialidad vertical con la que se tratan los contenidos.

PARTE

II

Propuesta de Innovación e Intervención Pedagógica

Capítulo 2

Presentación de la Propuesta

Introducción

La propuesta de intervención e innovación pedagógica que acá se expone, se levanta con el objetivo de lograr un mejoramiento cualitativo de la praxis docente en el ámbito de la didáctica de la programación orientada a objetos en la Universidad San Sebastián. La formulación de la misma articula dialécticamente la formación profesional de quien suscribe este proyecto con la formación recibida en el programa de Magíster en Pedagogía en Educación Superior.

En el capítulo anterior se expuso un resumen de lo que fue una investigación cualitativa de corte descriptivo-explicativo que estudió el ejercicio docente en el área de la enseñanza de la programación orientada a objetos, y que se llevó a cabo en la asignatura «*Diseño y Construcción de Software*», que se imparte para estudiantes de Ingeniería que cursan el tercer semestre de la carrera.

En términos generales, el producto que arrojó la investigación corresponde a un diagnóstico educativo que revela cuáles fueron las fortalezas y debilidades que se detectaron en esta materia. Un problema serio que se deriva de lo comentado en el capítulo primero es que durante el curso de «*Tecnologías de la Información*» se enseña la programación de una manera que a los estudiantes de segundo año se les hace difícil de asimilar: por imitación, y empleando una terminología técnica que requiere de una familiarización previa con el razonamiento algorítmico para poder entenderlos de manera exitosa[37], y de un entorno de trabajo como NetBeans que es propio de ambientes profesionales de desarrollo de aplicaciones. Esto último no significa que su uso sea inapropiado en contextos educativos, sino que sería bueno emplear alguna herramienta — junto a una estrategia metodológica — que le ayude al estudiante a enfrentar con mejores armas los desafíos cognitivos que se vienen por delante, sobre todo si se trata de personas que recién se están incorporando al ritmo que supone la vida universitaria.

[37] Como por ejemplo diagramas UML de casos de uso, de actividad, de clases, de secuencia.

Por otra parte, una de las falencias que arrojaron los resultados de la investigación corresponde al vacío pedagógico que tienen los estudiantes al tener que cursar todo un semestre académico sin ejercitar la habilidad de programar en alguna asignatura que sea de esta área, lo que ha obligado a los profesores entrevistados a tener que revisar desde cero todo lo que fue la parte algorítmica revisada en «*Tecnologías de la Información*», porque su testimonio da cuenta que aquellos contenidos disciplinarios fueron prácticamente olvidados por la totalidad del estudiantado que está cursando «*Diseño y Construcción de Software*», considerando que ésta es una habilidad que no se adquiere de forma inmediata, y que debe ser constantemente entrenada.

Entre otros temas de interés desde el punto de vista de la planificación y desarrollo de la enseñanza, lo que se ha investigado en el curso de «*Diseño y Construcción de Software*» permite aseverar que esta asignatura en la práctica se está conduciendo de acuerdo a la secuencia lógica en que se aborda la resolución de problemas a nivel computacional: análisis, diseño e implementación. Uno de los problemas detectados es que el programa de asignatura no es congruente con este ordenamiento, pero sí lo es la forma en cómo los profesores terminan haciendo la transposición didáctica en el aula, de acuerdo a las observaciones metodológicas y a las entrevistas que se aplicaron. Por lo tanto, un primer punto a resolver lo constituye la reorganización del contenido del programa, para que este documento sea coherente con lo que se da en la praxis docente. De acuerdo al juicio profesional del autor, se considera que el recurso NetBeans es apropiado para esta asignatura, por cuanto se trata de un curso que da por finalizada la formación en programación computacional de varias especialidades de Ingeniería, y que lo mismo resulta conveniente que los alumnos tengan un primer apronte con alguna herramienta de desarrollo profesional como la que se acaba de mencionar.

Lo que se acaba de comentar sirve de antecedente explicativo para hacer una intervención en el curso que le antecede a «*Diseño y Construcción de Software*», con el objetivo que los alumnos lleguen mejor preparados a esta asignatura. Lo que se quiere conseguir en esta propuesta es que la asignatura

«*Tecnologías de la Información*» se aboque exclusivamente a la enseñanza de la programación orientada a objetos desde una la perspectiva del uso otra estrategia metodológica de enseñanza, y de la incorporación de recursos didácticos que estimulen una mejor adquisición de las habilidades que conlleva la programación orientada a objetos. En este trabajo se mostrará la manera de enseñar a programar por medio del uso conjunto entre el aprendizaje basado en problema, y el uso de herramientas educativas como Alice y BlueJ que desempeñarán el papel de ser los recursos didácticos principales para apoyar el proceso de enseñanza-aprendizaje, que le permitan al alumno la posibilidad de construir mundos virtuales interactivos que le otorguen un cariz más lúdico a la enseñanza, sin que ello signifique una pérdida de generalidad o de profundidad en cuanto al tratamiento de los contenidos.

La innovación en cuanto a la actividades que emprenden los profesores para transmitir conocimiento en el aula se justifica en que una de las conclusiones que arrojó la investigación es que la estrategia metodológica de enseñanza basada en el aprendizaje por imitación era la que predominaba al interior de la sala de clase[38], y que presenta el gran inconveniente de suponer que el alumno será capaz de generar las habilidades necesarias con tan sólo hacer las actividades del mismo modo en que lo hace su profesor, lo que de por sí sólo podría funcionar bien a nivel intuitivo, pero no existe alguna evidencia que respalde esta estrategia. Además, las clases que desarrollan los docentes de aula se basan en el formato academicista clásico de corte expositivo que presenta falencias como las que se muestran a continuación:

- No es eficaz en programas de formación que apuntan al cambio de actitudes y hábitos.
- La iniciativa descansa exclusivamente en el docente, por lo que confina a los estudiantes a un rol principalmente pasivo y receptivo.

[38] En síntesis, plantea la idea que el estudiante sea capaz de reproducir los mismos patrones de comportamiento que el profesor ejecuta al momento de brindar alguna explicación o realizar algún ejemplo en el pizarrón, con el fin que pueda establecer las asociaciones necesarias para aplicarlos en otros contextos.

- Si el docente no estimula la participación, la clase se convierte en un acto unilateral en términos comunicativos.
- Puede generar aprendizajes memorísticos cuando no va acompañada de exposiciones claras y organizadas, así como cuando el docente no genera acciones reflexivas y críticas en los estudiantes, ni desarrolla actividades de retroalimentación, o que estimule la aplicación de técnicas de metacognición en los alumnos.
- Puede discriminar negativamente a aquellos estudiantes cuyo estilo de aprendizaje sea eminentemente kinestésico.

Complementando lo anterior, Villalobos[39] plantea que dentro del contexto que involucra a la programación, esta metodología de enseñanza evidencia fallas notables cuando el estudiante se ve sometido a la resolución de problemas que involucren la concepción de estructuras de programación más profundas, porque difícilmente un estudiante promedio será capaz de ejecutar con éxito la tarea asignada por su profesor, que cuenta con un nivel de experiencia mucho más elevado. Esto de por sí termina generando sentimientos de frustración explicados en parte por sentir que no serán capaces de razonar de la misma manera que sus profesores, o bien porque ellos no les explican con claridad cuál es la importancia que tiene la programación dentro de su proceso de formación. Es por eso que es necesario trabajar con una metodología de enseñanza que propicie la participación directa y dinámica de los estudiantes en sus procesos de aprendizaje.

Jorge Villalobos indica que el desarrollo de aplicaciones computacionales debe ir acompañada de grandes dosis de creatividad, y José Antonio Cobos[40] sostiene que la promoción de la creatividad debe ir de la mano con el establecimiento de un clima positivo de aula, donde los alumnos se sientan cómodos y sin miedos para poder expresarse libremente, donde las relaciones humanas se desarrollen en un ambiente de respeto, de tolerancia y de solidaridad,

[39] Villalobos, J. (2006). *El reto de diseñar…*
[40] Cobos, J. A. (2005). *Inteligencia Emocional: Necesidad insatisfecha en la Educación*. Revista Digital I+E: Investigación y Educación. Andalucía.

en el cual el docente promueva una retroalimentación hacia sus alumnos que sea constructiva y positiva hacia ellos.

2.1 Explicitación de la propuesta

En términos concretos, la propuesta se explicitará en función de las categorías A, y B que dieron origen al estudio:

En cuanto a la planificación de la enseñanza.

Planificación curricular. Debe estar contextualizada al ambiente de trabajo de la Universidad San Sebastián, y debe dar cuenta de la satisfacibilidad de los cinco principios que rigen a esta planificación: (1) secuencia vertical; (2) continuidad; (3) integración; (4) congruencia horizontal; y (5) comunicabilidad.

Planificación didáctica. Debe reflejar las actividades que se realizarán al interior del aula, en términos de la estrategia metodológica a utilizar — que incluya la descripción de cada actividad ABP a desarrollar —, los contenidos principales, recursos didácticos necesarios, bibliografía recomendada para cada sesión, y el tipo de evaluación que se vaya a aplicar. El objetivo es que este documento sea de público conocimiento del estudiante, para que ellos tengan una orientación general del devenir de la asignatura, en función de la cronología de los contenidos y evaluaciones que tendrán lugar a través de las sesiones de clase. Para cumplir con esta disposición, el syllabus estará disponible para los alumnos en la plataforma Moodle que se habilitará para el curso.

En cuanto al desarrollo de la enseñanza.

Estrategia metodológica. Para la primera parte del curso — donde se enseña a programar con Alice — se propone la estrategia del aprendizaje basado en problemas, y para la última — que coincide con el uso de BlueJ — se propone trabajar en la modalidad expositiva combinada con lecturas comprensivas y

construcción de mapas conceptuales que permitan hacer una transición más fluida hacia entornos de desarrollo de aplicaciones de carácter profesional.

Interacción pedagógica. En cierta medida, este aspecto viene prescrito en la propia naturaleza de las actividades ABP, ya que ahí es cuando los estudiantes deben asumir determinados roles que se configurarán en términos de la complejidad de los problemas que aborden, y de la manera en cómo el docente los estimule para que puedan alcanzar los objetivos de aprendizaje. En caso de ser necesario, el profesor contará con un tutorial donde vienen propuestos una serie de problemas adicionales que van en orden creciente de dificultad, y que el alumno puede resolver, a modo de entrenamiento.

Recursos didácticos. Junto con el proyector, pizarrón, plumones, esta propuesta hará uso de otros recursos, tales como:

> **Alice y BlueJ.** Ellos se ocuparán en lugar de NetBeans, por las razones que se indican en el tercer capítulo.
>
> **Apunte preparado por el profesor.** Este documento de texto enseña la técnica de la programación orientada a objetos en ambiente Alice, junto con ejercicios y proyectos propuestos al final de cada unidad didáctica.
>
> **Apuntes de otros autores.** Se usará el texto de Héctor Sanjuán que trata sobre la historia de la computación, y cuyo objeto de uso se justifica por abarcar los temas que constituyen la primera actividad de corte grupal que se desarrolle en el curso. También se ocupará el manual de BlueJ que elaboró Michael Kölling, y que será útil para revisar algunos temas de la última parte de la asignatura.
>
> **Plataforma Moodle.** Se empleará para guardar todos los archivos de trabajo de la asignatura, lo que incluye: material didáctico de cada unidad, trabajos que suban los grupos, enlaces a sitios en Internet que sean de utilidad para la asignatura, y referencias a los videos que tengan que desarrollar los estudiantes al finalizar el segundo ABP.

YouTube. En este popular servicio de Internet se creará un canal asociado con la asignatura, en el cual los grupos deberán subir las creaciones que hayan elaborado, para que se puedan compartir experiencias y sugerencias con otros usuarios del mundo, con el fin de dar a conocer el trabajo que se realiza desde primer año con los estudiantes de Ingeniería de la Universidad.

Organización del tiempo y del espacio. De manera explícita no se propone algo en este sentido, pero quedan implícitas a partir de las actividades ABP que se indican en el anexo B.

Clima de aula. Este es un aspecto en el cual no hubo mayores reparos, ya que en general había una preocupación de los docentes en cuanto a mantener un ambiente de respeto y de armonía al interior del aula.

2.2 Objetivos

Esta propuesta de intervención e innovación pedagógica va a estar orientada al rediseño de «*Tecnologías de la Información*», que le permita al estudiante asimilar de manera significativa los conceptos fundamentales de la programación orientada a objetos. Se considera que en las categorías A y B es donde se pueden hacer los aportes principales a la optimización de la docencia de la programación orientada a objetos, ya que tienen directa relación con la acción concreta que va a emprender el docente al interior del aula. El objetivo general y los específicos que emergen a raíz de esta idea se muestran a continuación.

Objetivo general. Aportar a los procesos de optimización de la docencia y mejoramiento de los aprendizajes de los estudiantes de la carrera de Ingeniería Civil Informática en la Universidad San Sebastián, mediante el rediseño y reconstrucción curricular y didáctica de la asignatura «*Tecnologías de la Información*».

Objetivo específico 1.1. Diseñar un programa de curso que refleje a nivel teórico y práctico cada uno de los principios que caracterizan a la planificación curricular.

Objetivo específico 1.2. Optimizar la planificación didáctica que se diseña clase a clase, mediante la incorporación de elementos tales como: actividades sugeridas al inicio, desarrollo y cierre, y bibliografía recomendada.

Objetivo específico 1.3. Aplicar la metodología de aprendizaje basado en problemas para poder soslayar las dificultades encontradas con el aprendizaje por imitación.

Objetivo específico 1.4. Innovar por medio de la creación de un material didáctico que conjugue un texto de apoyo a los estudiantes con una plataforma educativa que esté orientada a la generación de aplicaciones computacionales bajo un entorno de desarrollo con orientación educativa que sirva de soporte motivacional para el estudiante, de tal manera que el alumno valore la importancia que conlleva el desarrollo del pensamiento algorítmico.

Lo que se busca en este trabajo es hacer una reestructuración del curso antes mencionado, con el objetivo que éste ayude a pavimentar una sólida base con respecto al conjunto de conocimientos que el estudiante debiera tener asimilado, para poder enfrentar de mejor manera la asignatura «*Diseño y Construcción de Software*», que se imparte al año siguiente, y donde tendrá que trabajar en un entorno de desarrollo tradicional como es NetBeans, de acuerdo a lo que se registró en la investigación.

Esta reconstrucción de la asignatura apunta a hacer un cambio en la metodología de enseñanza-aprendizaje, ya que incorpora la estrategia del aprendizaje basado en problemas en la mayor parte del desarrollo de este curso, buscando — entre otros aspectos — que desde primer año el estudiante sea

capaz de desarrollar la capacidad de trabajar en equipo, junto a todas las responsabilidades personales y académicos que ello conlleve.

El otro aspecto está ligado con el uso de Alice y BlueJ como recursos didácticos, y también el uso de YouTube y Moodle como herramientas TIC que permitirán la difusión de los trabajos que se desarrollen durante el semestre. El resto de las subcategorías ligadas con el desarrollo de la enseñanza se desprenden de la construcción de las actividades ABP y de la forma en cómo se abordan los conceptos de la programación orientada a objetos en el apunte.

Con respecto a la cultura institucional, se consideró pertinente hacer una sugerencia de corte metodológica de enseñanza en relación a C.2 Cultura institucional en materias específicas de tipo pedagógicas. La categoría D que habla sobre las creencias pedagógicas personales no reciben propuestas, ya que no corresponde intentar cambiar la opinión de otras personas, sino más bien hacer que éstas se modifiquen a partir de los resultados que arroje la puesta en marcha de este proyecto.

Capítulo 3

Fundamentación Teórica de la Propuesta

Introducción

En este capítulo se abordará el sustento teórico sobre el que se basa la propuesta de innovación pedagógica que se describió en la primera parte de este informe. Para ello, parte con lo referido a la estrategia metodológica del aprendizaje basado en problemas, donde se abordarán aspectos como: origen, definición, diferencias con la enseñanza tradicional, construcción de una experiencia ABP — que en términos prácticos fue la que se empleó para la elaboración de las actividades propuesta en el anexo B —, formas de evaluación que hay que considerar, y de algunos bemoles que pueda generar su implementación. Luego, se entrega una revisión de los conceptos fundamentales que se abordan en cualquier curso de programación, para que el lector comprenda que con un conjunto tan reducido de definiciones e ideas se pueden lograr importantes aplicaciones computacionales.

A continuación, se trata sobre el uso de software educativo como recurso didáctico, junto a las fortalezas y debilidades que conlleva su uso al interior del aula. Esta sección profundiza en torno a los casos particulares de Alice y de BlueJ, que serán los que se ocupen en esta propuesta. Finaliza este capítulo brindando una reflexión que combina los temas que acá se exponen, para ver de qué manera se pueden articular en beneficio de una correcta implementación de este proyecto.

3.1 Metodología de Aprendizaje Basado en Problemas (ABP)

3.1.1 Introducción

El Aprendizaje Basado en Problemas (ABP) es uno de los métodos de enseñanza -aprendizaje que ha tomado más arraigo en las instituciones de educación superior en los últimos años.

Según lo que señala la Dirección de Investigación y Desarrollo Educativo del Instituto Tecnológico de Monterrey[41], el camino que se sigue en el proceso de aprendizaje basado en problemas es diferente del que recorre el método tradicional, ya que en el ABP el proceso comienza con el planteamiento de un problema, se identifican las necesidades de aprendizaje, se busca la información necesaria, y finalmente se regresa al problema.

El recorrido que experimentan los alumnos desde la formulación del problema hasta su solución, está marcado por el trabajo colaborativo que deben desempeñar en grupos, donde se busca que en esa experiencia de aprendizaje descubran la posibilidad para practicar y desarrollar habilidades, de observar y reflexionar sobre actitudes y valores que en el método convencional expositivo rara vez se ven en acción, dado el rol preponderante que toma la figura del profesor.

En la educación convencional, el alumno es un sujeto pasivo del grupo que sólo recibe la información por medio de lecturas, de la exposición del profesor y de aquella que pueda recibir de parte de sus compañeros. En cambio, en el ABP él es quien busca el aprendizaje que considera necesario para resolver los problemas que se le plantean, y que conjugan aprendizajes provenientes de diferentes áreas de conocimiento.

3.1.2 Inicios del ABP

En la década de los años 60’ y 70’, un grupo de educadores médicos de la Universidad de McMaster en Canadá reconoció la necesidad de replantear tanto los contenidos como la forma de enseñanza de la medicina, con la finalidad de conseguir una mejor preparación de sus estudiantes para satisfacer las demandas de la práctica profesional. En esta misma época, en paralelo comienza la

[41] Dirección de Investigación y Desarrollo Educativo. (2000). *El Aprendizaje Basado en Problemas como técnica didáctica*. Monterrey: Vicerrectoría Académica, Instituto Tecnológico y de Estudios Superiores de Monterrey.

implementación de esta metodología en la Escuela de Medicina de la Universidad de Case Western Reserve en Estados Unidos. En algunas escuelas de medicina como McMaster o Harvard, la organización del currículum de los primeros años de estudio ha sido modificada completamente al basarse a este tipo de estrategia de enseñanza.

En Latinoamérica, este enfoque lo aplican diferentes universidades, entre las que se cuentan la Universidad Estatal de Londrina y la Facultad de Medicina de Marília en Brasil, y la Universidad Nacional Autónoma de México, entre otras. En Chile destacan los casos de la Universidad de la Frontera, de Concepción y del Bío-Bío.

3.1.3 Qué es el ABP

El ABP es una estrategia de enseñanza-aprendizaje en la que tanto la adquisición de conocimientos como el desarrollo de habilidades y actitudes resultan importantes. El ABP se sustenta en diferentes corrientes teóricas sobre el aprendizaje humano, pero tiene particular presencia la teoría constructivista, ya que en esta metodología se siguen tres principios básicos:

1. Lograr un entendimiento de una situación real que surge a partir de las interacciones con el medio ambiente.
2. El conflicto cognitivo que genera en el estudiante el enfrentarse a una nueva situación estimula su aprendizaje.
3. El conocimiento se desarrolla mediante el reconocimiento y aceptación de los procesos sociales y de la evaluación de las diferentes interpretaciones individuales del mismo fenómeno de estudio.[42]

[42] Ibíd.

El ABP incluye el desarrollo del pensamiento crítico en el mismo proceso de enseñanza-aprendizaje, no como algo adicional sino como parte del mismo proceso de interacción para aprender. Esto es particularmente importante puesto que el ABP busca que el alumno comprenda y profundice adecuadamente en la respuesta a los problemas que se usan para aprender, abordando aspectos de orden filosófico, sociológico, psicológico, histórico, práctico, etc., siguiendo un enfoque de trabajo integral, de tal manera que la nueva información que reciba se enlace con las ideas y conceptos que el estudiante ya tiene asimilados dentro de su estructura cognoscitiva, de acuerdo a lo que se lee del artículo de Francisco Ramis.[43]

La estructura y el proceso de solución al problema están siempre abiertos, lo que motiva a un aprendizaje consciente y al trabajo de grupo sistemático en una experiencia colaborativa de aprendizaje. El ABP considera al alumno y sus competencias para la planificación de sus propias metodologías de aprendizaje, y de la forma en cómo desarrolle los contenidos.

Los alumnos por lo general se agrupan en equipos de cinco a ocho integrantes, donde el profesor es el facilitador que promoverá la discusión en la sesión de trabajo, de tal manera que el docente asume un rol que les permite a los alumnos apoyarse en él para la búsqueda de información. Algo que es importante destacar es que el objetivo no es resolver el problema, sino que éste sea utilizado como base para identificar los temas de aprendizaje para su estudio de manera independiente o grupal; vale decir, que el problema hace las veces de detonador para que los estudiantes puedan cubrir los objetivos de aprendizaje del curso. A lo largo del proceso de trabajo grupal, los alumnos deben adquirir responsabilidad y confianza en el trabajo realizado al interior del equipo, para lo cual deben desarrollar la habilidad para dar y recibir críticas orientadas a la mejor de su desempeño, y del proceso de trabajo de todo el grupo.

[43] Ramis, F., Bañados, C., & Muñoz, Á. (2007). *Aprendizaje basado en problemas en el contexto de resolución de problemas*. Concepción: Universidad del Bío-Bío.

El ABP puede ser usado como una metodología de trabajo a lo largo del plan de estudios de una carrera profesional, o bien ser implementado como una estrategia de trabajo a lo largo de un curso específico, e incluso como una técnica didáctica aplicada para la revisión de ciertos objetivos de aprendizaje de un curso, sin que sea necesario abordar una asignatura completa desde esta perspectiva.

Una de las características del ABP está en fomentar en el alumno la actitud positiva hacia el aprendizaje, ya que en este método se respeta la autonomía del estudiante, quien aprende sobre los contenidos y se apoya con la propia experiencia de trabajo que supone la dinámica de un grupo. Además, los alumnos tienen la posibilidad de observar en la práctica, algunas aplicaciones de lo que se encuentran aprendiendo en torno al problema.

Al trabajar con el ABP, la actividad gira en torno a la discusión de un problema, y el aprendizaje surge de la experiencia de trabajar sobre esa cuestión particular, lo que hace que este método estimule de manera positiva el autoaprendizaje y permita que el estudiante aprenda y al mismo tiempo ejercite lo que vaya asimilando en contextos reales, junto con la identificación de las deficiencias de conocimiento que experimente a lo largo del proceso. Esto permite concluir de manera parcial que dentro de las ganancias educativas que conlleva la elección de esta metodología de enseñanza-aprendizaje destaquen las siguientes:

1. Pensar de forma crítica y ser capaz de analizar y resolver problemas complejos de la vida real atingentes a su área profesional de desarrollo.
2. Encontrar, evaluar y utilizar las fuentes de información adecuadas para la adquisición de sus propios aprendizajes.
3. Trabajar colaborativamente en pequeños grupos, lo que favorece que los alumnos gestionen eficazmente los posibles conflictos que surjan entre ellos y que todos se responsabilicen de la consecución de los objetivos previstos. Esta responsabilidad asumida por todos los miembros del grupo ayuda a que la motivación por llevar a cabo la tarea sea elevada y que adquieran un compromiso real y fuerte con sus aprendizajes y con los de sus compañeros.

4. Desarrollar habilidades para la comunicación, tanto verbal como escrita.
5. Esta metodología favorece la posibilidad de interrelacionar distintas materias o disciplinas académicas. Para intentar solucionar un problema, los alumnos pueden necesitar recurrir a conocimientos de distintas asignaturas que ya se hayan adquirido, lo que les ayuda a generar una globalidad unificada de saberes.

3.1.4 Diferencia entre ABP y la enseñanza tradicional

En la Tabla 1 se muestran algunas de las diferencias más importantes que se pueden dar entre aprendizaje tradicional y el ABP, a partir de lo que se infiere del trabajo de Patricia Rojas.[44]

Tabla 1 Diferencias entre Aprendizaje tradicional y ABP.

Elemento del aprendizaje	En el aprendizaje tradicional	En el ABP
Responsabilidad de generar el ambiente de aprendizaje y los materiales de enseñanza.	Asumido por el profesor.	La situación de aprendizaje es presentada por el profesor y el material de aprendizaje es seleccionado y generado por los alumnos.
Secuencia en el orden de las acciones para aprender.	Determinadas por el profesor.	Los alumnos participan activamente en la generación de ésta secuencia.
Momento en el que se trabaja en los problemas de	Después de presentar el material de enseñanza.	Antes de presentar el material que se ha de aprender.

[44] Rojas, P. (2010). *El aprendizaje basado en problemas como estrategia metodológica de enseñanza y aprendizaje de la integral indefinida, en paralelo con derivadas y su incidencia en el rendimiento académico de los estudiantes de Ingeniería en Informática de INACAP Chillán*. Tesis para optar al grado académico de Magíster en Enseñanza de las Ciencias, mención Matemática. Chillán: Universidad del Bío-Bío.

Elemento del aprendizaje	En el aprendizaje tradicional	En el ABP
ejercicios.		
Responsabilidad de aprendizaje.	Asumida por el profesor, mientras que los alumnos absorben, transcriben, memorizan y repiten la información para actividades específicas como pruebas o exámenes.	Los alumnos asumen un papel activo en la responsabilidad de su aprendizaje, identifican necesidades de aprendizaje, investigan, aprenden, aplican y resuelven problemas.
Presencia del experto	El profesor representa la imagen del experto.	El profesor es un tutor sin un papel directivo, es parte del grupo de aprendizaje.
Evaluación	Determinada y ejecutada por el profesor. Los alumnos buscan la «*respuesta correcta*» para tener éxito en un examen.	El alumno juega un papel activo en su evaluación y la de su grupo de trabajo. Los profesores evitan solo una «*respuesta correcta*» y ayudan a los alumnos a armar sus preguntas, formular problemas, explorar alternativas y tomar decisiones efectivas.
Formas de evaluación.	La evaluación es sumatoria y el profesor es el único evaluador.	Los estudiantes evalúan su propio proceso así como los demás miembros del equipo y de todo el grupo. Además el profesor implementa una evaluación integral, en la que es importante tanto el proceso como el resultado.

Algunas de las consecuencias que se derivan a partir de la tabla anterior son las siguientes:[45]

- Este método estimula que los estudiantes generen mayores vínculos y motivación hacia el aprendizaje, ya que les hace sentir que tienen la posibilidad de interactuar con la realidad y observar los resultados de dicha interacción.
- El ABP ofrece a los alumnos una respuesta clara a interrogantes que apunten hacia cuestiones como la importancia de un determinado tema, o la forma en cómo se relaciona lo que hay que aprender con lo que ya se tiene incorporado. Esta respuesta es más clara que en el aprendizaje tradicional, dada la naturaleza de carácter investigativa y colaborativa que subyace a la formulación de esta estrategia metodológica.
- La propia dinámica del proceso ABP lleva a los estudiantes a desarrollar de mejor manera un pensamiento crítico y creativo, que se requiere sobre todo para aprender a programar.
- El ABP promueve que los alumnos sean partícipes de la evaluación de sus propios aprendizajes, ya que ellos construyen sus propias estrategias para la definición del problema, identificación de fuentes de información, análisis de datos, etc.
- El ABP permite que todo lo aprendido quede mejor internalizado en la estructura cognoscitiva del estudiante, y no que se memorice de manera transitoria, de tal manera que la información tiene un rol más significativo para ellos. [46] Este aprendizaje obedece a una necesaria integración de conocimientos que pueden provenir de diferentes áreas del saber.
- La estimulación de habilidades de estudio auto dirigido le permite a los alumnos mejorar su capacidad para estudiar e investigar sin ayuda de nadie para afrontar cualquier obstáculo, tanto de orden teórico como práctico, a lo largo de su vida.

[45] Dirección de Investigación y Desarrollo Educativo. (2000). *El Aprendizaje Basado en Problemas…*
[46] Esta característica es ideal para construir una buena base de conocimientos y adquisición de destrezas en programación que luego los alumnos tendrán que aplicar en cursos más avanzados.

- El ABP promueve el trabajo participativo del estudiante cuando se ve enfrentado a una situación de aprendizaje que requiere la pertenencia a algún equipo.

3.1.5 Implementación de ABP en el aula

A partir de esta sección de indican algunas orientaciones de corte práctico que permiten formular e implementar alguna experiencia ABP en el aula, que abarca desde la presentación del problema hasta algunas sugerencias que se pueden seguir para evaluar. Para que un problema sea considerado como uno de tipo ABP, debe reunir tres condiciones esenciales: (1) presentación; (2) escenario; y (3) condiciones de ejecución.[47]

Presentación del ABP

Para iniciar un trabajo ABP, el docente debe decidir en qué momento de la asignatura va a aplicar esta metodología, y cuáles serán los objetivos generales y específicos de aprendizaje que se espera que logren alcanzar los estudiantes, junto a las destrezas que se deseen alcanzar durante el proceso.

Escenario de trabajo

El eje fundamental de trabajo en el ABP lo constituye el planteamiento del problema que abordarán los alumnos. Lo importante es que ellos se sientan involucrados y con mayor compromiso en la medida que la situación presentada represente un reto y una clara posibilidad para que puedan adquirir aprendizajes significativos. Algunas características importantes que debiera tener son las que se indican a continuación:

[47] Estas ideas fueron consultadas desde un instructivo creado por el Instituto Profesional Virginio Gómez para un taller de formación docente en aprendizaje basado en problemas, en el año 2005. Los resultados de esta parte se pueden consultar en el anexo B.

1. Debe comprometer el interés de los alumnos y motivarlos a lograr una comprensión más profunda de los conceptos y objetivos que se deseen aprender, lo que significa que el problema planteado tiene que estar relacionado con los objetivos de aprendizaje del curso, y por tanto ha de estar elaborado en un contexto familiar, con una estructura compleja y retadora, para que los alumnos encuentren mayor sentido en el trabajo que tendrán que realizar. La misión en este caso consiste en motivar la búsqueda independiente de la información a través de todos los medios disponibles para el alumno, de tal manera que se favorezca la discusión a nivel de grupo.

2. Tiene que provocar algún conflicto cognitivo, porque el problema tiene que llevar a los alumnos a tomar decisiones, o hacer juicios basados en hechos, información lógica o fundamentada. Ellos tienen que justificar sus decisiones y razonamiento en los objetivos de aprendizaje del curso, de tal manera que el problema planteado requiere que los estudiantes definan qué suposiciones son necesarias y por qué, qué información es relevante y qué pasos o procedimientos son necesarios con el propósito de resolver la situación planteada.

3. Tiene que asegurar la participación de todos los miembros del grupo de aprendizaje, durante su desarrollo. La colaboración de todos los integrantes del grupo de trabajo es necesaria para poder abordar el problema de manera eficiente, y por ello se necesita que el problema sea complejo; que la situación dada no la pueda resolver sólo un alumno en particular, sino que se haga necesaria la participación de todo el grupo. Su diseño tiene que estimular que los alumnos utilicen el conocimiento previamente adquirido, tal como se comentó en párrafos anteriores. El trabajo del docente es fundamental, pues él tendrá que usar las estrategias metodológicas adecuadas que aseguren la participación de todo el grupo en la solución del problema, así como ir evaluando el proceso constantemente para lograr que todos los integrantes del grupo de trabajo participen y logren alcanzar todos los objetivos que se deseen adquirir con esta situación; por lo tanto es importante que el profesor tenga claro que la mayoría de las actividades se efectúan en la sala de clases.

4. La solución del problema planteado no ha de ser única: el ABP no se reduce a un simple problema con enunciado y solución única, sino que esta estrategia le permite al alumno — con toda la información necesaria y los conocimientos adquiridos — dar diversas respuestas a la situación planteada, y por ello el trabajo colaborativo es muy importante, pues con las ideas que cada integrante del grupo plantee, se podrá elaborar la solución que ellos consideren que sea la más adecuada. Es por esto que se considera que el producto final de una experiencia de esta naturaleza puede ser un informe, la elaboración de un afiche, de una campaña, de una maqueta, la creación de alguna revista, creación de una empresa, video, un software, preparación de un seminario, de una capacitación, etc., para permitir que ellos muestren soluciones creativas. Todo esto permite que los estudiantes trabajen en grupos, y no que ellos terminen participando de manera individual sin compartir sus ideas y avances propios.

Condiciones del ABP

Luego que el docente ha creado el escenario del ABP, debe establecer todas las condiciones que el estudiante ha de cumplir con respecto a:

Formación de grupos. Es necesario que quede por escrito y de la manera más clara posible la cantidad de alumnos que integrarán cada grupo de trabajo.

Producto final del ABP. El docente de la asignatura puede pedir diferentes productos, como por ejemplo: informe, maqueta, afiche, campaña publicitaria, programa computacional, video, animación, elaboración de una capacitación, elaboración de un seminario, creación de una revista, creación de una empresa.
Después de la presentación del problema, se debe indicar cuáles son las condiciones de entrega del producto final, sin que el alumno tenga la posibilidad de hacer supuestos.

Exposición. Todo ABP finaliza con una exposición de los grupos de trabajo; para ello el docente ha de indicar cómo se efectuará, indicando los aspectos más

relevantes, como por ejemplo: tipo de vestimenta — formal o informal —, tiempos para exponer, medio a usar para la exposición — papelógrafo, *PowerPoint*, *Prezi*, *Beamer*, etc. —, cantidad de alumnos que exponen, quién elige a los expositores, o qué sucede con los alumnos que no expongan.

Ponderaciones. Se le debe indicar a los alumnos cómo serán evaluados, y cuál es el porcentaje que cada uno de estos eventos tiene asociado. Se recomienda evaluar los siguientes procesos: producto final, exposición, actividades grupales por parte del profesor, actividades grupales por parte el alumno, a través de una co-evaluación, actividades individuales por el profesor.

Fechas. Es importante destacar cuáles son las fechas de entrega, tanto para el proyecto final como para la exposición. Como el trabajo en el ABP es colaborativo, se recomienda indicar qué sucederá en caso que algún alumno falte, en caso de haber alguna evaluación. También es importante destacar qué pasará en caso que el grupo no entregue el producto final en la fecha indicada, o si no expone.

Proceso ABP

Una vez que se ha hecho la formulación de la actividad ABP, es necesario planificar cómo se llevarán a cabo durante el periodo de tiempo que demore su ejecución, de tal manera que los alumnos puedan alcanzar los objetivos propuestas al elaborar un problema de esta naturaleza. Lo que se sugiere es construir una tabla que incluya estos elementos:

Sesión. Se indica el número correlativo de la sesión que será desarrollada, junto con la fecha de realización.

Etapas, pasos y acciones. Se indican las etapas, pasos o acciones que se llevarán a cabo en cada sesión, donde el foco principal de la atención gira en torno a las acciones individuales o grupales de los alumnos y del profesor.

Descripción detallada de las acciones. Se brinda una descripción clara y precisa de cada uno de los pasos o acciones. Se distinguen y explican cuáles son las actividades serán realizadas por el profesor y cuáles por el alumno.

Tiempo. Entrega una estimación del tiempo que se le asigne a cada actividad.

Productos. Corresponden a los productos intermedio que se deben ir generando en cada una de las sesiones de trabajo, donde el docente deberá indicar si acaso éstos serán evaluados o no. Es importante destacar que los productos que se construyan deben guardar alguna relación entre sí, como por ejemplo que uno constituya la base sobre la cual se fabricará el otro. La idea es que cada uno de estos resultados permita evidenciar que los alumnos hayan adquirido los aprendizajes esperados.

Materiales requeridos. En este punto se indican todos los materiales que serán necesarios para el desarrollo de cada una de las acciones y para la elaboración de los respectivos productos.

Actividades productos fuera del aula. Se identifican cuáles son las actividades y/o productos que deberán ser desarrollados por el alumno fuera de la clase.

Actividades que emprenden los estudiantes en la experiencia ABP

Tal como se ha comentado con anterioridad, el docente es sólo un facilitador de conocimiento, y no adquiere el rol expositor que es propio del enfoque academicista clásico; en este caso, son los alumnos quienes irán adquiriendo las herramientas necesarias para lograr los objetivos el profesor necesita para que ellos internalicen y les permita dar solución al problema planteado inicialmente, por

medio de las actividades planificadas. Se sugiere que éstas sean las que se indican a continuación:

Leer y analizar el escenario presentado. Los alumnos discuten en el grupo los puntos necesarios para establecer un consenso sobre cómo se percibe dicho escenario.

Identificar cuáles son los objetivos del aprendizaje. Los alumnos se informan de los objetivos que se pretenden cubrir con el problema que el profesor les ha planteado.

Identificar la información con la que cuenta. Los alumnos elaboran un listado de lo que ya se conoce sobre el tema, e identifican cuál es la información que se maneja entre los diferentes miembros del grupo.

Esquema del problema. Los alumnos elaboran una descripción breve del problema, que debe dar cuenta de lo que el grupo está tratando de resolver, reproducir, responder o encontrar de acuerdo al análisis de lo que ya se conoce, lo que debe ser revisado en cada momento donde una nueva información esté disponible.

Diagnostico situacional. Los alumnos elaboran grupalmente una lista de los que se requiere para enfrentar al problema, preparar un listado de preguntas de lo que se necesita saber para poder solucionarlo, junto a los conceptos que requieren dominar para poder brindar la mejor solución posible al problema planteado. Este punto en el que el grupo está trabajando en la elaboración de su propio diagnostico situacional en torno a los objetivos de aprendizaje y a la solución del problema.

Esquema de trabajo. Los alumnos preparan un plan con posibles acciones para cubrir las necesidades de conocimiento identificadas y donde se puede señalar las recomendaciones, soluciones o hipótesis. Es pertinente elaborar un esquema que señale las posibles opciones para llegar a cubrir los objetivos de aprendizaje y la solución del problema.

Recopilar información. Los alumnos en su correspondiente grupo de trabajo buscan información en todas las fuentes pertinentes para cubrir los objetivos de aprendizaje y resolver el problema.

Analizar la información. Los alumnos trabajando en el grupo analizan la información recopilada, buscan opciones y posibilidades y se replantean la necesidad de tener más información para solucionar el problema, en caso que fuera necesario.

Plantearse los resultados. Los alumnos en su respectivo grupo preparan una primera aproximación con recomendaciones, estimaciones sobre resultados, inferencias u otras resoluciones apropiadas al problema. Todo lo anterior debe estar basado en los datos obtenidos y en los antecedentes. Todo el grupo debe participar en este proceso de tal modo que cada miembro tenga la capacidad de responder a cualquier duda sobre los resultados.

Retroalimentar. Este proceso debe ser constante a lo largo de todo el proceso de trabajo del grupo, de tal manera que sirva de estímulo para la mejora de las actividades que se emprenden en este periodo. Se recomienda que al final de cada sesión se genere la instancia para la retroalimentación grupal. A lo largo del proceso el grupo debe estar atento a retroalimentar en tres diferentes aspectos:

1. La relación del grupo con el contenido del aprendizaje.
2. La relación de los miembros dentro del grupo.
3. La relación de los miembros con el tutor del grupo, por medio de reuniones que se celebren dentro de las clases.

En este tipo de situaciones que dan lugar a sesiones de aprendizaje colaborativo, el docente conforma grupos que apuntan en esta dirección, además de la enseñanza de conceptos y estrategias básicas de estudio, vigila el funcionamiento de los grupos, interviene para demostrar habilidades en grupos pequeños y brinda ayuda en las tareas cuando corresponde, evalúa el aprovechamiento de los estudiantes validándose de un sistema basado en

criterios y garantiza que los grupos procesan cuan eficazmente trabajaron juntos los integrantes. Los estudiantes buscan ayuda, retroinformación, refuerzo y apoyo de sus compañeros.

Actividades que emprende el docente durante la experiencia ABP

El docente o facilitador ha de desarrollar las siguientes actividades:

Especificar los objetivos de la clase. El docente ha de indicar no sólo los objetivos académicos sino también los objetivos de habilidades sociales que se requieren para que los estudiantes aprendan a colaborar eficazmente en forma colectiva.

Determinar el tamaño del grupo. Se recomienda que el tamaño de los grupos varíe entre dos y cuatro estudiantes, pero esta cifra puede cambiar según los objetivos y circunstancias particulares de una clase.

Asignar los estudiantes a los grupos. Los estudiantes pueden ser agrupados de muchas maneras, pero se podrían explorar otras alternativas, como por ejemplo: azar, por afinidad[48], por orden lexicográfico a partir de la lista de la clase, mediante un test para diferenciar los estilos de aprendizaje.

Organización física de la sala de clases. La distribución del espacio de aula transmite un mensaje simbólico sobre el comportamiento apropiado y la distribución que los grupos de aprendizaje han de tener para la realización de una tarea. Los integrantes de un grupo de aprendizaje deberían sentarse cara a cara y codo a codo, con el fin de estar lo suficientemente cerca como para compartir materiales, mantener el contacto con la vista, hablar bajo para no molestar a los demás, e intercambiar ideas en un ambiente cómodo, en el cual el docente tenga un libre acceso a cada grupo de trabajo.

Selección de los recursos didácticos. Los materiales seleccionados dependen del tipo de tarea que los estudiantes deben realiza. Cuando el docente decida lo

[48] Al final, resulta ser el más utilizado, ya que permite la conformación de grupos homogéneos de trabajo.

que se requiere, conviene distribuir los materiales entre los integrantes del grupo con el fin de que todos puedan participar y aprovechar. Si se trata de integrantes de grupo maduros con un nivel de habilidades interpersonales y en grupos pequeños posiblemente el profesor no tenga que organizar el material de algún modo determinado. Por otra parte, si se trata de un grupo nuevo — como en el caso de los estudiantes de primer año hacia lo que va dirigida la asignatura —, conviene que los profesores distribuyan los materiales de manera cuidadosa y planificada.

Asignar funciones para garantizar la interdependencia. Al planificar una clase, los docentes tienen que pensar en las acciones que maximizaran el aprendizaje de los estudiantes. Dichas acciones pueden ser definidas como papeles a desempeñar, o funciones que son asignadas individualmente a cada integrante del grupo. Las funciones podrían incluir a alguien que resuma contenidos, un comprobador de comprensión, un asesor, un fomentador de participantes y un observador. Asignar funciones complementarias e interconectadas a los integrantes del grupo resulta eficaz para enseñar habilidades sociales a los estudiantes y fomentar la interdependencia positiva.

Explicar la tarea académica, al inicio de una clase el docente tiene que explicar la tarea académica con el fin de que los estudiantes tengan bien claro lo que tiene que realizar y comprendan los objetivos de la clase.

Explicar los criterios para alcanzar el éxito y los comportamientos deseados. Los alumnos tienen que saber qué tipo de evaluación se les aplicará durante las clases: formativa o sumativa, y cuáles serán los criterios que se deberán seguir para esos efectos.

Técnicas que se emplean dentro del aula

Algunas de las técnicas que los docentes pueden emplear los docentes al interior del aula para conducir mejor la experiencia ABP entre sus alumnos son las siguientes:

Los guiones de aprendizaje colaborativo. Corresponden a métodos colaborativos convencionales de contenido libre que pueden servir para presentar lecciones genéricas y repetitivas (como por ejemplo redactar informes o dar presentaciones) o para manejar las rutinas de las clases (como por ejemplo revisar la tarea y los exámenes). Uno de los más conocidos es un guion simple para tratamiento de textos denominados MURDER (siglas en inglés que corresponden a movilizar, entender, recordar, detectar, expandir, revisar). En este guion los estudiantes son agrupados en parejas y primero movilizan sus recursos para aprender (1) estableciendo un estado de ánimo apropiado y (2) analizando el texto para establecer puntos de acción colaborativa (asteriscos que aparecen en el margen para indicar donde detendrán la lectura y comenzaran con el procesamiento de información colaborativo).

Ambos integrantes de la pareja leen en silencio para entender hasta que llegan al primer punto de acción. Uno de los compañeros recuerda/recita lo que ha aprendido hasta ese momento mientras el otro compañero detecta y corrige errores y omisiones, luego, ambos expanden sobre el material en forma colaborativa formando imágenes, analogías y conexiones directas con otra información, luego continúan leyendo en silencio para entender, hasta llegar al siguiente punto de acción, en ese momento invierten sus papeles y repiten los pasos de recordar, detectar y expandir, la pareja continua con el material, alternando sus papeles hasta completar la actividad, después, en forma colaborativa, revisan y organizan toda la información, alternando nuevamente entre la función activa y la función de vigilancia.

Estructura simple. Es una entrevista de tres etapas, en la que los alumnos son agrupados en parejas; el alumno A entrevista al B, el alumno B entrevista al A, y los dos comparten sus resultados con otra pareja.

Co-op co-op. En este método los alumnos son asignado a grupos heterogéneos de aprendizaje colaborativo, a cada grupo se asigna una parte de la unidad de aprendizaje y a cada miembro se le da un mini tema para que los complete solo y

después lo presente al grupo, luego cada grupo integra los mini temas de sus compañeros en una presentación en grupo para todos.

Investigación en grupo. Este método es parecido al Co-op co-op, pero su complejidad radica en que los estudiantes forman grupos colaborativos según sus intereses comunes en un tema, y ellos guían al grupo para definir la manera en cómo se investigará su tema. Luego dividen el trabajo, y cada integrante del grupo realiza parte de la investigación, el grupo compendia y resume su trabajo y presenta sus resultados ante el resto de sus compañeros.

Rompecabezas. En este método los alumnos son asignado a grupos colaborativos, el mismo tema se asigna a todos los grupos y cada integrante tiene que aprender una sección singular del tema e impartirlo después a los demás integrantes del grupo, los compañeros estudian el tema independientemente y luego lo presentan al grupo, el grupo compendia las presentaciones de los individuos en la prospectiva global, por ejemplo, si la materia de estudio es la vida de un cierto personaje, cada integrante del grupo recibe material correspondiente a una parte de su vida, pero ningún estudiante puede aprender todo al respecto a menos que todos los integrantes expongan su parte.

Independiente del método colaborativo empleado, el docente tiene que vigilar a los estudiantes mientras trabajan, y debe intervenir cuando corresponda, de tal manera que esa ayuda se canalice en pro de lograr una interacción más hábil entre los miembros del grupo de trabajo. Los docentes pueden determinar, escuchando atentamente cuando los estudiantes se explican entre ellos lo que están aprendiendo, lo que los alumnos entienden y lo que no entienden. A través del trabajo colaborativo los alumnos manifiestan proceso de razonamiento ocultos que pueden ser observados e interpretados y los docentes pueden detectar cómo construyen los alumnos su entendimiento del material asignado e intervenir cuando corresponda para aumentar la comprensión.

Además, una vez que los alumnos han completado una actividad, o que el tiempo concebido ha transcurrido, los alumnos deberían participar en

procesamiento en grupos pequeños o en procesamiento por toda la clase. Un error común que se comete en la enseñanza es conceder un plazo demasiado corto para que los alumnos procesen la calidad de su colaboración, los alumnos no aprenden de las experiencias sobre las que no reflexionan, si los grupos de aprendizaje han de funcionar mejor mañana de lo que funcionan hoy, los alumnos tiene que recibir retroinformación, tiene que reflexionar sobre cómo podrían ser más eficaces sus acciones y proyectar como ser aun competentes en la siguiente sesión en grupo.

Si los contenidos a trabajar en un determinado ABP se pueden subdividir en dos o tres o cuatro o cinco o seis temas no secuenciales, entonces es recomendable usar técnicas de especialistas, que se describe a continuación:

Sesión No. 1. A cada grupo original se le entrega información separada con cada uno de los subtemas, cada alumno se hace responsable de un subtema, lo lee y desarrolla individualmente en el grupo las actividades que se han planificado para ellos.

Sesión No. 2. Los alumnos que tiene el mismo subtema forman nuevos grupos de trabajo, analizan la información ya trabajada en la sesión Nº y profundizar aún más este tema ayudando de la nueva información que le entregara el docente, esta nueva información puede ser por ejemplo, resolución de problemas, un estudio más a fondo del tema, entre otros. Se recomienda en este momento evaluar a los alumnos para que usted como docente pueda corroborar que el alumno es un especialista en este tema.

Sesión Nº3. Los alumnos vuelven a su grupo de trabajo original e intercambian los conocimientos adquiridos en cada subtema, usted como docente tendrá que ir observando que todos los miembros del grupo vayan entendiendo cada uno de los temas trabajados por los especialistas.

Sesión Nº4. Es recomendable preparar una actividad donde el grupo tenga que relacionar todo el tema y de esta forma reafirmar el conocimiento que usted necesita que ellos adquieran para el logro de sus objetivos como para la resolución del problema original.

3.1.6 Evaluación en el ABP

El usar un método como el ABP conlleva también a usar diferentes formas de evaluar, y en este sentido los docentes tendrán que abordar diferentes alternativas. El uso pruebas convencionales cuando se ha expuesto a los alumnos a una experiencia de aprendizaje activo genera en ellos confusión y frustración, y por lo tanto se espera que cada evento evaluativo pueda cubrir los siguientes aspectos: (1) el conocimiento que el alumno pueda aportar al proceso de razonamiento grupal; y (2) las interacciones personales son los demás miembros del grupo. A partir de estos dos ejes temáticos, emerge la necesidad que el estudiante desarrolle la capacidad de evaluarse a sí mismo; evaluar a sus compañeros; evaluar al docente o facilitador; evaluar el proceso de trabajo del grupo y de los resultados obtenidos.

El propósito de estas evaluaciones es proveer al alumno de retroalimentación específica de sus fortalezas y debilidades, de tal modo que pueda aprovechar posibilidades y rectificar las deficiencias identificadas. La retroalimentación juega aquí un papel fundamental, ya que debe hacerse de manera regular, y su ejecución es de responsabilidad del docente. La retroalimentación no debe tener un sentido positivo o negativo, más bien debe tener un propósito descriptivo, identificando y aprovechando todas las áreas de mejora posibles.

Diferentes modelos de evaluación en el ABP

El proceso de enseñanza–aprendizaje es diferente en el ABP del proceso enseñanza convencional, y por lo anterior, la evaluación del alumno se convierte en un dilema para el docente, porque más que centrarse sobre hechos, en el ABP se fomenta un aprendizaje activo y un auto aprendizaje, por lo que los alumnos definen sus propias tareas de aprendizaje. Por lo mismo, los múltiples propósitos del ABP traen como consecuencia la necesidad de una variedad de técnicas de evaluación. A continuación se describe brevemente algunas formas de evaluación que se aplican en el proceso de ABP, que fueron extraídas desde el instructivo elaborado por el Instituto Profesional Virginio Gómez, y que se presenta en la Tabla 2:

Tabla 2 Algunas técnicas de evaluación empleadas en el ABP.

Nombre de la técnica	Descripción
Pruebas escritas	Pueden ser aplicadas a libro cerrado o a libro abierto, las preguntas deben ser diseñadas para garantizar la transferencia de habilidades a problemas o temas similares.
Pruebas practicas	Son utilizados para garantizar que los alumnos son capaces de aplicar habilidades aprendidas durante el curso.
Mapas conceptuales	Los alumnos representan su conocimiento y crecimiento cognitivo a través de la creación de relaciones lógicas entre los conceptos y su representación gráfica.
Evaluación del compañero	Se le proporciona al alumno una guía de categoría de evaluación que le ayuda al proceso de evaluación del compañero, este proceso también enfatiza, el ambiente cooperativo del ABP.
Autoevaluación	Permite al alumno pensar cuidadosamente acerca de lo que sabe, de lo que no sabe y de lo que necesita saber para cumplir determinadas tareas.
Evaluación al tutor	Consiste en retroalimentar al tutor acerca de la manera

Nombre de la técnica	Descripción
	en que participo con el grupo, puede ser dada por el grupo o por un observador externo.
Exposiciones	El ABP proporciona a los alumnos una oportunidad para practicar sus habilidades de comunicación, las exposiciones son el medio por el cual se pueden observar estas habilidades.
Informe escrito	Permiten a los alumnos practicar la comunicación por escrito.
Maquetas	Permite a los alumnos representar la solución de su problema a través de elementos visibles y tangibles que los ayudan a comprender mejor aquello que tiene que aprender.

Debido a las limitaciones que presenta la evaluación tradicional, la valorización integral está tomando fuerza, ya que cumple con cuatro objetivos principales para el proceso de aprendizaje: (1) constituirse en parte integral de dicho proceso; (2) suministrar al alumno información precisa sobre áreas en las que debe mejorar; (3) indicar al docente los cambios que debe realizar con el fin de mejorar sus métodos; y (4) y ofrecer a la institución información sobre los avance del alumno.

Dentro del sistema de la valoración integral, las métricas de valoración como las rúbricas se están utilizando cada día más, y por medio de ellas se facilita la calificación del desempeño del estudiante en áreas que incluyen matemáticas, ciencias, escritura, lenguas extranjeras, y otras disciplinas.

La matriz de valoración es útil como apoyo para otorgar una calificación, ya que le permite al docente que muestre a los alumnos los diferentes niveles de logro que se pueden alcanzar en un trabajo, e indicarles específicamente lo que deben hacer para alcanzar los niveles más altos, de tal manera que como instrumento de evaluación permite a los docentes hacer una apreciación justa e

imparcial de los trabajo de sus estudiantes mediante una escala que proporciona una medida clara de las habilidades y del desempeño de los alumnos.

3.1.7 Desventajas que presenta ABP

A pesar de todas las bondades pedagógicas que conlleva la implementación en el aula de esta estrategia de enseñanza, existen algunas barreras que ponen en jaque una correcta puesta en marcha de esta metodología, que se revisan a continuación:[49]

Transición difícil. Esto se explica fundamentalmente porque tanto los alumnos como los profesores tienen que cambiar la visión que tienen ellos de lo que representa el acto educativo, sobre todo por las actividades que impone esta metodología que no son propias dentro de un ambiente de enseñanza-aprendizaje tradicional.

Modificación curricular. El trabajo con problemas supone en cierta medida el establecimiento de una mayor integración con los saberes propios de otras disciplinas, con el fin de lograr una globalidad unificada de conocimientos. Esto obliga a profundizar más en torno a las relaciones que existen entre las distintas asignaturas que componen el plan de estudios de la carrera.

Tiempo. En el ABP, la adquisición de conocimientos no se da de manera tan rápida como pudiera ser con la mirada tradicional, porque los alumnos requieren de más tiempo para poder lograr sus metas de aprendizaje, y además el profesor va a tener que brindar soporte y retroalimentación a los estudiantes que estén rezagados con respecto al resto de sus pares.

El profesor ya no es el centro de la clase. Es importante que lo profesores comprendan que bajo este enfoque, los alumnos pasarán a ser el centro de la

[49] Ideas extraídas desde Dirección de Investigación y Desarrollo Educativo. (2000). *El Aprendizaje Basado en Problemas…*

clase y no ellos, como ocurre de la manera convencional, sobre todo porque la inercia de querer exponer conceptos es muy fuerte.

3.2 Conceptos de programación

En este apartado se verán algunos elementos que configuran parte del sustento teórico que todo Ingeniero Civil Informático formado en la Universidad San Sebastián debiera poseer. En particular, aquello que nos va a ocupar durante el transcurso de esta investigación está relacionado con la determinación de las principales falencias que incurren los profesores que están a cargo de dictar «*Diseño y Construcción de Software*» en cuanto a la enseñanza de la programación bajo el paradigma de la orientación a objetos. Este capítulo continúa haciendo una revisión de los principales aspectos que conforman el marco teórico sobre el cual se fundamenta la programación y luego se discutirá en torno a cómo se ha ido desarrollando la didáctica de la algorítmica.

3.2.1 Algoritmos

Una de las acepciones que trae el Diccionario de Real Academia de la Lengua Española con respecto a la palabra problema es «*planteamiento de una situación cuya respuesta desconocida debe obtenerse a través de métodos científicos*». Con miras a lograr esa respuesta, un problema se puede definir como una situación en la cual se trata de alcanzar una meta y para lograrlo se deben hallar y utilizar unos medios y unas estrategias. Juan Carlos López identifica algunos elementos que tienen en común: (1) estado inicial; (2) meta, que apunta a lo que se quiere lograr; (3) conjunto de recursos y reglas que determinan lo que está permitido hacer y/o utilizar; (4) y un dominio, el estado actual de conocimientos, habilidades y energía de quien va a resolverlo. Casi todos los problemas requieren

en quien los resuelva que sea capaz de dividirlo en submetas que, cuando son dominadas lleven a alcanzar el objetivo deseado.[50]

Cada ámbito disciplinario determina ciertas estrategias específicas para resolver problemas que sean propios de su campo de estudio. Por ejemplo: resolver problemas matemáticos implica utilizar estrategias propias de las matemáticas como uso de sistemas de ecuaciones, ecuaciones diferenciales, métodos numéricos, etc. Sin embargo, algunos psicólogos opinan que es posible utilizar con éxito estrategias generales que sean útiles para resolver problemas en muchas áreas. A través del tiempo, la humanidad ha utilizado diversas estrategias generales para resolver problemas, entre los cuales podemos destacar algunos como el de ensayo y error y los algoritmos. El primero consiste en actuar hasta que algo funcione, aunque puede demandar bastante tiempo hasta alcanzar la solución deseada, o puede que incluso jamás se pueda llegar. Su uso es apropiado cuando el conjunto solución abarca pocos elementos y es posible chequearlos todos, pudiendo empezar el proceso de iteración desde aquella solución que ofrezca la mayor probabilidad de poder resolver el problema. En cambio, los algoritmos se orientan a aplicar adecuadamente una serie de pasos detallados que aseguren una solución correcta del mismo.

Luego de analizar detalladamente el problema hasta entenderlo completamente, se procede a diseñar un algoritmo que lo resuelva por medio de pasos sucesivos y organizados en secuencia lógica. La idea de algoritmo nos establece que se trata de un conjunto de procedimientos y reglas que se deben seguir en un orden determinado para poder brindarle una solución. [51]

---

[50] López, J. C. (2009). *Algoritmos y Programación: Guía para Docentes.* Quito. Fundación Gabriel Piedrahita Uribe.

[51] La misión de un Ingeniero Civil Informático no es sólo ser eficaz en cuanto al planteamiento del algoritmo que orientará la resolución de un problema, sino que además debe ser hecha de manera eficiente tanto en el tiempo que demande su ejecución como en el espacio físico que requiera para almacenar los valores temporales que arrojen sus procesos internos. Durante las dos asignaturas que se estudian, este punto no se ve en profundidad, ya que es necesario que los estudiantes adquieran mayor familiaridad con las ideas del razonamiento algorítmico antes de aplicar alguna estrategia que les permita mejorar sus implementaciones, y profundizar aún más en matemática discreta, lo que recién se empieza a estudiar en el curso «*Especialidad I*».

El concepto de algoritmo se emplea en innumerables situaciones cotidianas de las que muchas veces no tenemos plena conciencia, como por ejemplo: la secuencia de actividades que hacemos por la mañana, que va desde abrir los ojos hasta salir del hogar para dirigirse al lugar de trabajo; seguir una receta de cocina; el proceso de digestión de los alimentos que consumimos a diario; cómo se suceden las estaciones del año; los ciclos cósmicos, entre otros.

En el ámbito de la computación, los algoritmos son una herramienta que permite describir claramente un conjunto finito de instrucciones, ordenadas secuencialmente y libres de ambigüedad, que debe llevar a cabo un computador para lograr un resultado previsible. [52] En términos generales, un algoritmo debe ser realizable — que debe terminar después de una cantidad finita de pasos —, comprensible — ser claro lo que hace, de forma que quien ejecute los pasos descritos en el cuerpo del mismo tenga claridad absoluta sobre cómo y cuándo hacerlo —, y preciso, es decir, que ante los mismos datos de entrada, genere siempre la misma salida, acorde con la definición de determinismo que proporciona Rosa Guerequeta. [53]

Estructura de un algoritmo

Independiente de cuál sea la implementación final que se use para codificarlos como producto de software, todo algoritmo se compone de variables, constantes, operadores lógicos y aritméticos, ciclos iterativos y las estructuras de control. Mediante estos elementos podemos estructurar cualquier solución algorítmica, constituyendo así su estructura básica. Cuando se estudien los lenguajes de programación, habremos de incluir otros ingredientes que por ahora no son necesarios de tener en cuenta, como por ejemplo la sintaxis, que prescribe el modo en que se tienen que ir declarando las sentencias dentro del cuerpo de cualquier programa.

[52] Ibíd. Página 21.
[53] Guerequeta, R. (2002). *Técnicas de diseño de algoritmos.* Málaga. Universidad de Málaga.

Variables y constantes. Las variables corresponden a un grupo de datos que pueden alterarse durante la ejecución del algoritmo, y se componen de tres partes:

(a) Nombre, que permite identificarla del resto.

(b) Tipo, que corresponde al dominio al cual pertenece, que puede ser numérica o alfanumérica. Dentro de los tipos de variables que podemos encontrar, se encuentran los contadores y acumuladores. Los contadores — como su nombre lo indica — se emplean para contar el número de veces que alguna instrucción (o grupo de ellas) se ejecuta. El ejemplo tradicional corresponde a las variables que desempeñan el rol de índices para recorrer algún vector o matriz. Por otro lado, los acumuladores corresponden a variables cuyo valor se incrementa o decrementa en un valor que no tiene por qué ser fijo (en cada iteración de un bucle). Un acumulador suele utilizarse para guardar progresivamente los resultados producidos en las iteraciones de algún ciclo iterativo.

(c) Valor, que debe ser válido para el tipo al cual pertenezca la variable.

Una constante, en cambio, corresponden a datos que, luego de ser asignados, no cambian en ninguna instrucción del algoritmo. Pueden contener constantes matemáticas (π) o generadas para guardar valores fijos (3.8, «*Jorge*», etc.).

Operadores lógicos y aritméticos. Los operadores son símbolos que sirven para manipular datos y variables. Los operadores y las operaciones que se pueden realizar con ellos se clasifican en:

(a) Aritméticos, que posibilitan las operaciones entre datos de tipo numérico y dan como resultado otro valor de tipo numérico. Ejemplo: potencia (∧); producto (*); división (/); suma (+); resta (-); asignación (←) o bien =.

(b) Alfanuméricos, que permiten operar con datos de tipo alfanuméricos. La mayoría de los lenguajes de programación admiten el operador + para realizar la concatenación (unión) de caracteres o cadenas.

(c) Relacionales, que se emplean para comparar datos del mismo tipo, y dan como resultado dos valores posibles: verdadero o falso. Ejemplo: igual a (==); menor que (<); mayor que (>); menor o igual (≤) o mayor o igual (≥).

(d) Lógicos, que posibilitan la evaluación lógica de dos expresiones de tipo lógico. Dan como resultado uno de dos valores posibles: Verdadero o Falso, como pueden ser: negación (no); conjunción (y); disyunción (o).

Ciclos iterativos. Los ciclos iterativos existen por la noción de que los computadores están fabricados para ejecutar tareas repetidamente. Los cálculos simples o la manipulación de pequeños conjuntos de datos se pueden realizar fácilmente a mano, pero las tareas grandes o repetitivas son realizadas con mayor eficiencia por una máquina computante. A este conjunto de sentencias se denomina ciclo iterativo, bucle o lazo, y el modo en el cual se controla el proceso de repetición consiste en ejecutar lo que esté especificado al comienzo de éste, mientras la condición dada sea verdadera. Los ciclos más usados son:

Mientras (while). La estructura repetitiva «*mientras*» (en inglés while) es aquella en que el cuerpo del bucle se repite mientras se cumple una determinada condición.

Desde/para (for). En estas, se conoce de antemano el número de veces que se desean ejecutar las acciones del bucle. Entonces, en aquellos casos donde el número de iteraciones sea fijo, se debe usar la estructura «*para*».

Estructuras de control. Las estructuras de control son mecanismos que un lenguaje requiere para realizar dos tareas indispensables: (1) seleccionar una opción entre algunos flujos de control alternativos; y (2) iterar la ejecución de una serie de sentencias. Una sentencia de selección provee al programa de medios para elegir entre dos o más «caminos» de ejecución. Su forma más sencilla es el condicional «*si (if)*» que selecciona entre dos opciones y puede extenderse a una selección prácticamente infinita mediante

constructores múltiples (como «*switch-case*») o la anidación [54] de varios *if*. De hecho, las sentencias de iteración representan el verdadero poder del cómputo, ejecutan un bloque de sentencias desde cero hasta un número teóricamente infinito de veces.

Representación de algoritmos

Las dos formas tradicionales de representación de un algoritmo corresponden a pseudocódigo y al diagrama de flujo.

Pseudocódigo. El pseudocódigo es una descripción de un algoritmo que utiliza las convenciones estructurales de un lenguaje de programación verdadero, pero que está diseñado para la lectura humana en lugar de la lectura en máquina. Normalmente, el pseudocódigo omite detalles que no son esenciales para la comprensión humana del algoritmo, tales como declaraciones de variables, código específico del sistema o algunas subrutinas propias del sistema operativo de la máquina donde se esté trabajando.

El lenguaje de programación se complementa — donde sea conveniente — con descripciones detalladas en lenguaje natural, o con notación matemática compacta. Una de las ventajas del uso de pseudocódigo es que éste es más fácil de entender para las personas que el código de lenguaje de programación convencional, ya que es una descripción eficiente y con un entorno independiente de los principios fundamentales de un algoritmo. Se utiliza comúnmente en los libros de texto y publicaciones científicas donde se documentan varios algoritmos, y también en la planificación del desarrollo de programas computacionales, para esbozar la estructura del programa antes de realizar el proceso de codificación propiamente tal. La estructuración de un programa escrito en este formato sigue dos partes:

[54] La anidación consiste en integrar estructuras de control dentro de otras.

Cabecera. En la cabecera indicamos cuáles son las variables que requiere el algoritmo para que pueda operar, y cuáles son los resultados que va a generar al usuario. En esta sección podemos hacer definiciones literales de las variables que vayamos a emplear como datos principales de entrada y de salida. Por ejemplo: si queremos especificar que *r* es la variable que representa el radio de una circunferencia, en la cabecera se escribe la sentencia *r* :=«*radio de la circunferencia, r*∈ $\mathbb{R}^+$». Es importante destacar que el uso de la notación «*:*=» es para significar «*definido como*», y que estamos indicando que *r* es una variable que pertenece al conjunto de los reales positivos.

Cuerpo. El cuerpo contiene todas las rutinas que se van a ejecutar empleando las variables definidas en la cabecera, con la posible agregación de otras que desempeñen un rol temporal, como por ejemplo alguna de tipo contador. Podemos encontrar:

(1) Instrucciones de asignación de la forma *x* = *y*, que significa que el valor que tome la variable *x* debe ser igual al de la variable *y*; operaciones matemáticas como *s* = *(s* + *x)*, entendida como que el nuevo valor de la variable*s* debe ser igual al antiguo valor que tenía más la variable *x*.

(2) Instrucciones secuenciales que se deben ejecutar de arriba hacia abajo, que pueden ser asignaciones de variables, condicionales o de flujo.

Diagrama de flujo. Un diagrama de flujo es una representación gráfica de un algoritmo o proceso. Estos diagramas utilizan símbolos con significados bien definidos que representan los pasos del algoritmo, y representan el flujo de ejecución mediante flechas que conectan los puntos de inicio y de fin de cada uno de los subprocesos involucrados. La estandarización de los símbolos para la elaboración de diagramas de flujo tardó varios años, y fue así como en 1985 la International *Organization for Standardization* — ISO — y el *American*

National Standards Institute — ANSI —, determinaron cuáles eran aquellos símbolos que tenían mayor aceptación, y que se enseñan hasta el día de hoy. Una de las ventajas de emplear esta clase de representaciones gráficas es que favorece la comprensión del proceso al mostrarlo como un dibujo, puesto que un diagrama de flujo bien construido es capaz de reemplazar varias páginas de texto debido a su poder de expresividad.

3.2.2 Lenguajes de programación

Uno de los productos que se le exigen a los profesionales del área de la Computación y de la Informática es un software o aplicación que sirva para resolver algún problema. Pues bien, para llegar a esa solución, lo que se hace es partir haciendo el diseño de la solución — expresado en un algoritmo en pseudocódigo o en diagrama de flujo — y luego se debe implementar. La herramienta que permite hacer la mediación entre el diseño de la solución y el producto final es lo que denominamos «*lenguaje de programación*». Se trata de un idioma artificial que se ha creado para traducir las ideas contenidas en un algoritmo al lenguaje propio de la máquina donde se va a desarrollar el software. El «código fuente» es un conjunto de líneas de texto que corresponde al conjunto de instrucciones que debe seguir la máquina para poder ejecutar el programa.

Por tanto, el código fuente es el documento oficial que describe por completo su funcionamiento. Entenderemos en esta investigación que en el contexto computacional la palabra «*programación*» alude al proceso mediante el cual se escribe, prueba y depuran las instrucciones contenidas en el código fuente.

Clasificación de los lenguajes de programación

Aun cuando existen varios criterios para clasificar los lenguajes de programación, nos inclinaremos por el que expone Isabel Gallego en el texto «*Algorítmica y*

programación para ingenieros» en relación a la cercanía que tenga con el hardware de la máquina. En esta línea, encontramos los lenguajes de alto nivel y los de bajo nivel. Los primeros son aquellos en los que las instrucciones son descritas en un lenguaje próximo al empleado por los seres humanos. Permiten la ejecución del programa en computadores muy diversos, y requieren su traducción a las características de cada máquina donde se vaya a trabajar el código fuente, mediante un proceso que se llama «compilación». Algunos ejemplos de este tipo son: Java, C, C++, C#, LISP, Visual Basic, PHP, Fortran, Pascal, entre otros. Y por otro lado, los de bajo nivel se caracterizan porque las instrucciones difícilmente son interpretables por las personas, y el proceso de traducción al lenguaje de la máquina es más rápido que en los primeros. *Assembler* es uno de los lenguajes más emblemáticos en esta categoría. En la actualidad estos últimos se emplean para generar los controladores de dispositivos que se conectan a un computador[55], útiles para manipular el conjunto de microinstrucciones que permiten lograr que, por ejemplo, una impresora funcione correctamente en el computador que usamos en nuestro hogar o en el trabajo. [56]

3.2.3 Paradigmas de programación

Héctor Zárate señala que probablemente fuera Robert Floyd[57] quien introdujera por primera vez la noción de «*paradigmas de programación*», quien los definió como «*un proceso de diseño que va más allá de una gramática, reglas semánticas y algoritmos, sino que es un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de programas*».[58]Todas las aplicaciones computacionales

[55] Los denominados *drivers*.
[56] Gallego, I. (2000). *Algorítmica y programación para ingenieros*. Barcelona. Edicions UPC.
[57] Robert Floyd (1936-2001) fue uno de los científicos estadounidenses más brillantes en el campo de la ciencia informática, donde sus aportes más emblemáticos fueron la determinación de un algoritmo para encontrar el camino más corto en un grafo, y la introducción del concepto «*invariante*» para demostrar la correctitud lógica de programas iterativos, los que a la larga le hicieron merecedor del Premio Turing en 1978, que es el galardón más importante conferido por la *Association for Computing Machinery* a quien se haya destacado de manera trascendental en el campo de las ciencias computacionales.
[58] Zárate, H. (2008). *Paradigmas de Programación*. Ciudad de México. Universidad Nacional Autónoma de México, Facultad de Ingeniería.

que se ocupan en el diario vivir han requerido del desarrollo de una gran cantidad de lenguajes de programación, cada uno con propósitos y formas distintas en cuanto a su concepción de representar las instrucciones que finalmente le darán vida a los programas que se generen. El criterio que se denomina paradigma también se puede emplear para clasificar lenguajes de programación. Al respecto, Zárate identifica cuatro enfoques, que son los más aceptados en la actualidad: (1) paradigma imperativo; (2) paradigma funcional; (3) paradigma lógico; y (4) paradigma de orientación a objetos.

Paradigma imperativo. Este paradigma debe su nombre a la concepción que tenemos del término imperativo, que concibe a un programa de computador como una serie de comandos que una máquina será capaz de ejecutar. Estas órdenes detallan de forma clara y específica el cómo hacer las cosas, y llevarán al programa a través de distintos estados, los que quedan plenamente identificados de acuerdo a los valores temporales que vayan tomando las variables en tiempo de ejecución. La definición de este paradigma sería imposible sin este elemento, ya que los estados de un programa son representados y diferenciados por su conjunto de variables y sus contenidos. Las expresiones que se utilizan para dar órdenes constituyen el núcleo de toda aplicación que se diseñe bajo este enfoque, donde las más importantes son las de asignación de variables. Algunos lenguajes que siguen esta línea son: C, Fortran, Pascal, Perl, PHP y Java, entre otros.

Paradigma funcional. La programación funcional se basa en el concepto matemático de función. A diferencia del paradigma imperativo — donde el énfasis se hacía en los cambios de estado que experimentaran los valores de las variables —, el foco se centra en concebir todo el programa como si fuera una gran función. Aunque no es el estándar que se aplica en la industria, algunos de los lenguajes que se han creado bajo esta mirada encuentran aplicaciones en el terreno de la Inteligencia Artificial, como en el caso de Lisp.

Paradigma lógico. El paradigma de la lógica nos propone que ésta representa un conocimiento que puede ser manipulado con mecanismos de inferencia[59]. La diferencia que se da con los otros enfoques es que los programas que se construyan en lenguajes que admitan esta mirada es que se centran en el qué hacer y no en el cómo hacerlo. Su propósito general es que a partir de un conjunto de reglas de entrada se puedan generar proposiciones lógicas que sean de interés para el usuario final. En este sentido, los lenguajes SQL — ampliamente utilizado por los motores de bases de datos existentes — y Prolog son los más conocidos por los programadores.

Paradigma de orientación a objetos. La programación orientada a objetos — POO — se basa en gran medida en las directrices de los paradigmas imperativo y funcional, que las combina con una serie de conceptos nuevos que incitan a contemplar las tareas de programación desde un nuevo punto de vista. La POO permite descomponer más fácilmente un problema en subgrupos de partes relacionadas del problema. Entonces, utilizando el lenguaje se pueden traducir estos subgrupos a unidades llamadas «*objetos*», que se constituyen en los bloques esenciales de la construcción de cualquier programa. Un objeto está compuesto por estados y métodos. Los estados son propiedades del objeto y están representados por variables con valores únicos para cada objeto y que son llamadas «*variables de instancia*» y los métodos corresponden a las acciones que puede emprender el objeto. Una «*clase*» es una colección de objetos que comparten los mismos atributos, de tal manera que un objeto pasa a convertirse en una particularización — o instanciación — de una clase. Por ejemplo: la clase «Persona» con atributos nombre y edad y método «lavarse-los-dientes» se puede instanciar con el objeto «*José Sánchez, 31 años de edad*» cuya única función en la vida será lavarse los dientes. Otros conceptos relevantes que sedan bajo este paradigma son los de encapsulamiento[60], herencia[61] y polimorfismo[62]. Los

[59] Como por ejemplo: *implicación, conjunción, disyunción, negación, modus tollens, silogismo disyuntivo, silogismo hipotético*, entre otros. Para más detalles, consultar Grimaldi, R. (1997). Matemáticas Discreta y Combinatoria. . . Página 88.

[60] El encapsulamiento significa que el código o datos de un objeto pueden estar ocultos para cualquier entidad externa a él.

elementos principales de este paradigma son los que detallamos de manera más acuciosa por el hecho que éste es el que se emplea para la enseñanza de la programación en la asignatura «*Diseño y Construcción de Software*».

Java es, dentro de los lenguajes que siguen la visión paradigmática de la orientación a objeto, el que más destaca en este ámbito, y sobre el cual se articula el curso que ha sido investigado. La historia dice que nació en el año 1991 cuando un grupo de ingenieros de *Sun Microsystems* trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos. La reducida potencia de cálculo y memoria de éstos llevó a desarrollar un lenguaje sencillo capaz de generar código de tamaño muy reducido. Debido a los continuos cambios que experimentaban los procesadores, se instaló la necesidad de contar con algún lenguaje que pudiera generar software con independencia de la arquitectura que tuviera la unidad central de procesamiento (CPU). Entonces, se desarrolló este lenguaje que podía generar un código que sólo tuviera que ser «*entendido*» por una máquina hipotética o virtual, denominada «*Java Virtual Machine*», la cual era independiente del dispositivo en que estuviera instalada.

Hasta el día de hoy, las aplicaciones que se han desarrollado en Java pueden correr en una amplia gama de dispositivos que cuentan con toda clase de sistemas operativos y procesadores instalados, lo que le confiere un alto nivel de versatilidad, en desmedro de otros lenguajes orientados a objeto como Visual Basic, que sólo admite máquinas instaladas sobre plataformas con sistema operativo Windows, y que es de pago, a diferencia de Java que es libre. Con esto, no es de extrañar que el lema bajo el cual haya sido creado fuera: «*Write Once, Run Everywhere*». El desarrollo de cualquier aplicación en Java no parte de cero,

---

[61] La herencia es la propiedad de crear nuevos objetos a partir de la definición de otros. Un objeto «*nuevo*» será idéntico al modelo que seguimos para crearlo, excepto por algunos cambios incrementales o re-definiciones de sus estados o métodos. Por ejemplo: las clase «*Gato*» y «*Ballena*» heredan de «*Mamífero*», aunque el atributo tipo de piel sea distinto en uno y otro caso.

[62] El polimorfismo se refiere a la capacidad para que varias clases derivadas de una antecesora utilicen un mismo método de forma diferente. Por ejemplo, la clase «*Animal*» tiene el método mover, y tanto «*Pez*» como «*Ave*» heredan de «*Animal*», pero la forma en que cada una de ellas se mueve es distinta, a pesar que la acción lleve el mismo nombre en ambas clases.

sino que de inmediato el usuario puede trabajar con una serie de clases preexistentes, a las que sólo requiere instanciar cuando las vaya a requerir.[63]

3.2.4 Presentación de Java

Java es, dentro de los lenguajes que siguen la visión paradigmática de la orientación a objeto, es el que más destaca en este ámbito y sobre el cual se articulan los cursos sobre los cuales se investigará. La historia nos dice que nació en el año 1991 cuando un grupo de ingenieros de *Sun Microsystems* trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos. La reducida potencia de cálculo y memoria de éstos llevó a desarrollar un lenguaje sencillo capaz de generar código de tamaño muy reducido. Debido a los continuos cambios que experimentaban los procesadores, se instaló la necesidad de contar con algún lenguaje que pudiera generar software con independencia de la arquitectura que tuviera la unidad central de procesamiento (CPU). Entonces, se desarrolló este lenguaje que podía generar un código que sólo tuviera que ser «*entendido*» por una máquina hipotética o virtual, denominada «*Java Virtual Machine*», la cual era independiente del dispositivo en que estuviera instalada.

Hasta el día de hoy, las aplicaciones que se han desarrollado en Java pueden correr en una amplia gama de dispositivos que cuentan con toda clase de sistemas operativos y procesadores instalados[64], lo que le confiere un alto nivel de versatilidad, en desmedro de otros lenguajes orientados a objeto como Visual Basic, que sólo admite máquinas instaladas sobre plataformas con sistema operativo Windows, y que es de pago, a diferencia de Java que es libre. Con esto, no es de extrañar que el lema bajo el cual haya sido creado fuera: «*Write Once, Run Everywhere*».

[63] García, J. (2000). *Aprenda Java como si estuviera en primero.* San Sebastián. Escuela Superior de Ingenieros Industriales, Universidad de San Sebastián. Página 1.
[64] Como por ejemplo: tener algún PC con Windows, Linux, teléfonos móviles, computadores MAC, etc.

El desarrollo de cualquier aplicación en Java no parte de cero, sino que de inmediato el usuario puede trabajar con una serie de clases preexistentes, a las que sólo requiere instanciar cuando las vaya a requerir. [65]

3.3 Software educativo orientado a la enseñanza de la programación

3.3.1 Software educativo

Como se ha comentado en el marco teórico de la investigación, la programación constituye una habilidad muy apetecida, sobre todo para un profesional de la Ingeniería. Es fácil observar cómo la influencia que tienen los computadores hoy en día ha llegado a tal punto que en los últimos años, la demanda de programadores se ha ido incrementado, junto al interés de los estudiantes por estudiar carreras ligadas con la programación, donde esta habilidad incluso sirve de soporte para la emulación de procesos físicos o químicos, de acuerdo a lo que sostienen autores como Nieves Carralero[66] o Jorge Giraldo[67].

Pero, aprender a programar no es una tarea fácil. Muchos programadores principiantes se enfrentan a una amplia gama de dificultades y deficiencias, lo que ha causado que se considere a los cursos de programación como una labor difícil. Lo anterior no es fácil negar si se considera que estos cursos con frecuencia son los que muestran las tasas más altas de deserción y de reprobación, tal como lo señala Sócrates Torres[68].

---

[65] García, J. (2000). *Aprenda Java como si estuviera en primero.* San Sebastián. Escuela Superior de Ingenieros Industriales, Universidad de San Sebastián. Página 1.
[66] Carralero, N. (2011*). Entornos para enseñar programación en secundaria. Nuevos enfoques.* Castilla La Mancha: Quaderns Digitals.
[67] Giraldo, J., & Mateus, S. (2010). *Aprendizaje de la Programación Orientada a Objetos a través de la creación de juegos de video*. Medellín: Cefalea.
[68] Torres, S., & Liñan, E. (2012). *Aprende programación con Alice.* Revista CIENCIACIERTA.

Un planteamiento común en la educación de programación es enseñar primero los conceptos básicos de un lenguaje de programación, y ofrecer una guía a los estudiantes hacia las estrategias eficaces para el proceso de programación. Por lo tanto, el aprendizaje de los conceptos básicos a menudo forma la base para el desarrollo de habilidades más avanzadas. Tal como se comentó en el capítulo anterior, uno de los objetivos que tiene esta propuesta de intervención pedagógica consiste en emplear algún recurso didáctico que permita el aprendizaje de la programación orientada a objetos de una manera que al alumno le resulte divertida, y que al mismo tiempo le permita asimilar de mejor forma las diferentes estructuras sobre las cuales se construye este paradigma.

Por otra parte, Giraldo indica que las Tecnologías de la Información y la Comunicación son consideradas como aquellas herramientas que permiten la adquisición, almacenamiento, procesamiento y distribución de la información, y que de alguna manera u otra han validado su presencia en muchos entornos, en especial en aquellos que relacionan a los procesos cognitivos de los seres humanos, como el caso de simuladores, ambientes virtuales de aprendizaje, tutoriales interactivos y su uso como apoyo a los cursos dictados de manera presencial en aulas de clase, o en experiencias que se puedan practicar en laboratorios especializados.

El tipo de tecnología de información que se aplicará para efectos de la formulación de esta propuesta corresponde al software educativo, que María Vidal[69] junto a otros autores definen como una aplicación computacional cuyas características estructurales y funcionales sirvan de apoyo al proceso de enseñar, o que esté destinado también al proceso de autoaprendizaje con el cual se puedan lograr ciertas habilidades cognitivas que son propias además del empleo del aprendizaje basado en problemas como estrategia metodológica. Además, el software educativo:

- Es concebido con un propósito específico: apoyar la labor del profesor en el proceso de aprendizaje de los estudiantes.

[69] Vidal Ledo, M., Gómez Martínez, F., & Ruiz Piedra, A. (2010). Software educativos. Scielo, 97-110.

- Debe contener elementos metodológicos que orienten el proceso de aprendizaje.
- Es un tipo de programa que se elabora con el fin de ser empleados por computadores, para recrear ambientes interactivos que posibiliten la comunicación con el estudiante.
- Debe ser sencillo de utilizar, de tal manera que los conocimientos computacionales que requieran los estudiantes sean mínimos.
- Debe ser un agente de motivación para que el alumno pueda interesarse en este tipo de recurso didáctico para que se involucre de manera significativa en el proceso de enseñanza-aprendizaje.

Estructura

En términos generales, esta clase de recursos didácticos se estructuran en torno a tres grandes componentes: (1) interfaz; (2) pedagógico; y (3) técnico, que se describen a continuación:

Componente interfaz. Es aquel que permite la interacción entre los usuarios y el programa, en el cual intervienen los mensajes que pueden ser entendidos por ambas partes, así como los dispositivos de entrada y salida de datos y las zonas de comunicación disponibles para el intercambio de datos e instrucciones. Este componente se desglosa en dos niveles: (1) programa-usuario, que permite la comunicación entre el computador y el usuario, a través dispositivos de entrada y salida como pantalla, impresora, sintetizador de voz, etc.; y (2) usuario-programa, que guarda relación con los modos en que el participante se comunica con la máquina, siendo el teclado y el ratón los periféricos más empleado en estas oportunidades, para introducir comandos y respuestas intermedias durante el proceso de aprendizaje. Dentro de los elementos constitutivos de las zonas de comunicación, se incluyen los sistemas de menús, las características de los textos que posibiliten una disposición estética y efectiva, y elementos visuales como gráficos, animaciones y videos.

Componente pedagógico. Es el que determina los objetivos de aprendizaje que se lograrán al finalizar el empleo del software, los contenidos a desarrollar con el programa en función a los objetivos educacionales, las secuencias de la instrucción, los tipos de aprendizajes que se quieren lograr, sistemas de evaluación que se deben considerar para determinar los logros y los sistemas de motivación extrínseca e intrínseca que se deben introducir.

Componente técnico. Permite establecer la estructura lógica para la interacción para que el software cumpla con las acciones requeridas por el usuario, así como ofrecer un ambiente al estudiante para que pueda aprender lo deseado y servir de entorno. A la estructura lógica del programa se liga íntimamente la estructura de datos, que organiza la información necesaria para que el software pueda cumplir con sus objetivos instruccionales. El algoritmo que se emplee determinará el tipo de ambiente de aprendizaje, y la interacción del programa.

Tipos de programas educativos

Los programas educativos a pesar de tener unos rasgos esenciales básicos y una estructura general común se presentan con unas características muy diversas. Se han elaborado múltiples tipologías que clasifican los programas didácticos a partir de diferentes criterios, que han sido adaptados desde el portal de Pere Marquès.[70]

Uno de estos se basa en la consideración del tratamiento de los errores que cometen los estudiantes, distinguiendo: (1) programas tutoriales directivos, que hacen preguntas a los estudiantes y controlan en todo momento su actividad; y (2) programas no directivos, en los cuales la máquina adopta el papel de un laboratorio o instrumento a disposición de la iniciativa de un alumno que pregunta y tiene una libertad de acción sólo limitada por las normas del programa. La computador no juzga las acciones del alumno, se limita a procesar los datos que éste introduce y a mostrar las consecuencias de sus acciones sobre un entorno.

[70] Marquès, P. (10 de Enero de 1996). *El software educativo.* Recuperado el 9 de Noviembre de 2012, de El software educativo: http://www.lmi.ub.es/te/any96/marques_software/#capitol7

Objetivamente no se producen errores, sólo desacuerdos entre los efectos esperados por el alumno y los efectos reales de sus acciones sobre el entorno. Otra clasificación el grado de control del programa sobre la actividad de los alumnos y la estructura de su algoritmo, que es la que se presenta a continuación.

Programas tutoriales. Son programas que en mayor o menor medida dirigen el trabajo de los alumnos. Pretenden que, a partir de unas informaciones y mediante la realización de ciertas actividades previstas de antemano, los estudiantes pongan en juego determinadas capacidades y aprendan o refuercen unos conocimientos y/o habilidades. En cualquier caso, son programas basados en los planteamientos conductistas de la enseñanza que comparan las respuestas de los alumnos con los patrones que tienen como correctos y guían los aprendizajes de los estudiantes.

Bases de datos. Proporcionan unos datos organizados, en un entorno estático, según determinados criterios, y facilitan su exploración y consulta selectiva. Se pueden emplear en múltiples actividades como por ejemplo: seleccionar datos relevantes para resolver problemas, analizar y relacionar datos, extraer conclusiones, comprobar hipótesis, etc. Las bases de datos pueden tener una estructura: (1) jerárquica, si existen unos elementos subordinantes de los que dependen otros subordinados, como los organigramas; (2) relacional, si están organizadas mediante unas fichas o registros con una misma estructura y rango; y (3) documental, si utiliza descriptores y su finalidad es almacenar grandes volúmenes de información documental: revistas, periódicos, etc. Por otra parte, dependiendo de la manera en cómo se acceda a la información, se pueden distinguir dos tipos: (1) bases de datos convencionales, que tienen la información almacenada en archivos, mapas o gráficos, que el usuario puede recorrer según su criterio para recopilar información; (2) bases de datos de tipo sistema experto, que guardan y procesan información altamente especializada sobre un tema en particular, y que tienen la particularidad de poder asesorar al usuario cuando busque alguna respuesta determinada.

Simuladores. Presentan un modelo o entorno dinámico (generalmente a través de gráficos o animaciones interactivas), y facilitan su exploración y modificación a los alumnos, que pueden realizar aprendizajes inductivos o deductivos mediante la observación y la manipulación de la estructura subyacente; de esta manera pueden descubrir los elementos del modelo, sus interrelaciones, y pueden tomar decisiones y adquirir experiencia directa delante de unas situaciones.

También se pueden considerar simulaciones ciertos videojuegos que estimulan la capacidad de interpretación y de reacción ante un medio concreto. En cualquier caso, posibilitan un aprendizaje significativo por descubrimiento y la investigación de los estudiantes/experimentadores.

Se pueden diferenciar dos tipos de simulador: (1) modelos físico-matemáticos, que presentan de manera numérica o gráfica una realidad que tiene unas leyes representadas por un sistema de ecuaciones deterministas. En esta categoría se incluyen los programas-laboratorio, algunos trazadores de funciones y los programas que mediante un convertidor analógico-digital captan datos analógicos de un fenómeno externo al computador y presentan en pantalla un modelo del fenómeno estudiado o informaciones y gráficos que van asociados; y (2) entornos sociales, que presentan una realidad regida por unas leyes no del todo deterministas. Se incluyen aquí los juegos de estrategia y de aventura, que exigen una estrategia cambiante a lo largo del tiempo.

Constructores. Son programas que tienen un entorno programable y que le permiten a los estudiantes construir elementos más complejos o entornos a partir de elementos simples. De esta manera potencian el aprendizaje heurístico y, de acuerdo con las teorías cognitivistas, facilitan a los alumnos la construcción de sus propios aprendizajes. Los constructores específicos corresponden a una de sus posibles clasificaciones, lo que ponen a disposición de los estudiantes una serie de mecanismos de actuación (generalmente en forma de órdenes específicas) que les permiten llevar a cabo operaciones de un cierto grado de complejidad mediante la construcción de determinados entornos, modelos o

estructuras, y de esta manera avanzan en el conocimiento de una disciplina o entorno específico.

Lenguajes de programación. Este tipo de programas ofrecen una suerte de *laboratorio simbólico* que permiten construir diferentes entornos de trabajo, como en el caso de Logo, Pascal o Basic, con los cuales se busca lograr que el alumno pueda hacer una manipulación concreta y práctica en un entorno informatizado que facilita la representación y comprensión del espacio y la previsión de los movimientos.[71]

Programas herramienta. Son programas que proporcionan un entorno instrumental con el cual se facilita la realización de ciertos trabajos generales de tratamiento de la información: escribir, organizar, calcular, dibujar, transmitir, captar datos, etc. Son programas de uso general que provienen del mundo laboral y, por tanto, quedan fuera de la definición que se ha dado de software educativo. No obstante, se han elaborado algunas versiones de estos programas «*para niños*» que limitan sus posibilidades a cambio de una, no siempre clara, mayor facilidad de uso. De hecho, muchas de estas versiones resultan innecesarias, ya que el uso de estos programas cada vez resulta más sencillo y cuando los estudiantes necesitan utilizarlos o su uso les resulta funcional aprenden a manejarlos sin dificultad.

Los programas más utilizados de este grupo son: (1) procesadores de texto, que permite que los estudiantes pueden concentrarse en el contenido de las redacciones y demás trabajos despreocupándose por la caligrafía. Otra ventaja es que el corrector ortográfico les ayudará a revisar posibles faltas de ortografía antes de entregar el trabajo. Además de este empleo instrumental, los procesadores de textos permiten realizar múltiples actividades didácticas, como por ejemplo ordenar párrafos, versos, estrofas, insertar frases o completar textos; (2) gestores

[71] Logo es un programa creado en 1969 por Seymour Papert que tiene una doble dimensión: (1) proporciona entornos de exploración donde el alumno puede experimentar y comprobar las consecuencias de sus acciones, de manera que va construyendo un marco de referencia, unos esquemas de conocimiento, que facilitarán la posterior adquisición de nuevos conocimientos; y (2) que facilita una actividad formal y compleja, próxima al terreno de la construcción de estrategias de resolución de problemas: la programación. A través de ella los alumnos pueden establecer proyectos, tomar decisiones y evaluar los resultados de sus acciones desde temprana edad.

de bases de datos, que permiten almacenar información de manera organizada y posteriormente recuperarla y modificarla; (3) hojas de cálculo, que son programas que convierten a la computador en una versátil y rápida calculadora programable, facilitando la realización de actividades que requieran efectuar muchos cálculos matemáticos. Entre las actividades didácticas que se pueden realizar con las hojas de cálculo están las siguientes: aplicar hojas ya programadas a la resolución de problemas de diversas asignaturas, evitando así la realización de pesados cálculos y ahorrando un tiempo que se puede dedicar a analizar los resultados de los problemas, o bien programar un nueva hoja de cálculo, que exige para estos casos un modelo matemático preciso para el fenómeno que se quiera implementar; (4) editores gráficos, que se emplean desde un punto de vista instrumental para realizar dibujos, portadas para los trabajos, murales, anuncios, etc. Además constituyen un recurso idóneo para desarrollar parte del currículum del área del diseño gráfico; (5) programas de comunicaciones, que permiten la transmisión de datos entre máquinas que estén alejadas entre sí, en términos físicos, con el fin de intercambiar mensajes, gráficos, programas, etc. Desde una perspectiva educativa estos sistemas abren un gran abanico de actividades posibles para los alumnos, como por ejemplo intercambiar ideas con personas que estén lejos, o poder acceder a bases de datos remotas para acceder a determinada información de interés para el profesor o el alumno; (6) programas de experimentación asistida, que a través de variados instrumentos y convertidores analógico-digitales, recogen datos sobre el comportamiento de las variables que inciden en determinados fenómenos. Posteriormente con estas informaciones se podrán construir tablas y elaborar representaciones gráficas que representen relaciones significativas entre las variables estudiadas; y (7) lenguajes de sistemas de autor, que facilitan la elaboración de programas tutoriales a los profesores que no disponen de grandes conocimientos informáticos. Algunos incluso permiten controlar vídeos y dan facilidades para crear gráficos y efectos musicales, de manera que pueden generar aplicaciones multimedia.

Ventajas del uso de software educativo

Dentro de las ventajas que tiene el uso el software educativo, destaca el factor motivacional, ya que la utilización del computador y los programas educativos genera en los estudiantes una expectativa, sobre todo en aquellos que no han tenido experiencias computacionales. Es por esto que el uso del software como recurso didáctico se ha transformado en un verdadero motor de aprendizaje, ya que incita a la actividad y al pensamiento, ya que la motivación de por sí permite que los estudiantes le otorguen mayor tiempo al trabajo de un tema concreto y logren aprender más.[72]

Por otro lado, está la interacción que se produce directamente con la máquina, y que ayuda a superar la falta de interactividad que es propia de los materiales clásicos de estudio. La introducción de los programas educativos estimula este intercambio de imágenes y sonidos entre ambos actores —estudiante y programa —, haciendo que el primero asuma un rol más activo con su proceso de aprendizaje.

Además, se sabe que todos los alumnos son personas diferentes entre sí, lo que significa que no todos aprenden de la misma manera, ya que no tienen los mismos conocimientos previos, o no han tenido experiencias similares, lo que en cierta medida dificulta al docente el logro de las metas educativas. El empleo del software educativo puede solucionar este problema, ya que a través de su uso permite individualizar el trabajo del estudiante, al hacer que éste se pueda adaptar a su ritmo de trabajo, y que al mismo tiempo le ayude a reforzar el estado de avance parcial de sus aprendizajes, lo que a la larga permite que este proceso se pueda ejecutar de una manera más flexible.

[72] Marquès, P. (10 de Enero de 1996). *El software …*

Algunos inconvenientes

Es necesario plantear algunos de los inconvenientes que supone el uso de software educativo como recurso didáctico en el aula. Por una parte, está el costo, ya que no todos son de libre distribución; además, es necesario que el estudiante cuente con acceso a algún computador para que haga uso de él, y tener conocimientos mínimos para poder ejecutarlo. Considerando que los estudiantes son nativos digitales, y dada la facilidad que existe hoy en día para acceder a algún equipo computacional, se cree que estas barreras estarían debidamente abordadas.

Además, requiere de una mayor inversión de tiempo por parte del profesor, para poder implementarlo (al menos en la fase inicial), ya que él debe conocer la herramienta que va a emplear antes de desarrollar las clases. Esto implica que el docente tenga que satisfacer ciertos requisitos mínimos de competencias tecnológicas, y también deberá hacer un trabajo adicional para que los estudiantes se empapen de las ideas del aprendizaje autónomo y colaborativo, que va en directa correspondencia con una de las ideas emblemáticas del ABP.[73]

3.3.2 Software Educativo Alice

Complementando los desafíos que conlleva enseñar a programar que indica Jorge Villalobos, Sócrates Torres y Ernesto Liñán[74] comentan que se han estado desarrollando una amplia variedad de soluciones que ayudan a simplificar esta ardua tarea, que van desde el empleo de distintos lenguajes de programación — con sus respectivos entornos de desarrollo y la ayuda que ellos brindan — hasta la concientización de los alumnos sobre la importancia que tiene el desarrollo del pensamiento algorítmico y sistémico.

[73] Algunas de estas ideas fueron extraídas y adaptadas desde el blog http://guardianrojo.blogspot.com/
[74] Torres, S., & Liñan, E. (2012). Aprende programación con Alice. CIENCIACIERTA.

Como esta propuesta apunta hacia una innovación de la praxis pedagógica en el área de la enseñanza de la programación orientada a objetos, en esta oportunidad se empleará como recurso didáctico el programa educativo Alice, que corresponde a un proyecto encabezado por la Universidad de Carnegie Mellon, y que se define como un entorno de programación 3D que facilita la generación de animaciones para contar alguna historia, crear un juego interactivo, o un video para compartir en Internet. Su propósito central se orienta a hacer divertido el proceso de enseñar a programar, de tal manera que su uso constituya una experiencia completamente novedosa con respecto a la manera tradicional en cómo se enseña la POO, y que se convierte en algo novedoso y atractivo para el principal beneficiario de esta propuesta, que es el estudiante.

Según sus creadores[75], esta herramienta ha sido diseñada con el objetivo de convertirse en la primera experiencia que tenga un estudiante durante su proceso de aprendizaje de la programación orientada a objetos, lo que coincide plenamente con la naturaleza de la asignatura propuesta, y cuyo diseño se puede encontrar en los siguientes capítulos.

En términos generales, la creación de programas computacionales se basa en arrastrar y soltar bloques gráficos en la interfaz interactiva de Alice. Cada uno de ellos corresponde a las declaraciones estándar en un lenguaje de programación orientado a objetos, tales como Java, C++, C#, PHP 5, y otros. De esta forma, el estudiante no requiere conocer en principio las reglas que rigen a un lenguaje determinado, ya que la propia interfaz lo invita a adentrarse en el uso de cada estructura de programación, de tal manera que le permita manipular inicialmente conocer todas las reglas de un lenguaje de programación para realizar programas, y sí podrá ir comprendiendo el funcionamiento y comportamiento de los diferentes elementos disponibles para cada objeto que haya integrado a su mundo virtual.

De esta manera, Alice permite a los estudiantes ver de inmediato cómo se ejecutan sus programas de animación, lo que les ayuda a comprender fácilmente

[75] Dann, W., Cooper, S., & Pausch, R. (2012). *Learning to Program with Alice*. Pittsburgh: Editorial Pearson.

la relación entre las declaraciones de la programación y el comportamiento que van experimentado los objetos en el contexto de la animación que estén desarrollando. Una de las primera barreras que se dan durante el proceso de aprendizaje de la programación tiene que ver con la ejecución del código que se haya escrito, ya que en reiteradas oportunidades es común la aparición de una serie de errores — de carácter sintáctico principalmente — que dificultan la depuración del trabajo que se esté realizando.

Los creadores de Alice comentan que los conceptos clásicos del paradigma de orientación a objetos se hacen tangibles por medio de los elementos que se incluyen en el escenario de trabajo, a medida que el usuario — que puede ser el alumno o el profesor — vaya poblando el mundo sobre el cual se construya la animación o el video juego interactivo que se esté implementando, con la ventaja que este modo de programar está exento de errores sintácticos. Además, este entorno de trabajo permite trabajar con métodos, funciones, variables, parámetros, listas, arreglos y eventos, que se constituyen en grandes rasgos en los elementos principales que se enseñan en un curso introductorio a la programación.

En su opinión, existen cuatro factores que configuran ciertos obstáculos a la hora de implementar alguna asignatura de esta naturaleza, y que a través de Alice se pueden subsanar: (1) problemas de carácter sintáctico, que se resuelven gracias a los mecanismos de arrastrar y soltar hacia el editor de código, y que todas las estructuras de programación que se puedan utilizar podrán ser empleadas en algún momento particular, con el fin de minimizar los problemas de consistencia que pudieran tener las creaciones de los estudiantes; (2) abstracción sobre el espacio de funciones y de variables: a menudo a los estudiantes se les hace difícil concebir en su esquema mental la manera en cómo evolucionan los valores que puedan tomar los elementos que forman parte de sus programas, tales como variables o resultados parciales de funciones; la intención de uso de Alice apunta a que en este caso — en lugar de trabajar con estructuras matemáticas abstracto — ellos puedan apreciar de manera explícita el cómo una clase instanciada por medio de un objeto es capaz de desplazarse por la pantalla,

con sólo invocar a los métodos que permiten tal acción, ya que la naturaleza tridimensional de esta herramienta permite que ellos puedan interactuar con las variables del mundo de una manera más abstracta, por medio de una simulación de su comportamiento; (3) la falta de motivación hacia la programación: lo importante es motivar a los estudiantes para que se interesen por la adquisición de esta habilidad, y no que tengan que aprenderla por el sólo hecho de tenerla como una obligación curricular. Es importante hacerles ver que se pueden generar aplicaciones interesantes dentro de su ámbito de especialidad, como una forma de otorgarle significancia a una asignatura de este tipo; y (4) sobre la dificultad de entender técnicas de diseño y lógica de desarrollo de aplicaciones: en Alice se busca que los programas desarrollados sean equivalentes a películas, donde los alumnos pasen a desempeñar los roles de director y guionista, ya que estas labores son semejantes a las tareas que desempeña un programador, salvo que en el primer caso el uso de este entorno de trabajo le permita orientar su aprendizaje desde otra perspectiva.

Algunos casos de uso

En un artículo escrito por Barb Maskal y otros autores[76] relatan la experiencia de haber trabajado en un proyecto de investigación educacional patrocinado por la *National Science Foundation*, que tiene como objetivo principal el desarrollo de un curso que permitiera mejorar el nivel de retención de los estudiantes que estaban tomando un curso introductorio a las Ciencias de la Computación. El estudio realizado por ellos da cuenta de la efectividad del uso de Alice como recurso didáctico para mejorar el rendimiento estudiantil. Y no sólo eso, sino que además permitió incrementar sus actitudes y predisposiciones hacia el aprendizaje de materias que son propias del campo de las Ciencias de la Computación. De hecho, ellos hicieron mediciones con algunos estudiantes que estaban en riesgo de reprobar un curso introductorio de programación, y descubrieron que aquellos a

[76] Moskal, B., Lurie, D., & Cooper, S. (2000*). Evaluating the Effectiveness of a New Instructional Approach*. Conference 2000 ACM.

los que se les había entrenado con Alice para sus exámenes finales lograron obtener mejores calificaciones que aquellos que no formaron parte del estudio.

Algo similar es lo que relatan Stephen Cooper, Wanda Dann y Randy Pausch[77], quienes llegaron a la conclusión que los lenguajes de programación que empezar a estudiar la teoría de la orientación a objetos desde el principio de una asignatura le ayuda a los estudiantes a subsanar las dificultades inherentes que conlleva el aprendizaje y la adquisición de las habilidades asociadas con este paradigma.

De manera adicional, ellos han detectado algunas fortalezas y debilidades que se puedan derivar a partir del uso de Alice como recurso didáctico.[78] Con respecto a las primeras, ellos detectaron que los estudiantes han podido desarrollar:

1. Un fuerte sentido del diseño, en el sentido que las actividades propuestas han incorporado en primer lugar el uso de pseudocódigo y de guiones textuales y gráficos[79] para escribir el libreto de alguna historia o de un videojuego interactivo, y en segundo lugar ellos pasan a trasladar sus ideas al computador, tal como ocurre en entornos reales de desarrollo, donde se empieza por el diseño, para pasar hacia las etapas posteriores de implementación.
2. Una contextualización clara para objetos, clases, y conceptos claros de orientación a objetos, que a juicio de ellos pasa a ser una de las grandes características que posee esta herramienta, simplemente porque todos los personajes y cosas que intervienen en los mundos virtuales que recrean corresponden a objetos. En estas animaciones, pueden hacer que las clases instanciadas como objetos puedan implementar sus

[77] Cooper, S., Dann, W., & Pausch, R. (2003). *Teaching Objects-first In Introductory Computer Science*. Pittsburgh: Carnegie Mellon University.
[78] Las observaciones que acá se mencionan son importantes para justificar el por qué primero se trabajará con Alice y luego con BlueJ en esta propuesta de innovación pedagógica.
[79] El uso de guiones textuales y gráficos se discutirá con más detalle en la primera unidad didáctica de la asignatura, que se encuentra en la tercera parte de esta propuesta.

propios métodos, como por ejemplo volar, nadar, bailar, desplazarse hacia la izquierda, derecha, arriba, abajo, etc.

3. La habilidad para depurar sus aplicaciones, en el sentido que Alice les provee los medios para que ellos puedan ejecutar sus programas y los métodos personalizados que hayan debido construir para llevar a cabo alguna característica propia de la animación que se desee recrear.
4. Una visión incremental para programar, tanto para los métodos de mundo como de clase[80], que son los que más trabajo demandan, junto con la definición de funciones adicionales que fueran necesarias en sus proyectos. De acuerdo a sus observaciones, los estudiantes programan de forma incremental, porque no escriben todo de una vez, sino que han adquirido la costumbre de probar cada bloque de software a medida que las van creando.
5. Una mirada mucho más concreta a los conceptos de objeto, lo que viene apoyado de manera potente por el propio entorno de trabajo.
6. Un sentido más intuitivo de lo que significa el concepto de encapsulamiento, sobre todo cuando tienen que modificar el estado de una escena cuando algunos de los métodos de objetos deban ser invocados. Por ejemplo, la posición de un objeto se puede modificar invocando al método *mover*, sin que sea necesario que ellos como programadores deban conocer con exactitud cuáles son las coordenadas espaciales del objeto en cuestión.
7. La noción de método, entendiéndolo como la acción que debiera ejecutar un método cuando se le requiera, donde la manera de lograr que el objeto haga la tarea que se le pide es por medio del paso de un mensaje hacia el mismo.
8. Un sentido fuerte de lo que es herencia, en la manera en cómo los estudiantes pueden escribir código adicional para ampliar el abanico de prestaciones que les brinde alguna clase.

[80] Estos conceptos se abordarán en la primera unidad didáctica de la asignatura.

9. La habilidad de colaborar en trabajos grupales, porque los alumnos trabajan en la construcción de personajes individuales, que luego pueden combinar con las creaciones de sus compañeros para terminar construyendo
10. Un mejor entendimiento de lo que son los datos *booleanos*[81], en cuanto a que el propio editor de código tiene la «*inteligencia*» para evitar que el estudiante arrastre expresiones o tipos de datos que no correspondan con las reglas de construcción de alguna sentencia condicional o de un bucle.[82]
11. La intuición para crear eventos, y para determinar cuándo resulta apropiado emplear este tipo de construcción de software.[83]

Esto les permite a los alumnos desarrollar programas desde cero, o construir aplicaciones con mundos y clases virtuales que hayan sido fabricadas con anterioridad. En este último caso se deja abierta la posibilidad para que ellos puedan experimentar con la modificación de elementos ya existentes, y así poder generar productos más robustos en términos de las nuevas funcionalidades que puedan añadir.

Con respecto a las debilidades observadas, se puede decir que Alice no estimula que el estudiante repare en la importancia que tienen las normas sintácticas de programación, a pesar que ellos puedan configurar el entorno para que el código fuente tenga una apariencia similar al de una aplicación Java. El problema radica en que esta herramienta ha sido construida bajo la filosofía de diseño de «*arrastrar y soltar*» — o «*drag and drop*» en su versión técnica más pura —, de tal manera que el alumno no experimenta errores comunes en programación como dejar una llave sin cerrar u olvidar poner un punto y coma al final de una sentencia. Lo que ellos buscan es que primero los estudiantes puedan aprender las nociones iniciales de programación, y que luego hagan una

[81] Son tipos de datos que pueden tomar valores dentro de la lógica binaria: verdadero o falso.
[82] Las ideas de sentencias condicionales y bucles serán abordadas en la tercera unidad del curso.
[83] En la segunda unidad del curso se aborda el tema de eventos.

migración[84] hacia Java que a la larga les permita dominar la sintaxis particular de esta herramienta.

3.3.3 Software educativo BlueJ

BlueJ es un entorno de desarrollo integrado[85] que ha sido desarrollado — al igual que Alice — con propósitos educacionales para apoyar la enseñanza de Java, que está siendo desarrollado y mantenido por la University of Southern de Dinamarca, la *Deakin University* en Australia, y la *University of Kent* en Canterbury, Reino Unido.

Dada su orientación pedagógica, este recurso didáctico presenta algunas características que lo diferencian de otros entornos de desarrollo, como NetBeans o Eclipse:[86]

- Interfaz de usuario simple de utilizar. Los desarrolladores aseguran que un estudiante que no tiene mayor experiencia con este paradigma de programación pude llegar a tener un uso competente de esta herramienta en 20 minutos. Lo positivo de esto es que la enseñanza se puede concentrar en los temas que son realmente importantes, como por ejemplo orientación a objetos, encapsulamiento, herencia, polimorfismo, Java, en desmedro de otros asuntos técnicos como rutas de clase, variables de entorno del sistema operativo, posibles conflictos con archivos DLL, etc.
- El entorno de trabajo cuenta con algunas herramientas de enseñanza que no están disponibles en otros. Una de ellas corresponde a la visualización que el estudiante puede tener de las clases que componen el proyecto en el

[84] La transición hacia lenguajes de programación orientados a objetos tradicionales como Java se aborda en la quinta unidad de la asignatura.
[85] Un entorno de desarrollo integrado — o IDE por sus siglas en inglés — es un ambiente de programación que provee una serie de herramientas que vienen empaquetadas dentro de la misma aplicación, para que no sea necesario recurrir a otras de carácter externo, que incluye editor de código, compilador, depurador — o *debugger* — y constructor de interfaz gráfica.
[86] En los ámbitos profesionales, tanto NetBeans como Eclipe son las herramientas más utilizadas por los programadores, pero que no fueron concebidas inicialmente como recursos didácticos para apoyar el desarrollo de la enseñanza de la programación orientada a objetos.

cual está trabajando. La utilidad de esto es poder representar en un diagrama UML[87] las clases y sus relaciones con otras, de tal manera que el aprendizaje se puede hacer de una manera gráfica[88], a diferencia de lo que ocurre en NetBeans donde la interacción que tiene el usuario con el programa se da de manera única a través de las líneas de código.

- Una de las grandes fortalezas que tiene BlueJ es la facilidad de acceso que tiene el usuario a los objetos de cualquier clase, y de poder interactuar y experimentar con los métodos de manera directa.[89]

Por otra parte, las funciones de interacción visual de BlueJ están diseñadas para simplificar la enseñanza de algunos conceptos o sentencias clásicas de programación que de forma tradicional resultan de gran dificultad para los educadores, como por ejemplo: (1) la función principal de una aplicación escrita en Java empieza con la sentencia

```
public static void main(String[] args)
```

que difícilmente un estudiante pueda comprender la primera vez que se vea enfrentado a escribir un programa en este lenguaje. Existen palabras claves como `public` y `static` que no se pueden explicar de manera satisfactoria en un principio, ya que es necesario conocer otros conceptos previos, como el de clase, objeto, métodos, argumentos, arreglos, tipos de datos. En este sentido, BlueJ permite que la creación de determinados bloques de programas se pueda hacer de manera interactiva, sin que el usuario tenga que preocuparse — por el momento — de los comandos utilizados.

[87] UML corresponde a la sigla que en español significa «*lenguaje unificado de modelamiento*», que es una de las herramientas para modelar sistemas de software más utilizadas en la actualidad.
[88] El docente o el estudiante puede desarrollar por completo un sistema de software profesional empleando por completo las características de expresividad que provee este lenguaje.
[89] Algunas de estas características esenciales del entorno BlueJ fueron obtenidas desde Barnes, D., & Kölling, M. (2007). *Programación Orientada a Objetos: Una introducción práctica usando BlueJ*. Editorial Pearson Educación.

Algunos casos de uso

Dentro de las múltiples experiencias de uso de BlueJ como recurso didáctico, se puede mencionar las de Svetlana Kouznetsova[90], o la de Kelsey Van Haaster junto con Dianne Hagan[91], por ser las más actualizadas del repositorio de artículos técnicos que se pueden consultar directamente desde bluej.org.

En el primer caso, la autora relata cómo ha sido el trabajar con BlueJ para que los estudiantes desarrollaran el juego de mesa llamado «*black jack*», como una forma de lograr que ellos reforzaran los conceptos fundamentales de la programación orientada a objetos de una manera lógica y consistente. Reconoce que el entorno de trabajo de BlueJ es ameno, incluso para novatos como los estudiantes que tuvo a su cargo en un curso introductorio a las Ciencias de la Computación, quienes lograron mejores niveles de aprendizaje gracias a la utilización de esta herramienta, lo que explica a través de dos factores: (1) la facilidad de uso del editor de código; y (2) por los códigos de ejemplo que le permiten a los estudiantes crear imágenes sin que sea necesario tener mínimos conocimientos de alguna clase que implemente el motor gráfico de Java, como el caso de *Swing*.

En el segundo caso, Van Haaster y Haggan coinciden con Kouznetsova, en el sentido que esta herramienta permite lograr un mejor entendimiento de la teoría que subyace al paradigma de la programación orientada a objetos. Una diferencia entre ambos trabajos, es que los primeros aplicaron un estudio en el cual se les enseñó en una unidad didáctica a escribir y ejecutar código que sólo empleara el Java SDK[92], y en otra donde usaran BlueJ, donde el beneficio de los estudiantes

[90] Kouznetsova, S. (2007). *Using BlueJ and blackjack to teach object-oriented design concepts in CS1*. Huntsville, Texas: Sam Houston State University.
[91] Van Haaster, K., & Hagan, D. (2004). *Teaching and Learning with BlueJ: an Evaluation of a Pedagogical Tool*. Melbourne: Monash University.
[92] SDK, por sus siglas en inglés, significa «*kit de desarrollo de software*», que — como su nombre lo indica — corresponde a un conjunto de herramientas que le permite al programador desarrollar aplicaciones computacionales para algún sistema particular. Por ejemplo, está el *SDK de Java* construido por Sun Microsystems, y que ahora está en poder de Oracle; o el *SDK de DirectX de Microsoft*, en el que se codifican la mayoría de los videojuegos para el sistema operativo Windows.

fue notable, de acuerdo a los resultados que se pueden apreciar en el artículo consultado.

Reflexión final del capítulo

Tanto el aprendizaje basado en problemas como el software educativo pueden convertirse en un excelente aliado del profesor en su tarea de enseñarle de manera significativa al estudiante. Sin embargo, existen algunos inconvenientes derivados de sus usos, como los mostrados en las secciones 3.1.7 y 3.3, a los que no se puede estar ajeno.

Un denominador común que ayudaría a paliar en cierta medida estas dificultades inherentes de estas herramientas, es que el profesor sea preciso en cuanto a las instrucciones de uso que indique en algún apunte o en una guía, como una forma que el alumno se enfoque en la actividad principal que tenga que desempeñar, y que él mantenga en todo momento un control sobre el curso de las acciones que se desarrollen al interior del aula, como una forma de evitar — entre otros aspectos — que la atención del estudiantado se desvíe hacia las atracciones multimedia que ofrezca el entorno. Es por esto que el material de estudio que se ha elaborado va en la dirección de mostrarle al alumno cuáles son las funcionalidades que ofrece Alice para desarrollar algunas acciones específicas a partir de las clases que sea necesario instanciar en un momento determinado, a partir de un amplio surtido de imágenes que ayudan a guiar al estudiante a desarrollarlas de manera eficiente.

Además, la estrategia metodológica del aprendizaje basado en problemas estimula el trabajo colaborativo entre los estudiantes, y que el profesor esté alerta para monitorear en todo momento el estado de avance parcial de los proyectos en los cuales ellos se hayan embarcado. La importancia de esto radica en que uno de los graves problemas que Pere Marquès señala con respecto a la utilización del software educativo, es que tiende al empobrecimiento de las relaciones humanas, o sensación de aislamiento: estas características adversas se pueden sopesar

gracias al trabajo en equipo y también a la asistencia que el profesor deba brindar en cualquier momento para evitar efectos colaterales como ansiedad, cansancio o monotonía, por medio de alguna estrategias de aprendizaje que se describen en la sección 3.1, como los guiones de aprendizaje, investigación en grupo o co-op co-op. En este sentido, el profesor tiene la responsabilidad de detectar a tiempo estos problemas, y de aplicar alguna medida pedagógica para mitigar o paliar los efectos de estos problemas.

El diseño de las experiencias ABP[93] apunta a validar que todos los miembros de un equipo haya trabajado, a sabiendas que esa es una realidad que no se reparte por igual entre los integrantes del grupo, pero que al menos estimula la participación de todos, ya que su desempeño al momento de la evaluación va a influir en el rendimiento de sus compañeros. Nuevamente, el profesor ha de desempeñar un rol fiscalizador mucho más riguroso para minimizar estos problemas.

En este capítulo también se ha hablado de Alice y de BlueJ como los principales recursos didácticos que se va a utilizar como medios de apoyo para la asignatura propuesta. A partir de lo que se ha expuesto en párrafos anteriores, Alice es un entorno de desarrollo que permite a los estudiantes aprender de manera básica elementos básicos de las Ciencias de la Computación, y que al mismo tiempo le permite crear películas animadas o videojuegos sencillos, a través del control de objetos y personajes 3D en un mundo virtual, sin que sea necesario que ellos tengan experiencia en programación. Es por este motivo que Alice es lo primero que se enseña. Aunque BlueJ también podría ir en un principio, lo que se quiere es que los alumnos puedan vivenciar la programación a partir de elementos tangibles, y que luego vayan aumentando el nivel de abstracción hasta llegar al uso de elementos abstractos como los que se muestran en BlueJ, que no tiene el carácter lúdico — y por lo tanto motivador — de Alice.

Por otra parte, Alice está pensado para abstraer al alumno de la complejidad de las normas sintácticas del lenguaje, como una manera de hacer

[93] Ver *Anexo B. Experiencias de Aprendizaje Basado en Problemas*.

que éste se concentre fundamentalmente en la construcción del programa, por medio de sistema basado en «*arrastrar y soltar*» que le permite tener acceso a todas las construcciones necesarias para el aprendizaje de la programación. Lamentablemente, a su vez esto se ha convertido en una debilidad, porque en los lenguajes «*reales*» la sintaxis puede llegar a ser un problema que impida la ejecución de algún proyecto. En este sentido, emerge BlueJ como la herramienta para introducir la programación en Java, que es el lenguaje oficial que se revisa en la Universidad San Sebastián. En cambio, Alice se va a emplear para que el estudiante tenga un primer acercamiento a la teoría de una manera innovadora y no tan tradicional como la del otro software.

Al igual que la estrategia metodológica de enseñanza del aprendizaje basado en problemas, ambos programas están basados en la teoría constructivista, en la cual al alumno se le entregan las herramientas para que pueda crear sus propios procedimientos de análisis, a objeto de poder brindar una solución creativa a un problema abierto, por medio de experiencias concretas que se dan a partir del uso de estas herramientas didácticas que proveen vivencias concretas para entornos abstractos de desarrollo computacional, tales como relaciones entre clases con objetos, instanciación, llamadas a métodos, pasaje de parámetros, definición de funciones, aplicación de estructuras de programación como bloques condicionales, bucles, que se deben combinar con estructuras de datos ad hoc como listas o arreglos, por citar a las más comunes. Lo que estos recursos buscan en conjunto es que los estudiantes nóveles desarrollen de manera más sencilla modelos mentales que les permitan articular con creatividad soluciones de software mediante la lógica de la programación orientada a objetos, que — como se vio en la sección 3.2 — tiene pocos elementos con las cuales se puedan levantar aplicaciones: variables, constantes, estructuras de control, estructuras iterativas, y características propias del paradigma.

Capítulo 4

Planificación de la Asignatura

Introducción

Este capítulo se inicia con el programa de la asignatura «*Tecnologías de la Información*», que está sujeta al cumplimiento de los principios de secuencia vertical, continuidad, integración, congruencia horizontal y comunicabilidad. Es importante destacar que este primer documento debe estar contextualizado de acuerdo a la realidad de la Universidad San Sebastián, y en este sentido se van a incluir elementos como:

- Descripción general del curso, por medio de una identificación el mismo, en función de la cantidad de horas teóricas y prácticas, el número de créditos que ha asignado la institución, porcentaje mínimo de asistencia, etc.
- Una descripción que dé cuenta — entre otros aspectos — del aporte que haga al perfil de egreso del estudiante.
- Objetivos generales y específicos que se pretenden alcanzar con ella.
- Desglose de cada unidad didáctica, que incluya los contenidos y los aprendizajes esperados en ella en términos cognitivos, procedimentales y actitudinales.
- Metodología de trabajo.
- Evaluaciones que se vayan a realizar.

Posterior a eso, se incluye la planificación didáctica del curso, que tiene como objetivo dar a conocer las actividades curriculares que se darán clase a clase, a través de la explicitación de la estrategia metodológica a emplear, los recursos bibliográficos que sean necesarios, recursos didácticos que sean necesarios y el tipo de evaluación que se vaya a practicar.

Planificación curricular

El currículum constituye el eje desde el cual se construye y articula toda la acción pedagógica, constituyéndose así en un proceso de selección de cultura que

determina qué es lo que los estudiantes deben aprender para lograr una exitosa inserción en las estructuras económica, social y cultural de una sociedad.

El diseño del currículum conlleva la tarea de planificar, entendida como aquella actividad de corte reflexivo que realiza el profesor cuyo producto final permita dilucidar interrogantes relacionadas con: determinar las capacidades y limitaciones de los alumnos, sus intereses, necesidades; cuáles pueden ser los documentos de apoyo que van a servir para que la transmisión de conocimiento se haga efectiva de la mejor manera posible, qué contenidos disciplinarios se deberán dejar fuera y cuáles potenciar; cómo se abordarán los temas durante el desarrollo de la clase, bajo qué estrategia uno como docente podrá acercarse mejor a los alumnos; cómo se harán las evaluaciones, entre otros. Pero también es necesario validar que este instrumento satisfaga los cinco principios de la planificación curricular.[94] La labor de planificar conlleva una serie de ventajas en pro de una mejor coherencia y eficiencia de la enseñanza.

4.1 Identificación de la asignatura

NOMBRE DE LA ASIGNATURA	Tecnologías de la Información
FACULTAD	Ingeniería y Tecnología
NÚMERO DE CRÉDITOS	5
CARÁCTER	Profesional
EQUIPO DOCENTE	Milton A. Ramírez Klapp
RÉGIMEN	Semestral
PRE-REQUISITOS	No tiene.
NIVEL EN LA MALLA CURRICULAR	Primero
HORAS SEMANALES DE TEORÍA	6

[94] Los cinco principios son: secuencialidad vertical, continuidad, integración, congruencia horizontal y comunicabilidad, que fueron descritos en el primer capítulo de este informe.

HORAS SEMANALES DE AYUDANTÍA	2
HORAS SEMANALES DE LABORATORIO	6
NÚMERO DE SEMANAS DE CLASE	18
CARGA ACADÉMICA SEMANAL (HORAS)	5.3 horas directas, y 3 horas indirectas
TOTAL DE HORAS DE CLASE AL SEMESTRE	144
REQUISITO MÍNIMO DE ASISTENCIA	75%

4.2 Descripción

La asignatura «*Tecnologías de la Información*» forma parte del grupo de cursos que conforman los cinco semestres del Plan Común de Ingeniería Civil de la Universidad San Sebastián, conducente al Bachillerato en Ciencias de la Ingeniería, y que complementa y da sustento a las diversas disciplinas de la ingeniería, lo que permite alcanzar las competencias definidas en cada especialidad.

En cuanto al aporte específico que este curso pueda hacer al perfil de egreso de la carrera, se puede decir que esta asignatura busca desarrollar en el alumno el pensamiento algorítmico, lógico y analítico mediante la recreación computacional de situaciones propias de su ámbito de especialidad. En primera instancia, el estudiante adquirirá estas habilidades por medio del entorno de programación Alice, cuyo uso se basa en la construcción de mundos virtuales donde los alumnos podrán diseñar animaciones para solucionar determinados escenarios problemáticos, y luego por Blue, que será la herramienta empleada para introducir al alumno en entornos de programación reales, como Java.

4.3 Objetivos Generales

1. Reconocer las distintas etapas por las que ha transitado la historia de los computadores.
2. Reconocer los elementos que componen un algoritmo.
3. Identificar cuáles son las características esenciales de la programación orientada a objetos.
4. Diseñar un esbozo de solución algorítmica para un escenario determinado empleando pseudolenguaje o un guión de trabajo.
5. Construir animaciones estáticas o interactivas en Alice que permitan desarrollar los diseños planteados por los estudiantes, y que luego se puedan transportar a Java mediante BlueJ.
6. Valorar la importancia de usar un computador como herramienta de trabajo para solucionar problemas que se presenten en el campo de especialidad del alumno.
7. Adoptar una actitud responsable y comprometida frente al trabajo en equipo.

4.4 Objetivos Específicos

1. Diferenciar los conceptos de hardware y software, y cómo éstos influyen en el quehacer computacional.
2. Establecer una categorización del hardware y software que se emplea en un sistema computacional.
3. Reconocer las variables, sentencias, flujos de control y llamadas a subrutinas que conforman un algoritmo particular.
4. Comprender la utilidad de construir un diagrama de flujo de datos, o un guión de trabajo.

5. Construir un diagrama de flujo de datos, en función de las variables, estructuras de control y llamadas a subrutinas que defina el estudiante.
6. Instalar y ejecutar el software Alice en el computador.
7. Reconocer cuándo es necesario diseñar una animación estática o una de carácter interactivo.
8. Exportar los trabajos realizados a formato de video, para luego ser compartidos en sitios como YouTube o Google Video.
9. Apreciar la evolución histórica que ha ido experimentando la computación con el devenir del tiempo.
10. Compartir conocimientos con los demás compañeros de grupo en pro del éxito académico colectivo.
11. Mostrar disposición para realizar trabajos en equipo, y a colaborar en todo lo que sea necesario en pro del éxito del grupo.

4.5 Unidades didácticas

En la Tabla 3 se presentan las unidades didácticas que configuran la propuesta de asignatura:

Tabla 3 Unidades didácticas que componen la asignatura propuesta.

No.	Unidad	Horas
I.	Introducción a la Programación.	14
II.	Presentación de la Programación Orientada a Objetos.	16
III.	Estructuras de Programación.	18
IV.	Agrupación de elementos.	16
V:	Transición a Java.	44
	Total de horas	108

4.5.1 Unidad I. Introducción a la Programación

Aprendizajes Esperados

COGNITIVOS:

1. Reconocen los conceptos fundamentales relacionados con la Algoritmia.
2. Identifican los elementos que conforman el entorno de trabajo de Alice.
3. Conocen el modelo de diseño «*top-down*» para resolver problemas, y que esto se puede transportar a la descomposición de una historia en varios fragmentos narrativos para ser implementados en Alice.
4. Reconocen que un escenario en Alice es equivalente con la formulación de un problema.
5. Conocen en términos generales los conceptos de bucle, cómo escribir una expresión y cómo llamar a funciones.[95]

PROCEDIMENTALES:

1. Identifican los conceptos fundamentales de la Algoritmia.
2. Documentan un programa que se haya codificado en Alice.
3. Utilizan los atributos, métodos y funciones de un objeto para implementar una solución en Alice.
4. Traducen el enunciado formal de un problema a una historieta.
5. Codifican la historieta en un programa computacional hecho en Alice.
6. Realizan operaciones aritméticas y lógicas sobre los atributos de los objetos por medio de expresiones.
7. Realizan pruebas a los programas que desarrollen a través de simulaciones.

[95] Es general, porque de eso se hablará en la Unidad III. Funciones y Estructuras de Control.

ACTITUDINALES:

1. Valoran el propósito que conlleva la documentación de programas.
2. Internalizan el aporte que la computación puede dar a problemas de su campo de especialidad por medio de la automatización de tareas.

Contenidos

- Presentación de la Algoritmia, o ciencia de los algoritmos.
- Lenguajes de programación.
- Introducción a Alice.
- Escenarios de trabajo, historietas, animaciones.

4.5.2 Unidad II. Presentación de la Programación Orientada a Objetos

Aprendizajes Esperados

COGNITIVOS:

1. Reconocen la diferencia entre una clase y un objeto.
2. Conocen el concepto de método.
3. Diferencian los distintos tipos de parámetros que se pueden pasar a un método.
4. Conocen cómo invocar a un método en términos de los parámetros que se requieran.
5. Reconocen la diferencia entre una acción y un evento.

PROCEDIMENTALES:

1. Crean instancias de clases predefinidas.
2. Crean y aplican métodos que se dan a nivel de mundo y de clases.
3. Crean y eligen cuál es el tipo de parámetro más apropiado para pasárselo a un método.

4. Diseñan un programa interactivo a partir de un libreto dado.
5. Aplican el desarrollo incremental de aplicaciones para chequear el correcto funcionamiento de sus aplicaciones.
6. Crean y aplican parámetros específicos para activar algún método de una clase con algún evento.

ACTITUDINALES:

1. Valoran el hecho de trabajar con piezas de software reutilizable, como son las clases.
2. Proponen diferentes alternativas para solucionar el problema de construir un programa interactivo a partir de una especificación dada.
3. Aceptan las diferencias individuales que puedan tener con sus compañeros o con el equipo docente.
4. Reconocen la importancia del manejo de herencia como una manera de reutilizar las características de la clase madre, y de poder hacer modificaciones particulares de su comportamiento, sin necesidad de definirla de nuevo.

Contenidos

- Métodos de nivel de mundo.
- Métodos de nivel de clase.
- Parámetros.
- Objetos visibles e invisibles.
- Programación dirigida por eventos.

4.5.3 Unidad III. Estructuras de Programación

Aprendizajes Esperados

COGNITIVOS:

1. Conocen la manera de utilizar una función para encontrar información acerca de un objeto del mundo.
2. Conocen cómo utilizar la información provista por una función para poder tomar una decisión en cuanto al flujo de ejecución de un programa.
3. Reconocen cuándo es necesario anidar una estructura «*si-sino*» dentro de bloques de programa.
4. Conocen cuáles son los operadores booleanos que se pueden emplear en bloques de decisión.
5. Identifican las diferencias que hay entre un bucle «*para*» de un «*mientras*».
6. Conocen cuándo es necesario aplicar uno u otro tipo de bucle.

PROCEDIMENTALES:

1. Construir funciones de clase que sean apropiadas para determinados pasajes de un libreto o historieta, junto a los bucles que sean necesarios.
2. Aplican operadores booleanos para construir sentencias lógicas relacionadas con el control de ejecución de un programa.
3. Generan números aleatorios para recrear movimientos al azar dentro de un programa.
4. Construyen aplicaciones que requieran del anidamiento de estructuras de control y de bucles dentro de ellas.

ACTITUDINALES:

1. Valoran la existencia de bloques de control de un programa que permitan diferentes cursos de ejecución de los mismos.

2. Asumen una actitud de compromiso con sus aprendizajes, en función de la creciente dificultad que está tomando la asignatura, sobre todo en lo concerniente al anidamiento de estructuras de control y bucles.
3. Valoran la inclusión de números aleatorios como una alternativa válida para otorgarle originalidad a la creación de un programa.

Contenidos

- Funciones.
- Estructuras de control: «*si-sino*».
- Expresiones booleanas.
- Números aleatorios.
- Bucles: «*para*» y «*mientras*».

4.5.4 Unidad IV. Agrupación de Elementos

Aprendizajes Esperados

COGNITIVOS:

1. Conocen el concepto de arreglo, y sus múltiples usos en computación.
2. Reconocen cómo se puede acceder a determinados ítems de un arreglo en base a su posición dentro del mismo.
3. Conocen el concepto de herencia.

PROCEDIMENTALES:

1. Diseñan e implementan programas en Alice que hagan uso de listas y arreglos como estructuras de datos válidas para organizar elementos dentro de una animación.
2. Acceden a elementos particulares de un arreglo por medio de bucles.

ACTITUDINALES:

1. Valoran la forma en que se disponen los elementos de una lista y de un arreglo, con la finalidad de poder accesarlos de manera expedita durante la ejecución de un programa.

Contenidos

- Listas.
- Arreglos.

4.5.5 Unidad V. Transición a Java

Aprendizajes Esperados

COGNITIVOS:

1. Reconocen las similitudes que se pueden dar entre la programación en Alice con la programación en Java.
2. Conocen que un programa en Java se compone de una o más clases.
3. Reconocen que cada objeto posee métodos con los cuales puede comunicarse con otro objeto para desempeñar alguna acción determinada.
4. Conocen las partes de un código fuente escrito en Java.
5. Conocen la diferencia entre tipos de datos primitivos y aquellos que son propios de objetos.

PROCEDIMENTALES:

1. Instalar BlueJ en su computador de trabajo.
2. Identifican clases y objetos por medio del entorno BlueJ.
3. Emplear BlueJ para correr algún programa.
4. Manipular el constructor de una clase para instanciarlas.
5. Traducir un libreto a un programa escrito en Java.

6. Aplican la técnica de diseño «*top-down*» para manejar problemas de programación.
7. Conocen la manera de declarar tipos de datos y llamar a funciones y métodos dentro de un programa escrito en Java.

ACTITUDINALES:

1. Valoran el hecho de emplear Alice como una herramienta para enseñar los primeros conceptos de programación bajo un entorno de trabajo donde los únicos errores que se puedan cometer sean de tipo lógicos y no sintácticos.
2. Aprecian la simplicidad de trabajar con Alice, como una forma de transitar hacia un lenguaje de programación como Java.

Contenidos

- Presentación de Java.
- Introducción del entorno de trabajo BlueJ.
- Variables y tipos de datos usados en Java.
- Operadores y funciones en Java.

4.6 Metodología

Esta asignatura se desarrollará en base a la estrategia metodológica de enseñanza inspirada en el aprendizaje basado en problemas, como una forma de lograr que el foco de atención de las clases se centre en el alumno, y no en el docente como ocurre en el enfoque academicista tradicional. Como corolario de lo anterior, se dará relevancia a la discusión y el análisis de cada uno de los temas tratados en el espacio de aula, como una forma de lograr que los estudiantes desarrollen a partir del primer año de carrera la capacidad de razonamiento lógico-deductivo de una forma estructurada que se traduzca en productos de software que conjuguen la creatividad y la rigurosidad propia de un ingeniero, de tal manera

que la búsqueda de la solución a un problema le permita al estudiante movilizar un repertorio de conductas cognitivas, procedimentales y actitudinales.

Las clases se desarrollarán en ambiente de laboratorio de computación, donde el proyector, pizarrón, Alice y BlueJ constituirán los recursos didácticos principales que permitirán articular cada una de las sesiones previstas. Además, Moodle se utilizará para ordenar y presentar los contenidos disciplinarios, y YouTube para la publicación de los trabajos desarrollados por los estudiantes.

Además, las clases de ayudantía servirán para que los estudiantes aprovechen de reforzar los contenidos teóricos juntos a los ejemplos y ejercicios que vienen propuestos en el apunte que se va a subir de manera progresiva a la plataforma Moodle de la asignatura.

4.7 Evaluación

Este curso contempla tres Evaluaciones, determinadas según la siguiente tabla:

Evento Evaluativo	Ponderación
Prueba Solemne 1	20%
Prueba Solemne 2	20%
Prueba Solemne 3	40%
Test, taller o tareas	20%

Donde:

Prueba Solemne. Es aquella evaluación de carácter global que permite medir los objetivos logrados del programa de enseñanza aprendizaje en que participa el alumno, durante el periodo total anterior a la fecha en que se aplica dicha evaluación. Cada prueba solemne deberá aplicarse en los períodos establecidos según Calendario Académico.

Test. Corresponden a evaluaciones individuales realizadas al inicio de la clase de corta duración.

Taller. Corresponden a evaluaciones realizadas en forma grupal por los alumnos y dirigidas por el profesor.

Tarea: corresponden a evaluaciones realizadas por el alumno en forma individual fuera del horario de clase.

Además, de acuerdo a la normativa vigente de la institución, debe haber por lo menos tres evaluaciones parciales correspondientes a test, taller o tareas. Existe la instancia de un Certamen Recuperativo, evaluación global que se realiza con posterioridad al tercer Certamen, con el objeto de reemplazar la nota de algún Certamen anterior (por no haberse presentado o para mejorar dicha nota). Las notas obtenidas en evaluaciones parciales no serán reemplazables (test, controles, interrogaciones orales, presentaciones de trabajos, etc). Se deja constancia que la asignatura se aprueba con nota final igual o mayor que 4,0.

4.8 Bibliografía

BÁSICA:

1. Kölling, M. (2005). *El «tutorial» de BlueJ.* University of Southern Denmark.[96] [97]

2. Dann, W., Cooper, S., & Pausch, R. (2012). *Learning to Program with Alice.* Pittsburgh: Editorial Pearson.

[96] El libro oficial de aprendizaje de Java con BlueJ se llama «*Objects First Java*», editado por Pearson, y escrito por Michael Kölling. Los dos primeros capítulos están disponibles para el público en la URL http://www.bluej.org/objects-first/evaluation.html. Como el libro está en inglés, el apunte que corresponda en este caso será una traducción del primer capítulo, que estará a cargo del profesor.

[97] Es importante mencionar que en la página oficial de BlueJ están disponibles los ejemplos de proyectos que se pueden emplear para acompañar el desarrollo de las actividades. El recurso está disponible de manera gratuita desde la URL http://www.bluej.org/objects-first/resources/projects.zip.

COMPLEMENTARIA:

1. Deitel H., & Deitel, P. (2005). *Cómo programar en Java.* Editorial Prentice Hall.

2. Wu, T. (2008). *Programación con Java.* Editorial McGraw – Hill. Madrid.

Planificación didáctica

La planificación didáctica apunta a elaborar una secuencia que brinde una respuesta a qué se enseñará y cómo se enseñará a partir de los conocimientos que poseen los estudiantes para que ellos puedan lograr los objetivos de aprendizajes propuestos, y también para que este documento permita mostrar una organización lógica de los contenidos que se vayan a tratar, de tal manera que éstos sean enseñados de la manera más eficaz posible.

El orden y la temporalización de las actividades de aprendizaje representan la estructura sistemática para controlar las acciones pedagógicas durante el proceso educativo y lograr los propósitos educativos. Habrán de considerarse aspectos como: las características de los estudiantes, los contenidos de aprendizaje, los conocimientos previos de la asignatura, los recursos y medios didácticos, los objetivos educativos que se pretenden lograr, la metodología de trabajo, los tiempos disponibles para desarrollar las actividades, las características, métodos y criterios de evaluación entre otros, aunque este último queda fuera de nuestro estudio por las razones antes expuestas.

Para los educadores, la planeación de una asignatura permite la distribución y la organización de los contenidos de acuerdo con el tiempo disponible, así como la selección de la bibliografía correspondiente y su relación con los temas de estudio, y además se puede ver como una base sobre la cual pueden tomarse acuerdos de trabajo para apoyar el desempeño de los estudiantes, estableciendo vínculos entre las distintas asignaturas. Del punto de vista de los estudiantes, la planificación proporciona un conocimiento sobre los

propósitos, temas y actividades de la asignatura, de tal manera que se sientan en mejores condiciones para aprovechar al máximo el proceso de aprendizaje al saber cómo se ha estructurado en período académico que estén cursando

Dentro de la tarea docente, se puede decir que tanto la planeación curricular como la didáctica se constituyen en piezas centrales para llevar a cabo la propuesta de enseñanza del profesor y responder sobre cómo implementar dicha propuesta. Por otro lado, permite conjugar la teoría con la práctica; es decir, poder hacer uso de los contenidos de la forma más conveniente posible; así, una planificación apropiada implica que el docente pueda recurrir a diferentes herramientas y metodologías para que los contenidos lleguen de mejor manera a los alumnos y así lograr que ellos puedan adquirir aprendizajes que realmente sean significativos. Pensar con anterioridad la asignatura permite secuenciar y segmentar el contenido, haciéndolo coherente y funcional, lo que repercute directamente en la capacidad de los alumnos para apropiarse y asimilar los mismos de manera global e íntegra. Sin embargo, no hay que olvidar que la enseñanza es un proceso dinámico, en que influyen muchas variantes que a veces escapan al control y planificación. Por esto, no siempre hay que ver la planificación como una instancia rígida sin posibilidad de cambio, sino que debiera ser vista como una importante guía de apoyo, que a veces puede modificarse en virtud de circunstancias especiales que no se pueden prever desde principios de semestre.

La Tabla 4 da a conocer la planificación didáctica que se ha elaborado para esta asignatura, tomando en consideración que ésta se ha debido contextualizar al ámbito de la Universidad San Sebastián, en la forma del documento conocido en esta institución como syllabus.

UNIVERSIDAD SAN SEBASTIAN

SYLLABUS ESCUELA INGENIERÍA / CARRERA DE INGENIERÍA CIVIL INFORMÁTICA

Tabla 4. Planificación didáctica para la asignatura propuesta

NOMBRE DEL CURSO	Tecnologías de Información	**NOMBRE DEL DOCENTE**	Milton Ramírez Klapp	**CÓDIGO**	INGE1010

OBJETIVO DEL CURSO:

Al finalizar la asignatura, el alumno deberá ser capaz de:

1. Reconocer las distintas etapas por las que ha transitado la historia de los computadores.
2. Reconocer los elementos que componen un algoritmo.
3. Identificar cuáles son las características esenciales de la programación orientada a objetos.
4. Diseñar un esbozo de solución algorítmica para un escenario determinado empleando pseudolenguaje o un guión de trabajo.
5. Construir animaciones estáticas o interactivas en Alice que permitan desarrollar los diseños planteados por los estudiantes, y que luego se puedan transportar a Java mediante BlueJ.
6. Valorar la importancia de usar un computador como herramienta de trabajo para solucionar problemas que se presenten en el campo de especialidad del alumno.
7. Adoptar una actitud responsable y comprometida frente al trabajo en equipo.

NORMAS Y PROCEDIMIENTOS GENERALES (si corresponde):

El reglamento de docencia de la Universidad establece que el mínimo porcentaje de asistencia que tenga un estudiante debe ser igual al 75% del total de sesiones que componen la asignatura.

PARAMETRIZACIÓN DE EVALUACIONES	**Pruebas Solemnes No. 1 y 2: 20%**	**Prueba Solemne No. 3: 40%**	**Promedio de tareas y controles: 20%**

MÓDULO I. Introducción a la Programación Computacional

No. Sesión	Contenido	Estrategia metodológica	Lectura recomendada	Evaluación	Fecha de realización
1	Historia de los computadores.	Lectura comprensiva de cada generación de computadores, donde cada grupo elabora una línea de tiempo histórica según el periodo seleccionado, en base a la estrategia co-op co-op.	Héctor Sanjuán: «*Historia de los computadores*».	Formativa.	04/03/2013.
2	Historia de los computadores.	Lectura comprensiva de cada generación de computadores, donde cada grupo elabora una línea de tiempo histórica según el periodo seleccionado, en base a la estrategia Co-op Co-op.	Héctor Sanjuán: «*Historia de los computadores*».	Formativa.	06/03/2013.
3	Historia de los computadores.	Sesión expositiva a cargo de los distintos grupos de estudiantes.	Héctor Sanjuán: «*Historia de los computadores*».	Sumativa (tributa al 20% de la nota final en cuanto a tareas y controles).	08/03/2013.
4	Introducción a Alice.	Clase expositivo-demostrativa del profesor retroalimentada con lectura de los estudiantes.	Apéndice del texto de apoyo, preparado por el profesor.	Formativa.	11/03/2013.
5	Diseño de un programa con Alice. Fin de la unidad 1.	Primer ABP.	Apunte del profesor: Unidad 1.	Formativa.	13/03/2013.
6	Diseño de un programa con Alice.	Primer ABP.	Apunte del profesor: Unidad 1.	Formativa.	15/03/2013.
7	Clases, Objetos,	Primer ABP.	Apunte del	Formativa.	18/03/2013.

No. Sesión	Contenido	Estrategia metodológica	Lectura recomendada	Evaluación	Fecha de realización
	Métodos y Parámetros.		profesor: Unidad 1.		
8	Clases, Objetos, Métodos y Parámetros.	Primer ABP.	Apunte del profesor: Unidad 2.	Formativa.	20/03/2013.
9	Clases, Objetos, Métodos y Parámetros.	Primer ABP.	Apunte del profesor: Unidad 2.	Formativa.	22/03/2013.
10	Eventos.	Primer ABP.	Apunte del profesor: Unidad 2.	Formativa.	25/03/2013.
11	Eventos. Fin de la segunda unidad.	Primer ABP.	Apunte del profesor: Unidad 2.	Formativa.	27/03/2013.
12	Eventos.	Primer ABP.	Apunte del profesor: Unidad 2.	Formativa.	01/04/2013.
13	Eventos.	Primer ABP.	Apunte del profesor: Unidad 2.	Formativa.	03/04/2013.
14	Eventos.	Primer ABP.	Apunte del profesor: Unidad 2.	Entrega de las calificaciones por apreciación del estado de avance parcial de los proyectos, válida dentro del 20% de tareas, test y controles.	05/04/2013.
15	Unidad 1 y Unidad 2.	Finalización de la primera experiencia ABP.	No aplica.	Prueba Solemne No. 1	08/04/2013.

MÓDULO II. Estructuras de Programación

No. Sesión	Contenido	Estrategia metodológica	Lectura recomendada	Evaluación	Fecha de realización
16	Funciones y estructuras	Segundo ABP.	Apunte del profesor. Unidad 3.	Formativa.	10/04/2013.

No. Sesión	Contenido	Estrategia metodológica	Lectura recomendada	Evaluación	Fecha de realización
	condicionales «*if-else*».				
17	Funciones y estructuras condicionales «*if-else*».	Segundo ABP.	Apunte del profesor. Unidad 3.	Formativa.	12/04/2013.
18	Funciones y estructuras condicionales «*if-else*».	Segundo ABP.	Apunte del profesor. Unidad 3.	Formativa.	15/04/2013.
19	Funciones y estructuras condicionales «*if-else*».	Segundo ABP.	Apunte del profesor. Unidad 3.	Formativa.	17/04/2013.
20	Bucles.	Segundo ABP.	Apunte del profesor. Unidad 3.	Formativa.	19/04/2013.
21	Bucles.	Segundo ABP.	Apunte del profesor. Unidad 3.	Formativa.	22/04/2013.
22	Bucles.	Segundo ABP.	Apunte del profesor. Unidad 3.	Formativa.	24/04/2013.
23	Bucles.	Segundo ABP.	Apunte del profesor. Unidad 3.	Formativa.	26/04/2013.
24	Bucles.	Segundo ABP.	Apunte del profesor. Unidad 3.	Formativa.	29/04/2013.
25	Bucles. Fin de la Unidad 3.	Segundo ABP.	Apunte del profesor. Unidad 3.	Entrega de las calificaciones por apreciación del estado de avance parcial de los proyectos, válida dentro del 20% de tareas, test y controles.	03/05/2013.
26	Listas.	Segundo ABP.	Apunte del profesor. Unidad 4.	Formativa.	06/05/2013.
27	Listas, arreglos.	Segundo ABP.	Apunte del	Formativa.	08/05/2013.

No. Sesión	Contenido	Estrategia metodológica	Lectura recomendada	Evaluación	Fecha de realización
			profesor. Unidad 4.		
28	Arreglos.	Segundo ABP.	Apunte del profesor. Unidad 4.	Formativa.	10/05/2013.
29	Arreglos. Fin de la unidad 4.	Segundo ABP.	Apunte del profesor. Unidad 4.	Formativa.	13/05/2013.
30	Unidades 1, 2, 3 y 4: recapitulación.	Segundo ABP.	Apunte del profesor. Unidad 4.	Formativa.	15/05/2013.
31	Unidades 1, 2, 3 y 4: recapitulación.	Segundo ABP.	Apunte del profesor. Unidad 4.	Formativa.	17/05/2013.
32	Unidades 1, 2, 3 y 4.	Finalización de la segunda experiencia ABP.	No aplica.	Prueba Solemne No. 2	20/05/2013.

MÓDULO III. Estructuras de datos básicas y Transición a Java

No. Sesión	Contenido	Estrategia metodológica	Lectura recomendada	Evaluación	Fecha de realización
33	Java: sintaxis, objetos, clases, creación de nuevas clases.	Sesión expositiva a cargo del profesor, apoyada con retroalimentación desde y hacia los estudiantes.	Apunte del profesor. Unidad 5.	Formativa.	22/05/2013.
34	Java: encapsulamiento, tipos de datos.	Sesión expositiva a cargo del profesor, apoyada con retroalimentación desde y hacia los estudiantes.	Apunte del profesor. Unidad 5.	Formativa.	24/05/2013.
35	Java: tipos de datos, estructuras.	Sesión expositiva a cargo del profesor, apoyada con retroalimentación desde y hacia los estudiantes.	Apunte del profesor. Unidad 5.	Formativa.	27/05/2013.
36	Java: métodos, funciones, «*hacer en orden*», «*hacer en*	Sesión expositiva a cargo del profesor, apoyada con retroalimentación desde y	Apunte del profesor. Unidad 5.	Formativa.	29/05/2013.

No. Sesión	Contenido	Estrategia metodológica	Lectura recomendada	Evaluación	Fecha de realización
	orden».	hacia los estudiantes.			
37	Java: bucles «*for*», «*while*».	Sesión expositiva a cargo del profesor, apoyada con retroalimentación desde y hacia los estudiantes.	Apunte del profesor. Unidad 5.	Formativa.	31/05/2013.
38	Java: bucles «*for*», «*while*».	Sesión expositiva a cargo del profesor, apoyada con retroalimentación desde y hacia los estudiantes.	Apunte del profesor. Unidad 5.	Formativa.	03/06/2013.
39	Presentación de Java con BlueJ: objetos y clases.	Sesión expositiva a cargo del profesor, apoyada con ejercitación de los alumnos en el uso de la herramienta BlueJ.	Michael Kölling: «*Objects First Java*». Sección 1.1.	Formativa.	05/06/2013.
40	Presentación de Java con BlueJ: creación e invocación de objetos.	Sesión expositiva a cargo del profesor, apoyada con ejercitación de los alumnos en el uso de la herramienta BlueJ.	Michael Kölling: «*Objects First Java*». Unidad 5. Secciones 1.2 y 1.3.	Formativa.	07/06/2013.
41	Presentación de Java con BlueJ: parámetros, tipos de datos.	Sesión expositiva a cargo del profesor, apoyada con ejercitación de los alumnos en el uso de la herramienta BlueJ.	Michael Kölling: «*Objects First Java*». Unidad 5. Secciones 1.4 y 1.5.	Formativa.	10/06/2013.
42	Presentación de Java con BlueJ: instancias múltiples, estado.	Sesión expositiva a cargo del profesor, apoyada con ejercitación de los alumnos en el uso de la herramienta BlueJ.	Michael Kölling: «*Objects First Java*». Unidad 5. Secciones 1.6 y 1.7.	Formativa.	12/06/2013.
43	Presentación de Java con BlueJ: interacción entre objetos.	Sesión expositiva a cargo del profesor, apoyada con ejercitación de los alumnos	Michael Kölling: «*Objects First Java*». Unidad 5.	Control al inicio de la clase, válido por el 20%	14/06/2013.

No. Sesión	Contenido	Estrategia metodológica	Lectura recomendada	Evaluación	Fecha de realización
		en el uso de la herramienta BlueJ.	Secciones 1.8 y 1.9.	correspondiente a notas de test y tareas..	
44	Repaso general de la unidad 5.	Sesión expositiva a cargo del profesor, apoyada con ejercitación de los alumnos en el uso de la herramienta BlueJ. Comentarios del profesor en relación al desempeño del curso en el control aplicado en la clase anterior.	Michael Kölling: «*Objects First Java*». Capítulo 1.	Formativa.	17/06/2013.
45	Repaso general de la unidad 5.	Ejercitación conjunta con los alumnos.	No aplica.	Formativa.	19/06/2013.
46	Unidad 5.	No aplica.	No aplica.	Prueba Solemne No. 3	21/06/2013.

Reflexión final del capítulo

A modo de cierre de este capítulo, es necesario hacer un análisis de lo que ha sido la construcción de la planificación curricular de la asignatura, en virtud del cumplimiento de los cinco principios que se enunciaron con anterioridad.

En este caso, la secuencialidad vertical se respeta, por cuanto la asignatura parte dando una revisión básica sobre el concepto de algoritmo, y de cómo este concepto se puede transportar al diario vivir. Después, lo natural — en virtud de la opinión profesional de quien suscribe este proyecto[98] — es hacer una estructuración global — que luego dará pie a la documentación — del proyecto de software que se vaya a implementar, en función de la creación de los libretos de trabajo textual y gráfico[99]. A continuación, se revisan cuáles serán los métodos y funciones que serán necesarios de implementar con el fin de determinar las abstracciones de programación que le permitirán al estudiante incorporar el concepto de *reutilización de componentes de software*, que es útil para la construcción de grandes proyectos. Después el curso contempla la revisión de las estructuras clásicas de programación, como los condicionales y bucles, mezclados con algunos conceptos de orientación a objetos como encapsulamiento y herencia.[100]. Una vez que se han revisado todos estos componentes, se cree necesario que ellos aprendan sobre la creación de eventos, ya que a partir de este punto podrán aplicar los conocimientos ya vistos para dotar de interactividad las creaciones que desarrollen. Con estos conocimientos ya asimilados, los alumnos estarán en condiciones de aprender sobre estructuras de datos sencillas, como son las listas y los arreglos, que serán útiles para poder hacer abstracciones de orden sobre objetos y sus cambios de estado. Además, la revisión de estos tipos de datos en estas alturas del curso se justifica por el hecho que sus elementos

[98] Se considera pertinente emitir un juicio profesional del titular de esta propuesta, ya que ésta se pensó para que articule de manera dialéctica la formación del programa de Magíster en Pedagogía en Educación Superior con la experiencia personal del autor.
[99] Los guiones textuales y gráficos se explican con más detalle en la primera unidad de la asignatura.
[100] Incluso, en el apunte se enseña a mezclar todos estos componentes por medio del *anidamiento* de bloques de programación, que pasa a constituir la siguiente etapa en cuanto a nivel de dificultad.

principales se deben acceder por medio de bucles «*for*» o «*while*» que se describen en las unidades previas.

En relación al principio de continuidad, el curso completo sigue un ordenamiento que es lógico desde el punto de vista de la profundidad con que se aborden los contenidos disciplinarios, porque la primera parte — que comprende las cuatro primeras unidades — están enfocadas en enseñarle al alumno los principios básicos que rigen a la programación orientada a objetos a través de un entorno como Alice que ayuda a concretar desde el primer momento los objetos como elementos concretos de los mundos virtuales que serán capaces de construir, de tal manera que una vez que hayan adquirido un manejo mayor en la herramienta, estén en condiciones de pasar a trabajar en un entorno de desarrollo en Java como es BlueJ, que a lo mejor no es tan «*amigable*» como Alice, pero que sirve como recurso didáctico para hacer más amable la transición hacia un ambiente más duro de trabajo como es NetBeans.[101] La idea es que en la última etapa de la asignatura, se pueda hacer una recapitulación completa de todo lo que ha sido la experiencia de trabajo con Alice, y de cómo los estudiantes serán capaces de florecer los conceptos e ideas más importantes de la programación orientada a objetos, para poder llevarlos a un ambiente como BlueJ donde deberán lidiar con una característica propia de cada lenguaje, y que hasta el momento permanecía encubierta: la sintaxis.

También, el orden en que se revisan los contenidos respeta el principio de continuidad, porque hay que considerar que esta es una asignatura que está pensada para estudiantes de primer año de universidad, que tal vez nunca hayan tenido alguna experiencia concreta en el ámbito de la programación. Entonces, desde esta perspectiva es válido que de un principio a ellos se les inculque la idea que antes de usar una herramienta, es necesario que hagan un diseño de lo que quieran hacer, y que luego ese libreto de trabajo lo transformen en un producto de software. Y después, se revisa lo comentado en el párrafo anterior, para luego

[101] Las características de Alice y de BlueJ se describen con más precisión en el capítulo 3.

pasar al «*siguiente nivel*», que es el trabajo con lenguajes de programación que requieren entornos reales de desarrollo.

Con respecto a la integración con otras asignaturas, es posible que los alumnos puedan desarrollar simulaciones que sean propias de su ámbito de especialidad, como por ejemplo alguna experiencia de laboratorio o de algún proceso industrial. También se pueden construir proyectos de software que ayuden a explicar fenómenos físicos o químicos que se impartan en los cursos respectivos, aunque éstos no vayan dentro del mismo nivel curricular que el de «*Tecnologías de la Información*».[102]

En relación a la congruencia horizontal, los contenidos disciplinarios obedecen de manera efectiva a los objetivos de aprendizaje que se hayan trazado para esos fines, tal y como aparece declarado al inicio de cada unidad didáctica. Los recursos didácticos se muestran en la planificación de clase a clase. Por último, el principio de comunicabilidad se materializará a través de la plataforma Moodle que se habilitará para la asignatura, donde los objetivos de aprendizaje serán conocidos de manera explícita por los estudiantes.

Por otra parte, la planificación didáctica se ha levantado en la figura del syllabus que la Universidad San Sebastián ha instaurado como el documento oficial en la cual el profesor comunica a los estudiantes sobre las actividades que vaya a desarrollar clase a clase, procurando mantener una correspondencia biunívoca entre lo declarado en el programa de la asignatura con lo que se irá trabajando en cada una de las sesiones, empleando los aspectos declarados en la introducción del capítulo: estrategia metodológica a emplear, recursos didácticos que sean necesarios, y el tipo de evaluación que se vaya a practicar.[103] Además, las actividades propias de inicio, desarrollo y cierre de una clase se exponen en el anexo B donde vienen descritas las actividades de ABP.

[102] Sería posible, por ejemplo, que en un curso de física a los estudiantes se les planteara un problema abierto que desafíe a los estudiantes a generar algún producto de software que sirva para explicar lo que ocurre mientras un avión está en marcha, como se expone en la creación de un simulador de vuelo con Alice.
[103] En el caso de esta asignatura, la evaluación de tipo formativa es la que más se va a emplear, por cuanto ésta permite obtener información sobre el estado de avance parcial de los aprendizajes de los estudiantes.

También es bueno precisar que la última unidad didáctica de la asignatura se va a conducir de la manera tradicional, ya que en función del juicio profesional del autor, es conveniente que el profesor asuma el rol de experto en la materia que se va a preocupar de organizar la manera en cómo los estudiantes van a lograr una transición hacia los saberes que son propios de lenguajes como Java, por medio de actividades que no necesariamente hagan ver a los alumnos como meros receptores pasivos de la información, sino que se va a procurar que ellos sigan trabajando en solitario, o en pequeños grupos el apunte que se les entregue, por medio de confección de mapas conceptuales o diagramas sinópticos que ayuden a clarificar de mejor manera las ideas que se expongan, sin necesidad de abusar de la comunicación unidireccional. Además, El objetivo que este documento esté disponible en Moodle, es para que los alumnos adelanten la lectura del material para la clase que corresponda, y así lograr que el aprendizaje se pueda desarrollar de manera más eficiente.

Se espera que el trabajo que los estudiantes hayan realizado durante los dos meses aproximados bajo la experiencia ABP de los frutos necesarios, para que al llegar a la quinta unidad no se produzca el fenómeno descrito por Villalobos[104] del aprendizaje por imitación, que es precisamente lo que se intenta mitigar.

[104] Villalobos, J. (2006). *El reto de diseñar…*

Capítulo 5

Conclusiones

Introducción

A partir de todo el trabajo que se ha realizado en este proceso, emergen las conclusiones que se muestran en este capítulo, que tienen como objetivo principal comunicar las reflexiones que aparecen desde la visión innovadora de este proyecto de intervención pedagógica, junto con una síntesis de lo que ha sido un arduo trabajo que se ha extendido por cerca de un año.

5.1 Conclusión de la propuesta

En este trabajo se ha levantado una propuesta de intervención e innovación pedagógica en el ámbito de la programación orientada a objetos, donde la Universidad San Sebastián ha pasado a ser la unidad donde este proyecto se va a implementar.

Lo innovador de lo que se ha formulado en este informe viene dado por el hecho que esta será la primera vez que se trabaje con el aprendizaje basado en problemas como estrategia metodológica para apoyar la enseñanza de la programación a estudiantes de plan común. De hecho, en la Facultad de Ingeniería y Tecnología no existen registros que den cuenta de la aplicación de una experiencia pedagógica de esta naturaleza, a partir de la información que tanto la dirección de carrera como la coordinación de cursos de primer año han proporcionado para estos fines. Sin embargo, la tarea no es sencilla, ya que la implementación de esta técnica implica un profundo trabajo de planificación por parte del docente y de la selección de las actividades adecuadas que permitan al final de cada una de las actividades mostradas en el anexo B una apropiada ejecución del aprendizaje colaborativo entre los estudiantes.

Por otro lado, la enseñanza de la programación orientada a objetos había contemplado el uso de herramientas de desarrollo profesional[105] para apoyar la praxis didáctica de los docentes, y que no tuvieron una buena acogida por parte de los estudiantes, como lo reflejan los resultados de la investigación. Es por eso que se propuso la utilización de recursos didácticos como Alice y BlueJ que tienen una clara orientación pedagógica, ya que están categorizados como software de corte educativo, que permiten al estudiante sobrellevar de mejor manera la dificultad inicial para manejar las abstracciones mentales que deban emprender para lograr un mejor entendimiento de los conceptos e ideas fundamentales que son propios del paradigma de la orientación a objetos, donde el tratamiento con las clases instanciadas en los mundos virtuales le permite al estudiante experimentar una vivencia mucho más concreta y real con los elementos de programación, al concebir un algoritmo como una secuencia de acciones que se deben ejecutar para poder dar vida a una historia o a un videojuego, como en el caso de los problemas propuestos que se describen en la primera experiencia de aprendizaje basado en problemas.

Por tanto, la aplicación conjunta entre el ABP como estrategia metodológica de apoyo al proceso de enseñanza-aprendizaje junto a Alice y BlueJ generan una inédita combinación de herramientas pedagógicas que se pondrán en marcha al interior de esta casa de estudios, ya que es factible llevar a cabo una iniciativa como esta, porque: (1) la propia Universidad promueve que los docentes puedan hacer algún tipo de innovación pedagógica, siempre y cuando ésta no contradiga los objetivos de aprendizaje que se quieran alcanzar; (2) en términos técnicos el parque computacional es suficiente para satisfacer la demanda estudiantil, tanto en cantidad como en las características técnicas que configuran sus prestaciones individuales, que sobrepasan con creces los requerimientos mínimos que los desarrolladores de Alice y BlueJ exigen para que estos productos funcionen de manera apropiada; y (3) en lo financiero no es necesario adquirir nueva maquinaria o invertir dinero con conceptos de licencia de software o de capacitación en ABP.

[105] Como NetBeans o Eclipse.

Conviene destacar en esta parte que los otros aspectos propios del desarrollo de la enseñanza también aparecen dentro de la propuesta, aunque fueran de manera implícita. Por ejemplo: la interacción pedagógica es la que se da clase a clase por medio de la retroalimentación mutua que se genere entre el profesor y cada uno de los grupos de trabajo que se hayan conformado, junto con el tipo de desafíos que están planificados al término de cada una de las unidades de aprendizaje que están al final del texto de apoyo, cuyo orden de complejidad va aumentando a medida que se avanza con la lectura, en la cual es posible identificar algunas orientaciones de tipo pedagógico para que el profesor tenga en cuenta para abordar ciertas cuestiones atingentes a la enseñanza de la programación con Alice.

Dentro de la metodología del curso se pensó en la ayudantía como la instancia para que los alumnos aprovechen de reforzar los conceptos y técnicas de programación que se revisan durante el semestre académico, de tal manera que puedan complementar el trabajo que hagan en clase con el reforzamiento de ideas que puedan desarrollar en las sesiones de práctica. La organización de tiempo y del espacio viene dada por la propia metodología ABP, ya que ésta prescribe la ubicación de los alumnos en grupo de trabajo, y el desglose del tiempo se ha hecho de tal manera que en cada sesión existan las actividades clásicas de inicio, desarrollo y cierre que no siempre se daban en las sesiones de clase que se observaron. El clima de aula siempre ha sido percibido como bueno dentro de la Universidad, por lo que se ha considerado que este aspecto no reciba algún tipo de recomendación para su mejoramiento.

Es relevante mencionar la importancia que exista una promoción de un efectivo diálogo interdisciplinar entre las asignaturas que cursen los estudiantes, para que el proceso de aprendizaje autónomo del estudiante se pueda nutrir de manera apropiada de los saberes que haya adquirido en otros cursos — que haya tomado o que esté tomando en paralelo —, para que en su estructura cognoscitiva vaya generando una globalidad unificada de conocimientos, que tienden a ser más duraderos que los obtenidos bajo el enfoque convencional. Por lo tanto, este

trabajo se constituye en una buena oportunidad para proponer un cambio gradual en cuanto al tipo de metodología de enseñanza que se practica hasta el momento en la Universidad San Sebastián.[106]

Al implementar un proyecto con las características antes descritas, se pueden obtener beneficios que van en la línea de una optimización de la praxis pedagógica en el ámbito de la didáctica de la programación orientada a objetos, desde el punto de vista docente, del alumno y de la propia institución, tal como se expone en la Tabla 6:

Tabla 5. Beneficios del proyecto para cada entidad involucrada (elaboración propia)

Estamento / Unidad	En qué se beneficia
Institución	Tiene la posibilidad de ser pionera en Chile por implementar una combinación pedagógica entre el aprendizaje basado en problemas como estrategia metodológica, junto con el uso de potentes herramientas de enseñanza de la programación orientada a objetos, como son Alice y BlueJ, cuyas características más notables se describen en las secciones 3.3.1 y 3.3.2. Se dice pionera, ya que no se han encontrado registros formales del empleo de Alice y BlueJ dentro del concierto educativo chileno: a lo más, se conoce de unos hipervínculos que un profesor de la Universidad de Santiago de Chile ha publicado en su página web[107], donde se menciona a BlueJ como un recurso que se puede emplear para la enseñanza de Java. Esto significa que la Universidad San Sebastián en el mediano plazo puede llegar a convertirse en un referente autorizado en cuanto al uso simultáneo de estas herramientas educativas para apoyar la enseñanza de este paradigma, ya que a partir de esta experiencia se pretende construir y publicar un artículo que dé cuenta de lo que signifique la puesta en marcha de este proyecto en algún congreso chileno de educación en Ingeniería.

[106] Lo que tiene directa relación con la subcategoría C.2 Cultura institucional en materias específicas de tipo pedagógicas.
[107] http://fermat.usach.cl/~msanchez/

Estamento / Unidad	En qué se beneficia
Docente.	Podrá incorporar a su repertorio de estrategias de enseñanza la metodología de aprendizaje basado en problemas, que le permitirá abordar sus clases desde una óptica en la cual ellos no pasan a ser el centro de la atención — como ocurre con el formato academicista tradicional —, sino que ahora las actividades curriculares se enfocarán desde el enfoque de presentar alguna situación problemática que sirva de etapa inicial para que los estudiantes sean capaces de construir su propio aprendizaje. Esto representa un desafío notable para el profesor, ya que las clases se estructuran de acuerdo a la planificación de cada una de las experiencias ABP que se detallan en el anexo B. Además, el profesor tiene la posibilidad de enseñar a programar a través de recursos didácticos que tienen una clara orientación educativa, donde la interfaz que ambas herramientas le brindan al estudiante lo estimulan a aprender por su propia cuenta, a través de entornos que fomentan su libertad creadora.
Estudiante.	Puede asimilar nuevas técnicas de aprendizaje, ya que el curso propuesto estimula el trabajo colaborativo que se da a través de las distintas actividades ABP, que le ayudarán a desarrollar sus habilidades de programación por medio de la interacción con recursos didácticos como Alice y BlueJ que han sido especialmente diseñados para estos fines. Además, se espera que la adquisición de estos conocimientos se logre de mejor manera que en el enfoque tradicional, ya que estos medios le ayudan al alumno a hacer una contextualización mucho más real de lo que es la programación, en el sentido que estas herramientas le permiten abstraerlo de la complejidad inherente que tiene el desarrollo de la capacidad de abstracción. El texto de apoyo a la docencia es clave en este sentido, ya que los ejemplos que se presentan están pensados en esta dirección.

Los productos que se pueden obtener a partir de la formulación de este proyecto de intervención pedagógica, son los que se enumeran a continuación:

1. Un programa actualizado para la asignatura «*Tecnologías de la Información*» que incorpore la estrategia metodológica del aprendizaje basado en problemas, los recursos didácticos Alice, BlueJ, Moodle, YouTube, el desglose de los objetivos de aprendizajes en sus dimensiones cognitivas, procedimentales y actitudinales para cada una de las cinco unidades didácticas que componen la asignatura, junto con la bibliografía recomendada para estos propósitos.
2. El syllabus que la Universidad ha definido para estos propósitos, que refleje clase a clase lo que se ha formulado en el programa del curso.[108]
3. Explicitación de cada una de las actividades de aprendizajes basado en problemas que se han construido para la asignatura, que vienen a complementar el syllabus del segundo punto.[109]
4. El material de apoyo que se va a ocupar para cubrir la parte del desarrollo del pensamiento algorítmico en el estudiante por medio del uso de la herramienta Alice.[110]
5. La plataforma Moodle donde va a funcionar de manera virtual la asignatura, en la cual se incluirá el material didáctico que se vea en clase, apuntes de trabajo de libre distribución, como el texto de la historia de los computadores de Héctor Sanjuán.[111]
6. Una cuenta en YouTube que ha sido construida para que los estudiantes comuniquen las creaciones que hayan realizado en Alice, con el objetivo de difundir los trabajos realizados con el resto de la comunidad.

Una vez que lo propuesto se haya puesto en acción, se construirá un artículo de carácter científico que informe a la institución de la puesta en marcha de este proyecto de innovación pedagógica que plantea una forma de enseñar a

[108] Los productos 1 y 2 se muestran en el cuarto capítulo.
[109] Ver anexo B.
[110] Ver anexo C.
[111] Disponible en la URL http://www.educacionpersonal.com/

programar bajo el paradigma de la orientación a objetos a estudiantes de Ingeniería bajo la estrategia metodológica del aprendizaje basado en problemas.

5.2 Conclusión general de la tesis

La enseñanza de la programación orientada a objetos está directamente relacionada con la maduración del pensamiento algorítmico por parte de los estudiantes. La primera parte de este trabajo da a conocer los resultados de una investigación que siguió los lineamientos de la metodología cualitativa, que comprende una descripción de las características de la praxis docente en el ámbito de la didáctica de este paradigma computacional en la Universidad San Sebastián — que van desde la planificación y organización de la enseñanza hasta el desarrollo de la misma —, junto a una explicación del fenómeno desde aquellos aspectos que dan cuenta de las concepciones pedagógicas de la institución hasta las creencias educativas personales de los estamentos involucrados, siendo la triangulación hermenéutica el procedimiento empleado para procesar y analizar los datos provenientes de los instrumentos que se aplicaron.

Este estudio se hizo con la asignatura «*Diseño y Construcción de Software*» que se imparte para alumnos de segundo año de las especialidades de Ingeniería que ofrece esta casa de estudios, como una forma de comprender cuáles son los problemas que los alumnos arrastran hacia cursos posteriores. Dentro de los resultados obtenidos se puede decir que la metodología de enseñanza aplicada por los docentes se basa en el aprendizaje por imitación bajo un formato academicista clásico de clase expositiva, donde los estudiantes son receptores pasivos de la información entregada por el profesor. Además, los alumnos se agrupaban por afinidad, y el tiempo de clase no era suficiente para alcanzar a desarrollar a plenitud las actividades propuestas. Por otro lado, la Universidad fomenta que los docentes sean libres para realizar sus cátedras del modo que estimen conveniente, sin que tengan que ceñirse a alguna metodología didáctica

específica, ni tampoco se ha fomentado en el último tiempo que ellos hagan un perfeccionamiento en Educación.

El aporte principal que hace en primera instancia lo constituye un diagnóstico pedagógico que ayuda a brindar un soporte empírico que dio origen a la propuesta de intervención e innovación pedagógica que ha inspirado la construcción de este proyecto, y cuyo norte apunta a buscar una optimización en cuanto a la manera en cómo se conduce el ejercicio didáctico de la enseñanza de la programación orientada a objetos, por medio de dos grandes aristas: el empleo del aprendizaje basado en problemas como estrategia metodológica para conducir el proceso de enseñanza-aprendizaje, y de las herramientas Alice y BlueJ como recursos didácticos que pretenden simplificar el aprendizaje de esta disciplina de las Ciencias de la Computación, a objeto que los estudiantes queden capacitados para enfrentar con mejores armas el curso de «*Diseño y Construcción de Software*».

Desde otra óptica, el valor de lo que se ha hecho hasta el momento reside en la posibilidad de poder escalar este trabajo a otras asignaturas que sean de esta línea programática dentro de la carrera de Ingeniería Civil Informática en la Universidad San Sebastián, en el sentido de poder innovar la praxis pedagógica usando el enfoque del aprendizaje basado en problemas.

También este proyecto se puede insertar dentro de un programa de enseñanza de la programación que pueda comenzar desde la educación secundaria, ya que hoy en día algunos establecimientos[112] imparten talleres de educación tecnológica, donde la Robótica ha pasado a ser una de las grandes atracciones de estudio para los jóvenes, ya que los invita a palpar de manera práctica conceptos que se ven en Física, Mecánica, Electrónica, Matemática, Informática, donde el comportamiento de los robots se programan a través de lenguajes que siguen las directrices del paradigma de la orientación a objeto. Al

[112] Como por ejemplo, el caso de los liceos y colegios que participaron en el evento «*Expo Roboótica*» que se celebró en la Universidad San Sebastián el año 2012: Liceo Mauricio Hochschild del Centro de Estudios de Alta Tecnología, Instituto San Pedro, Colegio Concepción — todos de la comuna de San Pedro de la Paz —; Centro Educación Evangélico de la comuna de Hualpén; Liceo El Refugio de Penco; Colegio Etchegoyen de Talcahuano; Liceo de la Madera de Coronel; Escuela Lautaro de Nacimiento.

respecto, existen herramientas como Logo[113], Scratch[114] o el propio Alice que permiten adquirir desde temprana edad las habilidades necesarias que ayuden al aprendizaje de esta importante arista del conocimiento tecnológico que un estudiante debiera tener.

También es conveniente reflexionar en esta instancia final de la tesis sobre la importancia que tiene el que un profesional que no haya tenido alguna formación inicial en educación, y que se dedique a la docencia, cuente con alguna instrucción formal en materias pedagógicas, debido a la tremenda cuota de responsabilidad que a un profesor le cabe en cuanto a la formación académica y humana de las futuras generaciones.

El trabajo realizado ha servido para valorar la formación que ha entregado este programa de Magíster, que ha permitido comprender que no basta con tener el talento pedagógico para enseñar, o con poseer los conocimientos disciplinarios, sino que eso se tiene que trabajar, porque existen técnicas y metodologías que sirven para conducir con éxito un clase que seguramente escapen al sentido común de quien enseñe sin saber cómo hacerlo. Por ejemplo: saber y tener la oportunidad de conocer estrategias metodológicas de enseñanza diferentes a las del enfoque convencional de enseñanza — como el caso del ABP y de la gama de actividades que se pueden hacer al interior del aula, como se comentaba en el capítulo 3 —, o de la correcta utilización de recursos didácticos como software educativo que sirvan para apoyar el proceso de enseñanza-aprendizaje, por citar algunas de las maneras en cómo se desarrolla la praxis didáctica.

También haber tenido la oportunidad de leer e investigar — a través de bibliografía especializada y de entrevistas en profundidad con diferentes

[113] Es un software educativo creado por Seymour Papert, y que puede ser usado para enseñar a programar, ya que trae soporte para poder trabajar con listas, archivos y dispositivos de entrada y salida. En este caso, los retos que propone este lenguaje es ayudar al niño a desarrollar habilidades metacognitivas que le permiten poner en práctica procesos de autocorrección, donde los comandos están disponibles en español, y afectan a una tortuga virtual a las que se le entregan instrucciones para crear dibujos. Su uso se recomienda a partir de los 10 años de edad.

[114] Es un lenguaje de programación visual desarrollado por el *Lifelong Kindergarten group* en el Media Lab del MIT, orientado para la enseñanza de niños que tengan de seis años en adelante. Se caracteriza por ser gratuito, multiplataforma, estar disponible en español. La idea es que al niño le resulte extremadamente intuitiva la manera de ir disponiendo los distintos bloques, para ir creando programas y animaciones.

estamentos universitarios — sobre la importancia que tiene el hecho de hacer una debida planificación de la enseñanza, tanto en los ámbitos curricular como didáctico, y de comprender que estas actividades van allá de rellenar una tabla: permiten tener una visión holística de lo que va a ser el proceso educativo a lo largo de un semestre académico, que obliga a reflexionar profundamente sobre la manera en cómo conduce el proceso de enseñanza-aprendizaje entre los alumnos, en torno a: la secuencialidad lógica de los contenidos que se vayan a desarrollar; el diálogo interdisciplinar que permita construir aprendizajes en la estructura cognoscitiva del estudiante como una globalidad unificada, que se pueda transportar a lo largo de su preparación académica; determinar cuáles serán las actividades que se desarrollarán a lo largo de una o varias sesiones para que los estudiantes puedan adquirir los aprendizajes esperados; la elección de los recursos didácticos y bibliográficos que sean pertinentes de emplear, en base al juicio profesional del docente y de su experiencia como educador; o la propia manera en cómo se van a medir la adquisición de los saberes y actitudes que un estudiante debiera tener adquirido hasta un determinado instante, por medio del tipo de evaluación que se considere más apropiada.

Capítulo 6

Validación

Introducción

En este capítulo se estudiará la viabilidad de la propuesta que se ha levantado a lo largo de este informe, junto al estudio sobre la implementación de la misma. El objetivo de esta parte es demostrar con indicadores que este proyecto de intervención e innovación pedagógica se puede implementar dentro de la unidad de estudio donde se hizo la investigación, que en este caso corresponde a la Universidad San Sebastián. Para estos efectos, la demostración se hará en torno a tres ejes temáticos: institucional, operacional y financiero. La segunda parte presenta un plan de validación, que permita darle una legitimidad teórica a lo que se ha expuesto a lo largo de los capítulos predecesores, que se realizó en base a las opiniones de los estamentos universitarios que permitirán validar lo que se ha propuesto: director de carrera, directora de plan común, docentes y estudiantes.

6.1 Viabilidad de la propuesta

6.1.1 Desde el punto de vista institucional

En su parte explicativa, la investigación realizada revela que la Universidad San Sebastián promueve que los docentes puedan implementar algún tipo de innovación pedagógica, siempre y cuando ésta no vaya en contra de los objetivos de aprendizaje de los alumnos. En este sentido, esta propuesta tiene como objetivo fortalecer las competencias de programación que vayan adquiriendo los estudiantes a lo largo del transcurso de la asignatura, de acuerdo a lo que el plan de estudios de la asignatura establece dentro de su formulación: «*Este curso busca fomentar una visión crítica en el alumno en las tareas relacionadas con el análisis de problemas simples del mundo real, a los cuales se buscará dar solución mediante técnicas de lógica de programación. Esto permitirá al alumno*

desarrollar el pensamiento sistémico, lógico y analítico, aprendiendo a resolver problemas complejos, con una clara orientación al trabajo en equipo.»

6.1.2 Desde el punto de vista operacional

La asignatura «*Tecnologías de la Información*» es un curso de carácter teórico-práctico, donde el laboratorio de computación se constituye en el escenario ideal para poder desarrollar las clases y las sesiones de ayudantía.

El ámbito técnico se refiere a la factibilidad de poder emplear la infraestructura computacional de la Facultad de Ingeniería y Tecnología de la Universidad San Sebastián para poder ejecutar sin problemas el recurso didáctico Alice, desde el punto de vista de la instalación como de la puesta en marcha del mismo. Los requerimientos computacionales mínimos que demanda esta herramienta son los que se indican a continuación:[115]

Requerimientos de hardware de Alice

El computador debe contar con un procesador que funcione a más de 500 MHz, tener al menos 250 MB de espacio disponible en disco duro, 1 GB de RAM instalados como mínimo, tarjeta de video que soporte una resolución mínima de 1024x768 pixeles, tarjeta de sonido, y ratón de dos o tres botones.

Requerimientos de software de Alice

Tener una máquina cuyo sistema operativo sea Windows XP, Vista de 32 bit, Vista de 64 bit, Windows 7 de 32 bits, Windows 7 de 64 bits, Windows 8, Ubuntu, Red Hat — para el caso de Linux —, o Mac OS X 10.4 o superior, para los equipos Apple.

[115] Estos datos fueron extraídos desde la URL oficial del proyecto Alice: http://help.alice.org/w/page/54959064/System%20Requirements

Por otra parte, se trabajará con la versión 3.0.8 de BlueJ, cuyos requisitos se exponen a continuación:[116]

Requerimientos de hardware de BlueJ

El computador debe contar con un procesador Pentium III que funcione a 400 MHz o más, y 128 MB de memoria principal.

Requerimientos de software de BlueJ

BlueJ puede funcionar en máquinas que tengan sistemas operativos como Windows 2000, XP, Vista o 7; Debian o Ubuntu; o bien Mac OS X. Además, el computador debe contar con Java 5 JDK o posterior.

Por otra parte, en la Tabla 6 se presentan las características técnicas más relevantes de los equipos computacionales que componen los tres laboratorios de la Facultad de Ingeniería y Tecnología de la Universidad San Sebastián. Cabe destacar que todos los equipos que conforman una sala tienen entre sí las mismas características técnicas.[117]

Tabla 6. Características técnicas del parque computacional.

Laboratorio	Cantidad de Computadores	Procesador	Memoria Principal	Disco Duro	Video[118]	Sistema Operativo
F.301	30, pero 26 operativos.	Intel Core 2 Duo de 2.3 GHz.	2 GB.	160 GB.	256 MB	Windows 7 Ultimate 32 bits.
F.303	30, y 25 operativos.	Intel Core 2 Duo de 2.3 GHz.	2 GB.	160 GB.	256 MB.	Windows 7 Ultimate 32 bits.
F.304	19.	Intel Core i5 de 2.3 GHz.	4 GB.	500 GB.	256 MB.	Windows 7 Ultimate 64 bits.

[116] Estos datos fueron extraídos desde la URL oficial del proyecto BlueJ: http://www.bluej.org/help/requirements.html
[117] Fuente: Información proporcionada por la administradora de los laboratorios de computación de la Facultad de Ingeniería y Tecnología de la Universidad San Sebastián.
[118] Todos los computadores funcionan con una resolución de pantalla de 1024 x 768 pixeles, a 60 Hz.

En primer lugar, el requerimiento de procesador se cumple de sobremanera, ya que todos ellos funcionan a más de 500 MHz, como indica la especificación del proyecto Alice. Además, el hecho de contar con CPU que operen con dos o más núcleos, permite asegurar una ejecución más fluida de ambas aplicaciones. En cuanto a memoria principal, tampoco existen inconvenientes técnicos, dado que la cantidad instalada en cada uno de ellos resulta ser mayor o igual a 2 GB, que supera con creces a lo que ambas herramientas necesitan para operar de manera apropiada.

Con respecto al espacio disponible en disco duro, es preciso señalar que una de las políticas de administración del laboratorio consiste en dejar instalado un programa que permite restaurar el computador cada vez que se reinicia, lo que en términos prácticos significa que una configuración predeterminada se vuelve a cargar cada vez que la máquina se inicia. En particular, esto es demasiado importante, ya que será necesario generar el hábito en los alumnos de respaldar por su propios medios — ya sea en alguna unidad flash, cuenta de correo o servicio en la nube[119] — el trabajo que haya estado haciendo en alguna sesión de clase, ya que los datos de los usuarios se pierden cuando el computador se apaga o reinicia.[120] Y dado que el disco duro tiene una partición única para el sistema operativo, es posible garantizar que el requisito de contar con un espacio libre de al menos 250 MB se satisface.

En cuanto al video, los computadores trabajan con una resolución de pantalla igual a 1024 x 768 pixeles, lo que permite asegurar que esta exigencia se cumple. Por otra parte, la información proporcionada por la administración de laboratorios revela que las máquinas presentes cuentan con tarjeta de sonido, y que se encuentra plenamente habilitada para funcionar.[121] El tipo de ratón que se necesita para trabajar es uno estándar, por lo que este requisito pasa a ser trivial.

[119] Como Dropbox, SkyDrive o Google Drive.
[120] Como los alumnos son «*nativos digitales*», se cree que estas costumbres serán asimilada con mucha facilidad por parte de ellos.
[121] Es importante destacar esto, ya que el sonido muchas veces contribuye a la generación de espacios de trabajo donde resulta ser un factor que atenta contra la productividad.

Aunque no aparezca mencionado dentro de los requisitos de operatividad, conviene destacar que las tasas de transferencia — que bordean entre 30 y 300 kilobytes por segundo para bajada y 9 kilobytes para subida — son apropiadas para descargar el material que se publica en la plataforma Moodle, y también para hacer lo propio con los archivos grupales de trabajo. Además, el acceso a YouTube no está restringido dentro de los laboratorios, por lo que no existen inconvenientes para que los estudiantes puedan acceder a este servicio.

Con respecto al software, la Tabla 6 indica que Microsoft Windows pasa a ser el sistema operativo que está instalado en cada una de las máquinas, ya sea en 32 o 64 bits. Independiente de cuál sea, cumple con el requisito exigido por los desarrolladores de Alice, y también de BlueJ, ya que ambos programas se han ejecutado sin problemas en máquinas que tienen cualquier versión de esos sistemas operativos instalados.

Por lo tanto, todos los computadores que pertenecen a la Facultad de Ingeniería y Tecnología de la Universidad San Sebastián se pueden utilizar, ya que cada uno de ellos satisface a cabalidad los requerimientos técnicos que demandan los desarrolladores del proyecto Alice. Con respecto a la disponibilidad de equipos por alumno, es posible asegurar que esto tampoco es un problema, ya que por una parte los laboratorios de computación se caracterizan por tener horarios de funcionamiento que tienen grandes ventanas de tiempo que los estudiantes pueden aprovechar para avanzar con sus proyectos.

La versión con la que se desea trabajar es la 2.3, que tiene la particularidad de contar con una interfaz nativa en español, lo que en cierta medida ayuda a reducir la barrera idiomática entre el estudiante y el software. La más actualizada es la 3.1, pero ésta aún se encuentra en etapa de evaluaciones, lo que significa que por el momento su uso no está recomendado, ya que no se trata de un producto debidamente terminado.

6.1.3 Desde el punto de vista financiero

Desde una óptica monetaria, es preciso señalar que Alice es un software gratuito que se puede descargar desde la plataforma web oficial del proyecto —www.alice.org —, donde los autores autorizan a que cualquier persona lo pueda instalar y emplear como recurso didáctico para apoyar la enseñanza. Por lo tanto, la Universidad San Sebastián no deberá incurrir en algún gasto que se pueda derivar a partir de este concepto.[122] La instalación de Alice no requiere de grandes conocimientos computacionales, ya que sólo requiere la ejecución de un archivo ejecutable, lo que en este contexto significa que no es necesario pagar para poner en marcha esta herramienta. Además, el docente tendrá libre acceso a las funcionalidades adicionales que están presentes en alice.org:

- Descargas de otras versiones del proyecto.
- Galerías de sonidos y objetos, que sirven para enriquecer la materia prima de los desarrollos de proyectos.
- Tutoriales en línea.
- Foros donde se puedan intercambiar experiencias, consultas y sugerencias con otros usuarios y colegas docentes, inclusive con los miembros del equipo que desarrollaron esta herramienta.[123]
- Descarga de artículos y de testimonios que dan cuenta de la experiencia de uso de esta plataforma educativa.

Por otra parte, el material instruccional será el que se adjunta en este informe de tesis, y que no representará costo para la Universidad, ya que será tratado como parte de los recursos didácticos que elabora el docente, y de esta manera no será necesario comprar libros adicionales de Alice.

En este informe se menciona también que se hará uso de la plataforma Moodle, que servirá de soporte tecnológico para albergar todo el material

[122] El proyecto Alice se financia a través de instituciones y empresas que lo patrocinan, como el caso de la Universidad de Carnegie Mellon, Oracle, Electronic Arts, The National Science Foundation, Google, Disney, Hyperion, DARPA, The Heinz Endowments, y The Hearst Foundations.
[123] Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, Steve Cooper, Jacobo Carrasquel, Caitlin Kelleher, Cleah Schlueter.

pedagógico que se distribuya a los estudiantes. Este recurso es software libre, lo que implica — entre otros aspectos — una libertad total para poder usar esta herramienta para los propósitos que sean necesarios. Tampoco hay que pagar por el uso de licencias. Por otro lado, Moodle debe operar en un servidor, cuyo uso está asociada con una cuenta que el autor tiene en un servidor, y que dispondrá de manera libre para su uso en beneficio del correcto desarrollo de las actividades curriculares de la asignatura, lo que incluye la administración de todo lo concerniente al manejo del espacio virtual asignado para estos fines.

Tampoco será necesario invertir en el parque computacional de la Facultad, ya que la capacidad instalada es suficiente para satisfacer la demanda interna de los estudiantes y docentes, considerando que los primeros cuentan con una máquina portátil que la Universidad le provee cuando ingresan a la carrera, de tal manera que no debiera existir una figura de carencia de maquinaria en este sentido.

En el caso de BlueJ, tampoco será necesario hacer una inversión financiera para adquirir el software, ya que éste se distribuye de manera gratuita, al igual que el tutorial preparado por Michael Kölling, y que fue traducido al español por Germán Bordel. La restricción que impone el autor, es que este documento se puede utilizar sin que su empleo esté orientado a la generación de lucro por parte del docente de la institución, tal como reza la sección de copyright, licencia y distribución de esta herramienta que se lee con claridad en el manual de la aplicación.[124]

Por otra parte, YouTube es otro de los recursos didácticos que se va a emplear, y que tiene costo cero en cuanto a uso, al igual que el texto de la historia de los computadores de Héctor Sanjuán.

[124] Kölling, M. (2005). *El «tutorial» de BlueJ*. Dinamarca: Editado por The University of Southern Denmark.

6.2 Plan de validación

Al momento que este trabajo se exponga, la propuesta que acá se encierra no se habrá implementado, ya que los tiempos de trabajo en una y otra actividad no coinciden. Sin embargo, es posible recurrir a una solución instrumental que consiste en la elaboración de un plan de validación con estructura de cronograma, en el cual se explicará la manera de cómo este trabajo se llevará a cabo en la Universidad San Sebastián, con el fin de alcanzar la validez y confiabilidad de la misma,

6.2.1 Validación de los estamentos

Los resultados que se exponen a continuación, corresponden a la síntesis de lo que fueron entrevistas que se practicaron a los estamentos director de carrera, directora de plan común de Ingeniería, docente y estudiante. Los docentes hacen clases en la Universidad San Sebastián, y también en otras casas de estudio, como el Instituto Profesional Virginio Gómez y el Instituto Profesional AIEP. Los estudiantes son alumnos regulares de la carrera de Ingeniería Civil Informática de la Universidad San Sebastián, y están cursando el segundo y tercer año.

Director de carrera. Se sostuvo una reunión con el director de carrera de Ingeniería Civil Informática de la Universidad San Sebastián, a quien se le comentó de manera detallada el tenor de esta propuesta, tanto en su génesis — expresada en una síntesis de los resultados de la investigación — como en su formulación general, caracterizada mediante el uso del aprendizaje basado en problemas, junto con la incorporación de las herramientas Alice y BlueJ para apoyar el proceso de enseñanza-aprendizaje. El director se mostró de acuerdo y entusiasmado con lo expuesto, en virtud del sentido innovador que representa la implementación del ABP en Ingeniería, ya que a su juicio sería una experiencia inédita dentro de la carrera. Además, el uso de estos recursos

didácticos le pareció interesante, desde el punto de vista que permiten romper con el molde con que tradicionalmente se enseña la programación orientada a objetos, ya que le permite al alumno construir el conocimiento a partir de nociones concretas que lo conecten de mejor manera con lo abstracto de las construcciones algorítmicas propiamente tales. Además, se mostró interesado en hacer observación metodológica de aula para poder aprender más acerca del ABP, y de cómo se desarrolla la combinación con herramientas como Alice y BlueJ.

Directora de Plan Común de Ingeniería. [125] Ella opinó que le parece adecuada la incorporación del ABP como estrategia metodológica, ya que hace algunos años atrás se había barajado la idea de incorporarlo en el aula, pero la idea no prosperó. Con respecto a Alice y BlueJ, manifestó que desconocía la utilidad de esas herramientas, y que le parece positivo que se pueda innovar la enseñanza de la programación por medio del uso del uso de software educativo, considerando que esta es una habilidad transversal que cualquier profesional de la Ingeniería debiera poseer. Subrayó el hecho que como esta es una asignatura que es común para todas las especialidades de Ingeniería, era necesario contar con la aprobación de la Vicerrectoría Académica para poder actualizar este curso en todas las ciudades donde la Universidad imparta la carrera, que corresponden a Concepción y Santiago.

Docentes. Las conversaciones sostenidas con docentes revelan que la mayoría de ellos ha trabajado alguna con la metodología ABP, y su experiencia indica que los talleres generan un mayor provecho de conocimientos por parte de los estudiantes, ya que permite un mejor desarrollo de las competencias que se quieran lograr, aunque no recomiendan hacer un uso abusivo de ella, ya que en algunas oportunidades queda la sensación que la clase se tornara tediosa, y que el rol del profesor se simplifica a pasar por los puestos de trabajo sin que se produzca una interacción más fluida con los alumnos que les ayude a enriquecer sus procesos de aprendizaje. En cuanto al uso de software

[125] Se justifica la incorporación de la directora de plan común de Ingeniería, ya que la asignatura «*Tecnologías de la Información*» está incluida dentro de las responsabilidades de esta unidad.

educativo orientado a la enseñanza de la programación[126], ellos han empleado NetBeans, Power Designer, Dev C++, Pseint, Oracle, SQL Server, que han complementado con elementos como pizarra acrílica, diapositivas hechas con PowerPoint, guías de estudio, textos de apuntes elaborados por ellos mismos, y otros recursos bibliográficos adicionales que pudieran encontrar, ya sea en formato impreso o virtual que emplean como complemento de lo que revisen. Ninguno de ellos ha escuchado hablar de Alice o de BlueJ; sin embargo, cuando se les explicó en qué consisten, coincidieron en que la primera resultaría ser bastante atractiva para los alumnos, ya que permite aplicar los conceptos principales de la programación orientada a objetos por medio de la interacción con los elementos «*tangibles*» que formen parte de los mundos virtuales que los estudiantes podrán recrear. Por último, manifiestan que sería una buena idea emplear Alice y BlueJ — en ese orden — para introducir de mejor manera las nociones básicas de programación que debieran incorporar los estudiantes al cabo del primer año de estudio, pensando que ellos estarán trabajando bajo las directrices metodológicas del ABP durante las primeras cuatro unidades del curso.

Estudiantes. En relación a la incorporación de ABP como estrategia metodológica de enseñanza, los estudiantes han manifestado reparos con la idea de trabajar en grupo, ya que tienen el temor de no saber cómo enfrentar con éxito el problema que el profesor les plantee, pero se manifiestan abiertos a la posibilidad de descubrir por ellos mismos sus propias necesidades de aprendizaje, pero les embarga un sentimiento de incertidumbre al no saber cómo conectar esa información con el resto de sus compañeros. No se muestran de acuerdo con tener que exponer un trabajo, porque esa forma de evaluar termina ayudando a aquellas personas que menos aportaron a la tarea grupal, lo que desde esa perspectiva les parece injusto. Creen que sería una buena experiencia en la medida que el profesor se muestre involucrado con el

[126] La lista de programas educativos se refiere a los que emplean para enseñar programación estructurada y programación orientada a objetos. Ambos tipos se consideran válidos dentro de las respuestas que brindaron, ya que ellos apuntan al uso de estas herramientas para ayudar a desarrollar la capacidad de razonamiento algorítmico y sistémico.

desempeño de cada uno de los estudiantes, por medio de retiradas instancias de retroalimentación. Por otra parte, ellos corroboran con su relato que NetBeans es una herramienta que puede ser usada para enseñar a programar, pero que a lo mejor no es la más apropiada para principiantes, ya que ofrece opciones que al alumno le cuesta manejar, como por ejemplo: opciones de depuración, manejador de proyectos, creación de clases empaquetadas, etc. Consideran que Alice y BlueJ serían unas buenas alternativas de trabajo, ya que éstas permiten separar la sintaxis de las instrucciones que se quieran codificar, haciendo que el aprendizaje en una etapa inicial resulte ser mucho más intuitivo que con otras herramientas de corte más tradicionalistas como NetBeans o Eclipse.

6.2.2 Cronograma de trabajo

Los hitos más relevantes que se han considerado para la puesta en ejecución de lo propuesto son los que se enumeran a continuación, y cuyo desglose temporal se expone en la Tabla 7:

1. Diseño de la propuesta de innovación e intervención pedagógica.
2. Elaboración de los documentos de planificación de asignatura que determina la Universidad San Sebastián.
3. Construcción del material de apoyo para el curso, que incluye: texto de estudio para las unidades de aprendizaje, configuración de la plataforma Moodle, creación de una cuenta en YouTube para difundir los trabajos finales.
4. Elaboración de las experiencias de aprendizaje basado en problemas.
5. Validación teórica de la propuesta: que incluye reuniones y entrevistas con los estamentos universitarios a los cuales va dirigido el proyecto: director de carrera, directora de plan común de Ingeniería, docentes y estudiantes.
6. Validación empírica de la propuesta: recoger sugerencias de expertos que puedan hacer observación metodológica de aula, con el fin de hacer los ajustes necesarios que permitan fortalecer aquellos aspectos donde el

proyecto se vea más deficitario, que incluya además entrevistas con docentes y estudiantes.

Por razones de legibilidad, el cronograma de trabajo se muestra en la página siguiente.

Tabla 7. Cronograma de trabajo para la validación del proyecto.

	2012					2013				
Mes / Hito	Agosto	Septiembre	Octubre	Noviembre	Diciembre	Enero	Marzo	Abril	Mayo	Junio
Diseño de la propuesta.										
Planificación de la asignatura.										
Construcción del material de apoyo del curso.										
Elaboración de las actividades ABP.										
Validación teórica de la propuesta.										
Validación empírica de la propuesta.										

Reflexión final del capítulo

La propuesta que se ha formulado es viable, ya que desde todos los puntos de vista analizados — institucional, operacional y financiero — no se registraron barreras que pudieran poner en jaque su implementación en la Universidad San Sebastián, tanto en lo institucional, operacional como en el ámbito financiero.

Con respecto a la validación teórica que brindan los usuarios, se puede comentar que los profesores tienen mayor claridad sobre los alcances que tiene la aplicación de una metodología como el ABP al interior de la sala de clase, dada la mayor experiencia que tienen con respecto a los alumnos consultados, y que evalúan de manera positiva la inclusión de Alice y de ABP como recursos didácticos ordenados de esa manera, de acuerdo a todo lo que se ha explicado hasta el momento, lo que permite concluir que en teoría constituye una secuencia lógica para abordar de manera exitosa una introducción al aprendizaje de la programación orientada a objetos.[127] Sin embargo, a pesar de los buenos comentarios de los profesores, es natural que aquellos que aún no hayan trabajado con ABP tengan ciertos reparos que tal vez no encuentren o no existan en el esquema de enseñanza tradicional. De acuerdo lo que comenta el académico de la Universidad Politécnica de Cataluña Miguel Valero-García[128], bajo esta metodología se experimenta con mayor frecuencia con los errores y trabajos mal hechos que desarrollen los alumnos, pero que es absolutamente necesario para que se produzca el aprendizaje en ellos. Además, remarca la importancia del hecho que el profesor monitoree en todo momento el desempeño de los estudiantes, para apoyarlos y orientarlos de la mejor manera en pro del éxito de la experiencia.

Los estudiantes, han manifestado reparos que son entendibles desde su punto de vista, ya que es común que tengan ciertas reticencias que son propias

[127] Se dice que es en teoría solamente, ya que ninguno de los profesores consultados ha trabajado alguna vez con los recursos didácticos propuestos en este proyecto de intervención pedagógica, y por lo tanto no pueden elaborar un discurso que se sustente en alguna evidencia empírica al respecto.

[128] Valero-García, M. (2007). *Las dificultades que tienes cuando haces PBL*. Cataluña: Universidad Politécnica de Cataluña.

del temor al cambio. De acuerdo a Valero-García, el alumno puede percibir que este proceso como algo traumático, sobre todo si ha estado acostumbrado a los patrones tradicionales de enseñanza, donde lo natural es que transite por estas fases, que culminan por lo general con una sensación de satisfacción por los resultados obtenidos:[129]

Shock. Es la reacción inicial negativa que tiene el estudiante cuando se entera que tendrá que abordar un problema en un grupo de trabajo sin que el profesor le haya brindado alguna explicación previa de la teoría.

Negación. Es una etapa en la cual el alumno no puede creer que el profesor sea capaz de mantener firme la convicción de trabajar bajo esta modalidad. En ese momento, el estudiante mantiene la esperanza de volver a lo tradicional.

Emoción fuerte. Es la sensación de experimentar que el problema pueda escapar a su control, y que evalúe con seriedad la posibilidad de abandonar la asignatura, y tomarla al año siguiente. Lo normal según Valero-García es que el alumno desista de esta actitud, y que pase a la siguiente fase.

Resistencia y abandono. Acá el alumno sigue trabajando en esta experiencia, pero lo hace a regañadientes, sin importarle si debido a su mala disposición pueda pasar a perjudicar al grupo.

Rendición y aceptación. Una vez superada la etapa de la resistencia, puede experimentar el deseo de otorgarle una nueva oportunidad a la experiencia de trabajo basada en ABP, considerando que a esas alturas del partido el profesor no va a variar el enfoque.

La lucha y la exploración. Es posible que el estudiante vea que otros compañeros como él han experimentado algún grado de avance. Es en ese momento cuando el profesor tiene que percibir a tiempo para tratar de convencer a los más reticentes que ellos también pueden lograr los resultados esperados.

[129] El autor hace una adaptación de un artículo publicado el año 1994 por el académico de la McMaster University D. R. Woods, titulado *Problem-based Learning: How to Gain the Most from PBL*.

El retorno de la confianza. Acá es cuando el alumno adopta un cambio de actitud, ya que pasa de lo reacio a una sensación de estar controlando la situación de aprendizaje.

Integración y éxito. En este momento, el estudiante se da cuenta que era capaz de llevar a cabo esta experiencia de manera eficaz.

Bibliografía

Abarca, A. (2002). *Notas sobre la Teoría de Autómatas Finitos, Expresiones Regulares y Gramática Formal.* El Salvador: Departamento de Electrónica e Informática, Universidad Centroamericana José Simeón Cañas.

Barnes, D., & Kölling, M. (2007). *Programación Orientada a Objetos: Una introducción práctica usando BlueJ.* Editorial Pearson Educación.

Brassard, G. (1997). *Fundamentos de Algoritmia.* Montreal: Editorial Prentice Hall.

Careaga, A. (2007). *El desafío de ser docente.* Montevideo: Departamento de Educación Médica, Facultad de Medicina, Universidad de la República.

Carralero, N. (2011). *Entornos para enseñar programación en secundaria. Nuevos enfoques.* Castilla La Mancha: Quaderns Digitals.

Cisterna, F. (2005). *Categorización y triangulación como procesos de validación del conocimiento en investigación cualitativa.* Chillán: Revista Theoria: Ciencia, Arte y Humanidades, Universidad del Bío-Bío.

Cisterna, F. (2007). *Currículum Educacional. Concepciones teóricas y desarrollo en el aula.* Chillán: Ediciones Universidad del Bío-Bío.

Cisterna, F. (2011). *Investigación Educacional: Paradigmas, Planteamiento Problemático, Marco Teórico, Procedimientos de Recolección y Análisis de la Información.* Chillán: Ediciones Universidad del Bío-Bío.

Cobos, J. A. (2005). Inteligencia Emocional: Necesidad insatisfecha en la Educación. *Revista Digital I+E: Investigación y Educación*.

Cooper, S., Dann, W., & Pausch, R. (2003). *Teaching Objects-first In Introductory Computer Science.* Pittsburgh: Carnegie Mellon University.

Dann, W., Cooper, S., & Pausch, R. (2012). *Learning to Program with Alice.* Pittsburgh: Editorial Pearson.

Denning, P. (1989). Computing as a discipline. *Communications of the ACM, 32*(12).

Dirección de Investigación y Desarrollo Educativo. (2000). *El Aprendizaje Basado en Problemas como técnica didáctica.* Monterrey: Vicerrectoría Académica, Instituto Tecnológico y de Estudios Superiores de Monterrey.

Formella, A. (2010). *Teoría de Autómatas y Lenguajes Formales.* Vigo: Universidad de Vigo.

Forouzan, B. (2003). *Introducción a la Ciencia de la Computación: de la manipulación de datos a la Teoría de la Computación.* Cupertino, California: Editorial Thompson.

García, J. (2000). *Aprenda Java como si estuviera en primero.* San Sebastián: Escuela Superior de Ingenieros Industriales, Universidad de San Sebastián.

García-Valcárcel, A. (2009). *Medios Informáticos.* Salamanca: Universidad de Salamanca.

Giraldo, J., & Mateus, S. (2010). *Aprendizaje de la Programación Orientada a Objetos a través de la creación de juegos de video.* Medellín: Cefalea.

Grimaldi, R. (1997). *Matemáticas Discreta y Combinatoria: Una introducción con aplicaciones.* Editorial Addison-Wesley Iberoamérica.

Guardián Rojo. (16 de Noviembre de 2009). *Software Educativos.* Recuperado el 10 de Diciembre de 2012, de Software Educativos: http://guardianrojo.blogspot.com/

Guerequeta, R. (2002). *Técnicas de diseño de algoritmos.* Málaga: Universidad de Málaga.

Institute for Human and Machine Cognition. (20 de Abril de 2000). *The Institute for Human and Machine Cognition web page.* Recuperado el 9 de Noviembre de 2012, de El software educativo: http://cursa.ihmc.us

Instituto Profesional Virginio Gómez. (2005). *Taller de formación en aprendizaje basado en problemas.* Concepción: IPVG.

Kölling, M. (2005). *El "tutorial" de BlueJ.* Dinamarca: Editado por The University of Southern Denmark.

Kouznetsova, S. (2007). *Using BlueJ and blackjack to teach object-oriented design concepts in CS1.* Huntsville, Texas: Sam Houston State University.

López, J. C. (2009). *Algoritmos y Programación: Guía para Docentes.* Quito: Fundación Gabriel Piedrahita Uribe.

Marquès, P. (10 de Enero de 1996). *El software educativo.* Recuperado el 9 de Noviembre de 2012, de El software educativo: http://www.lmi.ub.es/te/any96/marques_software/#capitol7

Moskal, B., Lurie, D., & Cooper, S. (2000). Evaluating the Effectiveness of a New Instructional Approach. *Conference 2000 ACM.*

Ramis, F., Bañados, C., & Muñoz, Á. (2007). *Aprendizaje basado en problemas en el contexto de resolución de problemas.* Concepción: Universidad del Bío-Bío.

Rojas, P. (2010). *El aprendizaje basado en problemas como estrategia metodológica de enseñanza y aprendizaje de la integral indefinida, en paralelo con derivadas y su incidencia en el rendimiento académico de los estudiantes de Ingeniería en Informática de INACAP Chillán.* Chillán: Tesis para optar al grado académico de Magíster en Enseñanza de las Ciencias, mención Matemática. Universidad del Bío-Bío.

Russell, S. J. (2004). *Inteligencia Artificial: Un enfoque moderno.* Madrid: Editorial Pearson Educación.

Salcedo, P. (2004). Un curso de lenguajes formales e introducción a la teoría de compiladores que usa una plataforma basada en el conocimiento para educación a distancia. *Revista de Ingeniería Informática*.

Sipser, M. (2005). *Introduction to the Theory of Computation.* Massachusetts: Editorial MIT Press.

Torres, S., & Liñan, E. (2012). Aprende programación con Alice. *CIENCIACIERTA*.

Valero-García, M. (2007). *Las dificultades que tienes cuando haces PBL.* Cataluña: Universidad Politécnica de Cataluña.

Van Haaster, K., & Hagan, D. (2004). *Teaching and Learning with BlueJ: an Evaluation of a Pedagogical Tool.* Melbourne: Monash University.

Vidal Ledo, M., Gómez Martínez, F., & Ruiz Piedra, A. (2010). Software educativos. *Scielo*, 97-110.

Villalobos, J. (2006). *El reto de diseñar un primer curso de programación de computadores.* Bogotá: Departamento de Ingeniería de Sistemas Bogotá, Universidad de los Andes.

Villarroel, S. (2002). *Proyecto Educativo Institucional: Marco legal y Estructura básica.* Santiago de Chile: Gobierno de Chile, Ministerio de Educación.

Zárate, H. (2008). *Paradigmas de Programación.* Ciudad de México: Universidad Nacional Autónoma de México, Facultad de Ingeniería.

Anexos

Anexo A

Vinculación de categorías con subcategorías, instrumentos y estamentos

Tabla 8. Tabla de especificaciones para todas las categorías apriorísticas.

Categoría	Subcategorías	Estamentos en que se investigará	Instrumentos a utilizar
A. Planificación de la enseñanza.	A.1 Planificación curricular.	Director de carrera.	Entrevista.
		Docente de aula.	Entrevista.
		[redacted]	Revisión documental.
	A.2 Planificación didáctica.	Director de carrera.	Entrevista.
		Docente de aula.	Entrevista.
		[redacted]	Revisión documental.
B. Desarrollo de la enseñanza.	B.1 Estrategias metodológicas.	Docente de aula.	Entrevista, Observación de aula.
		Estudiante.	Entrevista, Focus group.
	B.2 Interacción pedagógica.	Docente de aula.	Entrevista, Observación de aula.
		Estudiante.	Entrevista, Focus group.
	B.3 Recursos didácticos.	Docente de aula.	Entrevista, Observación de aula.
		Estudiante.	Entrevista, Focus group.
	B.4 Organización del tiempo y del espacio.	Docente de aula.	Entrevista, Observación de aula.
		Estudiante.	Entrevista, Focus Group.
	B.5 Clima de aula.	Docente de aula.	Entrevista, Observación de aula.
		Estudiante.	Entrevista, Focus Group.

Categoría	Subcategorías	Estamentos en que se investigará	Instrumentos a utilizar
C. Cultura institucional.	C.1 Cultura institucional en materias de tipo general.	Director de carrera.	Entrevista.
		[redacted]	Revisión documental.
	C.2 Cultura institucional en materias de tipo pedagógica.	Director de carrera.	Entrevista.
		[redacted]	Revisión documental.
D. Creencias pedagógicas personales.	D.1 Creencias pedagógicas de tipo personal en cuanto al rol de la educación en general.	Director de carrera.	Entrevista.
		Docente de aula.	Entrevista.
	D.2 Creencias personales en cuanto al rol específico del docente como agente pedagógico.	Director de carrera.	Entrevista.
		Docente de aula.	Entrevista.

Anexo B

Experiencias de Aprendizaje Basado en Problemas

B.1 Primera experiencia de ABP

Presentación

Título: «*Diseño de un videojuego interactivo usando el enfoque de programación orientada a objetos***»**.

Institución: Universidad San Sebastián.

Asignatura: Tecnologías de Información.

Duración: 13 de marzo al 8 de abril de 2013.

Destinatarios: Alumnos de la carrera de Ingeniería Civil Informática.

Semestre - Año: Primer Semestre de 2013.

Objetivo General.

Plantear el diseño de una solución algorítmica con Alice, empleando conceptos iniciales ligados al paradigma de la programación orientada a objetos.

Objetivos Específicos.

1. Aplicar el modelo de trabajo «*top-down*» para la descomposición de problemas en fragmentos que sean codificables en Alice.
2. Documentar un programa que haya sido escrito en Alice.
3. Crear instancias de clases.
4. Aplicar métodos asociados con las clases instanciados para que los objetos se comporten del modo esperado.

Contenidos.

1. Diseño de programas con Alice.
2. Clases, objetos, métodos, parámetros.
3. Manejo de eventos de programación.

Escenario

A partir de este año, la Universidad San Sebastián dio por inaugurada una a iniciativa denominada «*Emprendimiento Estudiantil USS*», que tiene como objetivo brindar apoyo financiero a las empresas formadas por estudiantes de nuestra casa de estudios, que se hayan ido forjando en determinadas asignaturas presentes a lo largo de sus carreras. Cada grupo podrá optar a fondos concursables dentro de la Universidad para poder materializar las ideas que los inspiran, donde la creatividad y el aporte innovador pasarán a ser los elementos que ayudarán a determinar quiénes serán los beneficiarios de dichos recursos.

Si bien es cierto que la asignatura Tecnologías de Información es de primer año de carrera y que aún es muy pronto para la conformación de empresas consolidadas de alumnos, el equipo que está a cargo de esta unidad ha decidido incorporarla en «*Emprendimiento Estudiantil USS*», como una forma de incentivar la creación de redes colaborativas entre estudiantes a partir del momento que se incorporen a la Universidad, de tal manera que sus proyectos vayan madurando a medida que avancen con sus respectivos programas de estudio.

Por el momento, el curso estará conformado por una serie de empresas en gestación que se orientarán al diseño e implementación de animaciones y videojuegos para un usuario en entornos de programación 3D que sigan las orientaciones del paradigma de la orientación a objetos. En esta primera etapa, su empresa deberá entregar el diseño de un videojuego interactivo que deberá ser presentado al comité evaluador de «*Emprendimiento Estudiantil USS*» para su revisión y aprobación. Los proyectos disponibles son:

Misioneros y caníbales. Tres misioneros y tres caníbales están en una de las orillas de río junto a un bote, en el que sólo caben una o dos personas. Hay que encontrar la manera de pasarlos al otro lado del río, teniendo cuidado que en ningún momento queden menos misioneros que caníbales en cualquiera de las orillas. Considere que el bote no se maneja sólo, y por lo tanto debe ser manejado

por cualquiera de estos personajes. Cada uno de ellos está en condiciones de maniobrarlo, por lo que eso no representa un impedimento.

El barquero, la cabra y el repollo. Un granjero quiere cruzar un río junto con un lobo, una oveja y un repollo. Disponen de una barca, pero, naturalmente, sólo el granjero puede remar, en la cual caben dos personajes como máximo. Debe tener en cuenta que el lobo no puede quedarse solo con la oveja, ni la oveja sola con el repollo, porque de lo contrario se pueden comer.

Ranas verdes y ranas amarillas. Suponga que existen 7 piedras en línea en un charco, cada una de las cuales sólo puede ser ocupada por un sapo. Inicialmente, en cada una de las 3 piedras de más a la izquierda hay un sapo verde mirando hacia la derecha. En cada una de las 3 piedras de más a la derecha hay un sapo amarillo mirando hacia la izquierda. En consecuencia, inicialmente sólo la piedra central está desocupada. Los sapos verdes sólo pueden avanzar hacia la derecha y los amarillos solamente hacia la izquierda. Cuando un sapo avanza, sólo lo puede hacer a una piedra libre adyacente, o puede saltar un único sapo y caer en una piedra libre adyacente. El problema consiste en que los sapos amarillos finalicen en las 3 piedras de más a la izquierda, y los sapos verdes en las 3 piedras de más a la derecha.

Jarros de agua. Tiene un grifo de agua y dos jarros vacíos con capacidades de 3 y 4 litros, respectivamente. Usted puede llenar los jarros, vaciar el contenido de uno al otro o dejarlos en el suelo. Tiene que obtener exactamente un litro en cualquiera de los jarros.

Adaptación de Guitar Hero. Este es un popular videojuego que consiste en presionar ciertas teclas que van a simular notas musicales que van a ir apareciendo por la pantalla, a medida que se reproduce de fondo alguna canción. La idea es que el jugador presione las teclas que correspondan en el momento exacto en que aparezcan cercanas al comando asociado con cada cuerda de la guitarra, que en esta oportunidad se simulará por medio del teclado. A medida que

avanza, va ganando una mayor cantidad de puntaje que le permitirá acceder a los siguientes niveles de dificultad del juego.

Condiciones de cumplimiento

Conformación de los grupos de trabajo.

Cada grupo lo integrarán cinco o seis personas, y a la clase siguiente deberán entregar en un sobre cerrado:

1. Nombre creativo para la empresa que desee conformar el grupo con sus integrantes, con la firma de cada uno de ellos.
2. Identificación del jefe de grupo.
3. Logo que va a distinguir a su grupo, que debe ser creado de forma autónoma por parte de su equipo, y que debe guardar relación con el rubro de su empresa.
4. Cuál va a ser el proyecto que se va a desarrollar durante el semestre.

Producto final.

El producto que se deberá entregar el próximo 8 de abril consiste de un informe, que deberá convertir a formato PDF, y subirlo a la plataforma Moodle de la asignatura. Además, el trabajo deberá ser congruente con estas indicaciones:

1. Portada.
   - Nombre y logo de la Universidad.
   - Facultad.
   - Carrera.
   - Nuevo título para el trabajo.
   - Nombre con logo del grupo.
   - Integrantes del grupo ordenados por apellido paterno de manera ascendente.
   - Docente.
   - Fecha de entrega a pie de página.

2. Índice.
    - Debe contener como máximo tres niveles de profundidad.
    - Tiene que ser generado de forma automática.
    - Debe evidenciar que existe una coherencia interna en cuanto a la descomposición de los temas y subtemas que se van a desarrollar en el trabajo.

3. Introducción.
    - Resumen de lo que se va a hacer durante el proyecto.
    - Objetivos del trabajo, incluyendo los del informe anterior.
    - Extensión máxima de una página.

4. Contenido.
    - Descripción del problema.
    - Elaboración del libreto textual.
    - Describir las clases y los métodos derivados de ellas que se van a utilizar.
    - Descripción de los eventos que darán lugar a su proyecto.
    - Maquetación del proyecto, a través de capturas de pantalla que harán las veces de prototipo.

5. Conclusión general.
    - Consecución de los objetivos del proyecto.
    - Síntesis de los problemas y logros obtenidos con el trabajo.
    - Extensión máxima de una página.

6. Bibliografía.
    - Debe contemplar al menos dos fuentes bibliográficas.
    - Las citas deben contemplar el formato APA.
    - Debe ser generada de forma automática.

7. Elementos a evaluar en cuanto a la presentación.
    - Hoja tamaño carta.
    - Letra Arial en tamaño 12 para el cuerpo principal, y 18, 16 para los títulos y subtítulos secundarios.
    - Interlineado igual a 1,5.
    - Texto justificado.
    - Coherencia en cuanto a la redacción de las ideas.
    - Tablas e imágenes debidamente incluidas y explicadas.
    - Ortografía.
    - Informe anillado.

Exposición.

Para la presentación del proyecto semestral, el grupo deberá tener en cuenta estos factores:

- El tiempo máximo para exponer será de 15 minutos.
- Habrá una comisión evaluadora integrada por los jefes de grupo y el docente.
- La exposición se debe hacer en PowerPoint o en Prezi.
    - Contiene un resumen de los temas involucrados en el informe.
    - En total, debe contener como máximo 10 diapositivas.
- Sólo expone un integrante del grupo.
- Al final se realizará una ronda de preguntas a cada miembro del equipo.

Ponderaciones.

La calificación final que se obtenga equivale a la nota de la primera prueba solemne, que es el 20% del promedio final de la asignatura. Se descompone en estos términos:

Tabla 9. Ponderaciones para el primer ABP

Elemento	Ponderación
Informe.	60%
Exposición.	
1. Co-evaluación por parte de los demás jefes de grupo.	10%
2. Profesor	20%
Promedio de Auto-evaluación grupal	10%
Total	100%

Hitos dentro del trabajo.

1. Miércoles 20 de marzo. Haber escrito el guión del juego que van a desarrollar.
2. Lunes 1 de abril. Identificar cuáles son las clases y métodos presentes en el videojuego.
3. Lunes 8 de abril. Entrega del primer informe, junto a la exposición del mismo. El informe se debe entregar impreso y anillado al momento de hacer la exposición, y que luego se debe subir a la plataforma Moodle del curso en formato PDF.

Inasistencias.

La evaluación final del próximo 8 de abril es de carácter obligatoria. En caso de inasistencia, cada estudiante será calificado con nota 1,0, y deberá rendir a final de semestre la prueba solemne recuperativa.

Planificación de la actividad

La Tabla 10 muestra en detalle el cronograma de trabajo que se va a desarrollar a lo largo de este proceso, en el cual se incluye:

- Fecha de la sesión actual.
- Las acciones que se van a desarrollar al interior del aula.
- Descripción detallada de dichas acciones.
- El tiempo estimado de duración de cada una de ellas.
- Los productos que se tendrán que generar como consecuencia de las acciones antes mencionadas.
- Materiales o recursos didácticos que se van a utilizar.
- Actividades que los estudiantes en sus grupos de trabajo deberán realizar fuera del aula.

Tabla 10. Planificación de la primera actividad ABP

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
Miércoles 13/03/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Explicación general del profesor en torno a las actividades que habrán de realizarse. Además el profesor indica algunas técnicas de trabajo que puedan emplear los estudiantes para optimizar su trabajo dentro de la clase, como por ejemplo: guiones de aprendizaje colaborativo, estructura simple, co-op co-op, investigación en grupo, o rompecabezas.	Apunte del profesor. Moodle. Proyector. Computador. Alice. Pizarrón.	Confección del material que cada grupo debe entregar a la clase siguiente en un sobre cerrado.
	Repaso sobre generalidades del entorno Alice.	Actividad de síntesis donde se comentan los elementos principales que caracterizan a Alice.	10’			
	Entrega de las bases del trabajo ABP.	Cada grupo se forma por afinidad de caracteres, como lo han hecho hasta el momento. Los alumnos, en sus grupos de trabajo leen la especificación del problema y lo discuten entre ellos.	20’	Lectura comprensiva por parte de los estudiantes, para establecer un consenso sobre la manera en cómo se percibe el escenario de trabajo.		

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Entrega de las rúbricas para la confección del informe final y para la presentación.	El profesor le entrega a cada alumno las rúbricas que se emplearán para calificar el informe y la presentación del lunes 8 de abril. Además, aclara las dudas que pudieran surgir a partir de cada uno de los ítems considerados en estos documentos.	15’			
	Formulación de preguntas.	Los estudiantes elaboran preguntas en relación al planteamiento del problema. El profesor explica la importancia que conlleva el desarrollo del pensamiento algorítmico, y de cómo esto se puede transportar al ámbito de la resolución de la situación que se plantea.	20’	Los alumnos elaboran una lista con interrogantes que surgen a partir de la lectura del problema planteado. Los alumnos discuten en grupo cuáles son los objetivos de aprendizaje que deberán cubrir a partir de lo que el profesor les ha planteado.		
	Entrega del apunte que trata sobre diseño de programa con Alice (Anexo C, páginas 1 a la 48).	El profesor habilita en la plataforma Moodle el apunte sobre diseño de programas con Alice.	5’	Apunte publicado en la plataforma Moodle del curso. Los alumnos elaboran preguntas adicionales en torno a lo que vayan		

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
				a requerir aprender, con el fin de dar una mejor solución al problema planteado, de tal manera que puedan establecer un diagnóstico inicial de la situación.		
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Viernes 15/03/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Identificación de los grupos de trabajo. Identificación del proyecto asociado a cada uno de ellos.	Apunte del profesor. Moodle. Proyector. Computador. Alice. Pizarrón.	Elaboración del bosquejo sobre el proyecto que cada grupo realizará. Investigación autónoma sobre cómo se crea un informe y una presentación. Elaboración una descripción del problema, en términos de lo que tengan que resolver, de acuerdo al análisis parcial que los alumnos lleven hasta el
	Entrega del sobre con los datos del grupo.	Cada grupo le entrega al profesor el sobre que contiene el nombre de la empresa que quieren formar, junto a los integrantes, jefe, logo que los caracterizará y el proyecto que desarrollarán.	10’			

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
						momento.
	Taller grupal de diseño de programas con Alice.	Los alumnos trabajan en sus respectivos grupos los conceptos y ejemplos planteados en el apunte sobre diseño de programas en Alice.	60’	Lectura comprensiva en grupo. Identificación de las necesidades de aprendizaje.		
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’	Retroalimentación del profesor en torno a las necesidades de aprendizaje planteadas por los estudiantes.		
Lunes 18/03/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Explicación general del profesor en torno a las actividades que habrán de realizarse.	Apunte del profesor. Moodle. Proyector. Computador.	Los alumnos preparan un plan con posibles acciones para cubrir las necesidades de aprendizaje que han identificado, que incorpore las

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Taller grupal de diseño de programas con Alice.	Los alumnos continúan trabajando en sus respectivos grupos los conceptos y ejemplos planteados en el apunte sobre diseño de programas en Alice.	15’		Alice. Pizarrón.	recomendaciones dadas por el docente, por medio de un esquema de trabajo que por el momento les permita determina cuáles son las clases, objetos y métodos que se deberán emplear para poner en marcha el flujo de acción del videojuego.
	Entrega del apunte sobre clases, métodos y parámetros (anexo C, páginas 50 a la 101).	El profesor habilita en la plataforma Moodle el apunte sobre diseño de programas con Alice.	5’	Lectura comprensiva en grupo, de tal manera que ellos analizan la información entregada, buscan información en fuentes adicionales para complementar lo entregado, con el fin de replantear si es necesario disponer de datos adicionales. Los alumnos asimilan los conceptos de clase, objeto y método. Los alumnos aplican lo leído y lo contextualizan a su ámbito de trabajo.		
	Taller grupal sobre instanciación de clases en Alice.	Los alumnos trabajan en sus respectivos grupos los conceptos y ejemplos relacionados con la instanciación de clases en Alice, y de la colocación de objetos en el mundo principal.	50’			
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Miércoles	Introducción de las	El profesor brinda una	5’	Observación a cada	Apunte del profesor.	Mejoramiento de

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
20/03/2013	actividades a desarrollar en la clase.	síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.		grupo, donde se genere una instancia de retroalimentación que sirva de estímulo para la mejora de las actividades que los alumnos hayan estado emprendiendo hasta el momento. La idea es que esta instancia de reflexión sea entre los miembros del grupo, y entre el profesor con el grupo, para aunar ideas y criterios.	Moodle. Proyector. Computador. Alice. Pizarrón.	bosquejo general de trabajo por medio de la incorporación de las sugerencias dadas por el profesor. Incorporación de clases, objetos, métodos junto con los arreglos correspondientes.
	Revisión del guión de trabajo de cada grupo.	El profesor hace una revisión del estado de avance de la elaboración de los libretos de trabajo de cada grupo, y les brinda retroalimentación para que integren lo que han hecho con las actividades anteriores.	25’			
	Diseño y maquetación de programas con Alice.	Los alumnos integran el trabajo de las sesiones pasadas, y determinan cómo generar prototipos de sus proyectos.	45’	Los alumnos elaboran mejoras frente a su desempeño, y luego avanzan con la maquetación del programa, junto con una retroalimentación que reciban por parte del docente.		
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Viernes 22/03/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la	5’	Los alumnos generan prototipos de sus proyectos, lo discuten entre ellos y luego lo	Apunte del profesor. Moodle.	Mejoramiento de bosquejo general de trabajo por medio de la incorporación de las

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
		clase en sus grupos de trabajo.		conversan con el profesor, generándose una nueva instancia de retroalimentación entre ambas partes.	Proyector. Computador. Alice. Pizarrón.	sugerencias dadas por el profesor.
	Diseño y maquetación de programas con Alice.	Los alumnos integran el trabajo de las sesiones pasadas, y determinan cómo generar prototipos de sus proyectos.	55’			
	Revisión del guión de trabajo a cada grupo.	En caso de ser necesario, continúa la revisión de los avances parciales.	15’			
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Lunes 25/03/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Lectura comprensiva usando el método Co-op Co-op, como una forma de lograr que los estudiantes asimilen más rápido el	Apunte del profesor. Moodle. Proyector.	Identificación de los eventos que deberán considerar para que el juego funciones como se espera.

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Entrega del apunte que trata sobre eventos en Alice (anexo C, páginas 103 a la 131).	El profesor habilita en la plataforma Moodle el material didáctico que trata sobre eventos en Alice.	15’	concepto de evento. Los alumnos asimilan el concepto de eventos a partir de la lectura realizada, por medio de un análisis de la información proporcionada que les permita determinar cómo esta idea les puede ser útil para abordar de mejor manera el problema propuesto.	Computador. Alice. Pizarrón.	
	Taller grupal sobre creación de eventos en Alice.	Los alumnos trabajan en sus respectivos grupos los conceptos y ejemplos relacionados con la creación de eventos en Alice.	55’			
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Miércoles 27/03/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Los alumnos desarrollan problemas relacionados con creación de eventos. Retroalimentación del profesor, por medio de observaciones que conduzcan a una mejora del desempeño exhibido hasta el momento.	Apunte del profesor. Moodle. Proyector. Computador. Alice. Pizarrón.	Mejoras al libreto, clases, métodos, objetos, parámetros y eventos que el grupo haya considerado, de acuerdo a la retroalimentación del profesor.
	Taller grupal sobre creación de eventos en Alice.	Los alumnos continúan el trabajo del material que trata sobre eventos en Alice.	70’			

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Lunes 01/04/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Los alumnos presentan las mejoras que hayan hecho al profesor, y éste los retroalimenta. Los estudiantes incorporan las mejoras sugeridas por el profesor, y las integran al informe que están elaborando.	Apunte del profesor. Moodle. Proyector. Computador. Alice. Pizarrón.	Los alumnos preparan una versión preliminar de lo que será la entrega final del trabajo del próximo 8 de abril, donde cada integrante del grupo deberá hacerse presente, a objeto que todos sean capaces de responder las preguntas que se formulen el día de la presentación.
	Revisión del estado de avance en cuanto a la creación de clases, objetos y parámetros a cada grupo.	El profesor brinda una retroalimentación a cada grupo de trabajo, en función del desempeño exhibido hasta el momento.	25’			
	Diseño, maquetación y determinación de eventos.	Los alumnos en sus grupos de trabajo integran todo lo que han visto hasta el momento, y lo plasman en sus proyectos.	45’			
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima	5’			

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
		sesión.				
Miércoles 03/04/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Los grupos generan las versiones preliminares de sus informes. El profesor brinda retroalimentación de lo que se ha obrado hasta el instante, y genera algunas recomendaciones para optimizar el producto.	Apunte del profesor. Moodle. Proyector. Computador. Alice. Pizarrón.	Los alumnos analizan todo lo que llevan construido hasta el momento, e incorporan las observaciones que les haya aportado el profesor.
	Diseño, maquetación y determinación de eventos.	Los alumnos en sus grupos de trabajo integran todo lo que han visto hasta el momento, y lo plasman en sus proyectos. Retroalimentación general por parte del profesor.	70’			
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Viernes 05/04/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Entrega de la evaluación sumativa aplicada a cada grupo para determinar el grado de avance, junto con comentarios de	Apunte del profesor. Moodle. Proyector.	Elaboración del informe final y de la presentación final.

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Síntesis del trabajo realizado.	El profesor brinda una síntesis del proceso que han estado desarrollando los alumnos, y brinda antecedentes que ayuden a enriquecer el contenido del informe.	35’	carácter general que sean útiles para todo el curso.	Computador. Alice. Pizarrón.	
	Entrega de los resultados de las evaluaciones a cada grupo.	El profesor le entrega y comenta con cada grupo los resultados particulares que obtuvieron de las observaciones realizadas hasta el momento.	35’			
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Lunes 08/04/2013	Exposición de grupos.	Cada grupo entrega y expone su trabajo.	80’	Informe y exposición de cada grupo.	Moodle. Proyector. Computador. Alice. Pizarrón.	

Evaluación

Informe

Tabla 11. Rúbrica para evaluar el primer informe

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Portada	Incluye todos los elementos solicitados en la descripción del trabajo de manera clara y ordenada.	Incluye todos los elementos solicitados, pero los nombres de los integrantes del grupo no vienen ordenados por apellido paterno en forma ascendente, o bien la portada abarca entre cinco y siete de los elementos contemplados, con algunos errores moderados de formato.	La cantidad de elementos incorporados incluye entre dos y cuatro del total de ítems totales, con algunos errores moderados de formato.	No incluye portada, o bien se consideró sólo uno de los ocho elementos considerados, o la información global contenida en la portada no es suficiente.
	10 puntos.	**6 puntos.**	**4 puntos.**	**0 punto.**
Índice	Incluye índice completo, con un máximo de tres niveles de profundidad, generado de manera automática, y con una adecuada descomposición de los temas.	El índice ha sido construido de manera automática, pero no contiene un desglose apropiado de los temas solicitados en el cuerpo principal del informe.	El informe está generado de forma manual, con algunos errores en la numeración de las páginas, y sin que se aprecie una secuencia lógica en el orden de los puntos solicitados.	El índice no está presente en el informe, o bien ha sido realizado en forma manual con errores en casi todos los números de páginas, sin que se aprecie una secuencia lógica en el orden de los puntos solicitados.
	5 puntos.	**3 puntos.**	**1 punto.**	**0 punto.**

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Introducción	La introducción incluye el resumen del trabajo, junto con los objetivos a alcanzar de una manera clara y ordenada.	Incluye resumen y objetivos, con algunos errores moderados que no afectan la legibilidad ni la coherencia interna de lo expuesto.	El resumen del trabajo no considera todos los aspectos involucrados, y los objetivos no son coherentes con lo que se solicita.	No está incluida en el informe, o bien no es clara en cuanto al planteamiento de las ideas, y no especifica los objetivos del trabajo.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**
Contenido / Descripción del Problema	La descripción del problema es completa, y no deja lugar a ambigüedades, ya que es congruente con lo enunciado en la introducción.	La descripción del problema excluye algunos elementos menores que no alteran de manera significativa la formulación del mismo.	La descripción del problema presenta ciertas lagunas argumentativas que alteran de manera notoria su conformación interna.	No está presente, o bien es completamente diferente con lo descrito en la introducción.
	10 puntos.	**6 puntos.**	**3 puntos.**	**0 punto.**
Contenido / Elaboración del libreto	El libreto o algoritmo general del proyecto mantiene una correspondencia clara con lo enunciado en la descripción del mismo.	El libreto del proyecto presenta algunos errores menores que se pueden superar rápidamente.	El libreto presenta serias incoherencias con respecto a lo formulado en la descripción del proyecto de trabajo.	El libreto no está presente, o bien evidencia graves incongruencias con respecto a la descripción dada, al no contener todos los elementos descritos en el punto anterior.
	25 puntos.	**20 puntos.**	**10 puntos.**	**0 punto.**

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Contenido / Clases y métodos	Las clases y métodos son pertinentes y suficientes para la descripción del proyecto.	Tanto las clases como los métodos aparecen descritos y especificados, con errores moderados.	Los métodos que se han definido no son suficientes para describir el comportamiento de las clases, pero al menos no se observa una carencia de éstas.	No hay clases ni métodos definidos, o bien lo que se ha elaborado no alcanza a descomponer de manera mínima la especificación del proyecto, ya que faltan clases importantes.
	20 puntos.	**15 puntos.**	**10 puntos.**	**0 puntos.**
Contenido / Descripción de los eventos	Los eventos que el grupo ha considerado permiten identificar de manera clara y suficiente la manera en cómo se dará el curso de las acciones y de las interacciones propias del proyecto.	Los eventos de programación que se han considerado permiten describir claramente la mayor parte del flujo de acciones del proyecto, con algunas omisiones menores.	Los eventos que se han definido dan cuenta de una cantidad significativa de omisiones de continuidad de las acciones del producto final, que alteran en demasía la esencia del proyecto.	No existe una descripción de los eventos, o bien, lo que se ha definido no permite en absoluto mantener una secuencia clara y ordenada del curso de acciones del producto que se va a implementar.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Contenido / Maquetación	Los prototipos que se muestran en el informe expresan con claridad todos los posibles escenarios que se puedan dar durante el desarrollo del programa.	Los prototipos que se muestran presentan errores moderados en cuanto a los diferentes escenarios del proyecto.	La mayoría de los prototipos evidencian grandes problemas de diseño e incongruencia con respecto al planteamiento del proyecto, y que en cuanto a cantidad alcancen un	No se muestra una maquetación del trabajo, o bien que los prototipos exhibidos son insuficientes y que en su mayoría evidencian graves incongruencias
			número significativo.	con el planteamiento del proyecto formulado al inicio de esta parte.
	10 puntos.	**6 puntos.**	**3 puntos.**	**0 punto.**
Conclusión	La conclusión del trabajo abarca de manera completa cada uno de los puntos solicitados en el enunciado del trabajo.	La conclusión presenta algunas incongruencias leves con respecto a los objetivos del proyecto, y hace un buen tratamiento de los logros y dificultades encontradas.	La conclusión no abarca a todos los objetivos del trabajo, ya que se limita a unos cuantos, y no hay una reflexión grupal en torno a logros y dificultades encontradas.	La conclusión no existe, o bien no contempla objetivos o alguna reflexión sobre logros y dificultades.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Bibliografía	La bibliografía es pertinente al desarrollo del trabajo, el sistema de citas respeta el estándar APA, y fue generada de forma automática.	La bibliografía presenta algunas omisiones leves, pero las citas siguen el estándar APA, y fue generada de forma automática.	La bibliografía no es completa, el estándar APA no fue empleado de manera correcta.	La bibliografía no existe, o bien fue hecha de forma manual, las fuentes no están debidamente documentadas y no hay presencia de citas en formato APA.
	5 puntos.	**4 puntos.**	**2 puntos.**	**0 punto.**
Presentación	El informe cumple a cabalidad con cada uno de los aspectos formales solicitados.	El informe presenta algunas faltas moderadas de ortografía y/o de redacción, y preserva una estructura tipográfica uniforme a lo largo de la escritura del mismo.	El informe presenta reiteradas faltas de ortografía y/o de redacción, y en algunos pasajes se aprecia que la estructura tipográfica que se solicita no se respeta.	El informe no presenta una cohesión interna con respecto a la redacción de las ideas, y en varios pasajes se aprecian severas faltas de ortografía que dificultan su lectura.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**

Exposición

Tabla 12. Rúbrica para evaluar la primera exposición[130]

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Contenido	Demuestra un completo entendimiento del tema, junto a una presentación que los contempla a todos.	Demuestra un buen entendimiento del tema, con algunos errores moderados.	Demuestra un dominio de una escaza cantidad de los temas abordados.	No pareciera tener un dominio de lo que está hablando. Se nota inseguridad a lo largo de toda la presentación.
	10 puntos.	**6 puntos.**	**4 puntos.**	**0 punto.**
Despierta interés por el proyecto.	El planteamiento del proyecto se formula de manera interesante, ya que genera debate y preguntas de la audiencia.	Quien expone contribuye a la generación de preguntas, pero no despierta mayor interés por iniciar algún debate en torno a lo presentado.	La persona que expone no despierta mayor interés entre sus compañeros, con la salvedad de algunas tibias preguntas.	El tema ha sido mal planteado desde el principio, y no genera preguntas ni debates.
	10 puntos.	**6 puntos.**	**4 puntos.**	**0 punto.**

[130] Válida tanto para el profesor como para los jefes de grupo que forman parte de la comisión.

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Respuestas grupales.	Todos los miembros del grupo responden de manera asertiva a las preguntas formuladas por el profesor.	Las respuestas dadas por los miembros del grupo presentan algunos errores moderados.	Las respuestas dadas por los integrantes del grupo presentan severas inconsistencias.	La respuesta de algunos miembros del grupo evidencia que no hubo una participación activa dentro del desarrollo del proyecto, que se hizo notoria debido a la inseguridad mostrada al hablar.
	20 puntos.	**15 puntos.**	**5 puntos.**	**0 punto.**
Presentación en Prezi o en PowerPoint	Buen tamaño letra, buen contraste con el fondo de las diapositivas, diseño de pantalla no recargado, buena selección gráficos, fotos y/o figuras	El tamaño y contraste de las letras no es lo óptimo, demasiada escritura, imágenes adecuadas pero no suficientes, sobre todo en cuanto a la maquetación.	Tamaño de letras muy reducido, escaza cantidad de figuras.	Demasiada escritura, casi nada de síntesis, figuras poco apropiadas que configuran una presentación poco atractiva para la audiencia.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**
Pronunciación	Habla claramente y distintivamente todo el tiempo, y no tiene mala pronunciación.	Se expresa de manera clara en casi todos los aspectos, con algunos errores moderados de pronunciación.	Más de la mitad del tiempo habla bien, pero sin destacar.	A menudo habla entre dientes, o no se entiende lo que dice la mayor parte del tiempo.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Vocabulario	Emplea un vocabulario apropiado para la audiencia, y cuando es necesario explica con claridad todos los términos que se empleen.	Emplea un vocabulario apropiado para dirigirse a la audiencia, pero no define los términos nuevos que vayan surgiendo.	Usa un vocabulario que es limitado para la audiencia, y no hace explicación de los términos propios que emerjan del proyecto.	Usa palabras u oraciones que no son entendidas por la audiencia, el lenguaje empleado es bastante pobre, y tiende a repetir en demasía las mismas palabras.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**
Volumen de voz	El volumen es lo suficientemente alto para ser escuchado por todos los miembros de la audiencia a través de toda la presentación.	El volumen es lo suficientemente alto para ser escuchado por todos los miembros de la audiencia al menos el 90% del tiempo.	El volumen es lo suficientemente alto para ser escuchado por todos los miembros de la audiencia al menos el 60% del tiempo.	El volumen con frecuencia es muy débil para ser escuchado por todos los miembros de la audiencia.
	10 puntos.	**6 puntos.**	**3 puntos.**	**0 punto.**
Tiempo	El tiempo de exposición varía entre 10 y 15 minutos.	La presentación dura entre 8 y 10 minutos.	La presentación dura entre 5 y 7 minutos.	La presentación dura menos de 5 minutos, o más de 15.
	10 puntos.	**6 puntos.**	**3 puntos.**	**0 punto.**

Auto-evaluación

Para cada una de las aseveraciones que se indican a continuación, marque con una X aquella que mejor refleje lo que usted ha trabajado por su grupo.

Tabla 13. Primera autoevaluación

Aspecto	Totalmente de acuerdo (4 puntos)	De acuerdo (3 puntos)	En desacuerdo (2 puntos)	Totalmente en desacuerdo (0 punto)
Me siento cómodo en el grupo.				
He tomado con responsabilidad las tareas encomendadas por el grupo.				
Me siento comprometido con el trabajo que estoy realizando por el grupo.				
Creo haber aportado como debiera para mejorar el rendimiento del grupo.				
Creo que mis aportes son tomados en cuenta por los demás integrantes del grupo.				
Aporto ideas que contribuyen a la toma de decisiones importantes dentro del grupo.				
Llego a tiempo para las reuniones de trabajo del grupo.				
Cumplí con los plazos de entrega de cada tarea que me fue encomendada.				
Mantengo buenas relaciones con los demás integrantes del grupo.				
He logrado dar a conocer mis planteamientos con claridad.				
He contribuido a que otros compañeros también participen.				

B.2 Segunda experiencia ABP

Presentación

Título: «*Implementación de un videojuego interactivo*»

Institución: Universidad San Sebastián.

Asignatura: Tecnologías de Información.

Duración: 10 de abril al 20 de mayo de 2013.

Destinatarios: Alumnos de la carrera de Ingeniería Civil Informática.

Semestre - Año: Primer Semestre de 2013.

Objetivo General.

Implementar en Alice el proyecto de trabajo que se formuló en la primera experiencia ABP, de acuerdo a las directrices del paradigma de la programación orientada a objetos.

Objetivos Específicos.

1. Refinar el libreto de trabajo, para que el diseño dé cuenta de lo métodos, funciones, estructuras condicionales, bucles, listas y arreglos que se desprendan de la formulación del proyecto.
2. Implementar todo lo que se haya documentado en el objetivo específico anterior.

Contenidos.

1. Funciones.
2. Estructura condicional.
3. Bucles finitos y bucles no acotados.
4. Listas, arreglos.

Escenario

El equipo que está a cargo de la recepción de los proyectos ha quedado conforme con lo que su empresa ha propuesto. Por tratarse de la primera vez que se hace un trabajo de esta naturaleza, es evidente que van a haber algunas acotaciones que su empresa deberá subsanar para que su propuesta culmine de manera exitosa.

Por lo tanto, para que su proyecto sea aprobado y cuente con el beneplácito de la comisión evaluadora, su empresa deberá resolver todas las observaciones que el profesor les haya indicado. Y una vez que todo haya culminado, llega el momento de implementar todo lo que usted ha planificado a lo largo de la primera experiencia, considerando los principales bloques de programación que intervienen en cualquier aplicación computacional.

El comité de «*Emprendimiento Estudiantil USS*» ha indicado que si su empresa hiciera un «*tráiler*» — o resumen audiovisual — del videojuego, el proyecto quedaría mucho mejor construido en términos de la publicidad que se pueda derivar de éste. Como una forma de incentivar su trabajo, los dos mejores programas serán premiados con cinco décimas en la calificación que cada integrante de cada grupo obtenga al cabo de la segunda evaluación.

Condiciones de cumplimiento

Conformación de los grupos de trabajo.

Los grupos de trabajo continúan tal como estaban establecidos en la primera experiencia ABP.

Producto final.

El producto que se deberá entregar el próximo 20 de mayo se desglosa en un informe y un programa. El informe deberá convertirlo a formato PDF, y subirlo a la plataforma Moodle del curso. Ambos elementos deberán seguir estas indicaciones:

Informe.

1. Portada.
   - Nombre y logo de la Universidad.
   - Facultad.
   - Carrera.
   - Título original para el trabajo.
   - Nombre con logo del grupo.
   - Integrantes del grupo ordenados por apellido paterno de manera ascendente.
   - Docente.
   - Fecha de entrega a pie de página.

2. Índice.
   - Debe contener como máximo tres niveles de profundidad.
   - Tiene que ser generado de forma automática.
   - Debe evidenciar que existe una coherencia interna en cuanto a la descomposición de los temas y subtemas que se van a desarrollar en el trabajo.

3. Introducción.
   - Resumen de lo que se va a hacer durante el proyecto.
   - Objetivos del trabajo.
   - Extensión máxima de una página.

4. Contenido.
   - Descripción del problema.
   - Elaboración del libreto textual.
   - Maquetación del proyecto, a través de capturas de pantalla que harán las veces de prototipo.
   - Describir las clases y los métodos derivados de ellas que se van a utilizar. Cada método se debe acompañar de una explicación y del código fuente comentado en Alice. La explicación debe incluir:
     - Utilidad dentro del proyecto.
     - Estructuras condicionales empleadas.

- Bucles «*for*» y bucles «*while*». No es necesario que ambos vayan, ya que esto va a depender de la naturaleza de su proyecto.
- Recursiones que haya tenido que emplear.
- Cuáles son las listas y/o arreglos que haya tenido que usar.

- Explicación de cada una de las funciones que intervienen en el programa, junto al código fuente comentado de cada una de ellas. La explicación debe incluir:
  - Utilidad dentro del proyecto.
  - Estructuras condicionales empleadas.
  - Bucles «*for*» y bucles «*while*». No es necesario que ambos vayan, ya que esto va a depender de la naturaleza de su proyecto.
  - Recursiones que haya tenido que emplear.
  - Cuáles son las listas y/o arreglos que haya tenido que usar.
- Descripción de los eventos que darán lugar a su proyecto, junto a la forma en que se vayan a implementar. Cada evento debe incluir el código fuente documentado.

5. Conclusión general.
   - Consecución de los objetivos del proyecto: procure mantener consistencia con lo que ha declarado en la introducción del trabajo.
   - Síntesis de los problemas y logros obtenidos con el trabajo, a través de una reflexión grupal de todo lo que han sido estas dos experiencias de trabajo.
   - Extensión máxima de dos páginas.

6. Bibliografía.
   - Debe contemplar al menos dos fuentes bibliográficas.
   - Las citas deben contemplar el formato APA.
   - Debe ser generada de forma automática.

7. Elementos a evaluar en cuanto a la presentación.
   - Hoja tamaño carta.
   - Letra Arial en tamaño 12 para el cuerpo principal, y 18, 16 para los títulos y subtítulos secundarios.
   - Interlineado igual a 1,5.
   - Texto justificado.
   - Coherencia en cuanto a la redacción de las ideas.
   - Ortografía.
   - Informe anillado.

Aspectos a evaluar en cuanto al funcionamiento del programa.

1. Al abrir el proyecto, el programa debe informar al usuario cómo ocuparlo, a través de un instructivo sencillo.
2. El instructivo debe contener en alguna parte el logo de la empresa.
3. El funcionamiento debe ser consecuente con lo que ha declarado en el instructivo.
4. El desarrollo del juego tiene que ser congruente con lo que ha documentado en el informe.
5. En cuanto al «*tráiler*»:
    - Se debe entregar aparte.
    - La inclusión de alguna melodía de fondo es obligatoria.
    - La duración de este resumen debe coincidir con la longitud de la pista de audio que haya incluido en el fondo.
    - El video tiene que subirlo al canal que el curso tiene en YouTube.
6. El archivo «*a2w*» debe subirlo a la cuenta que el grupo tiene en la plataforma Moodle de la asignatura.

Exposición.

Para la presentación del proyecto semestral, el grupo deberá tener en cuenta estos factores:

- El tiempo máximo para exponer será de 15 minutos.
- Habrá una comisión evaluadora integrada por los jefes de grupo y el docente.
- La exposición se debe hacer en PowerPoint o en Prezi.
    - Contiene un resumen de los temas involucrados en el informe.
    - En total, debe contener como máximo 10 diapositivas.
- Se debe dar énfasis a la última parte del trabajo, dejando pocos minutos para recordar lo que se hizo la primera vez.
- Debe mostrar el juego en funcionamiento.
- Sólo expone un integrante del grupo.

- Al final se realizará una ronda de preguntas a cada miembro del equipo.

Ponderaciones.

La calificación final que se obtenga equivale a la nota de la segunda prueba solemne, que corresponde al 20% del promedio final de la asignatura. Se descompone en estos términos:

Tabla 14. Ponderaciones para la segunda experiencia ABP

Elemento	Ponderación
Informe.	35%
Programa.	35%
Exposición.	
3. Co-evaluación por parte de los demás jefes de grupo.	10%
4. Profesor	15%
Promedio de auto-evaluación grupal	5%
Total	100%

Hitos dentro del trabajo.

1. Miércoles 17 de abril. Cada grupo debe haber incorporado las modificaciones recomendadas por el profesor.

2. Lunes 29 de abril. Cada grupo ha definido todos los métodos y funciones que se van a desarrollar en el proyecto.

3. Lunes 20 de mayo. Entrega del segundo informe, junto a la exposición del mismo. El informe se debe entregar impreso y anillado al momento de hacer la exposición, y luego se debe subir el archivo en formato PDF a la plataforma Moodle del curso junto con el código fuente del proyecto.

Inasistencias.

La evaluación final del próximo 20 de mayo es de carácter obligatorio. En caso de inasistencia, cada estudiante será calificado con nota 1,0, y deberá rendir a final de semestre la prueba solemne recuperativa.

Planificación de la actividad

Carrera: Ingeniería Civil Informática.

Asignatura: Tecnologías de la Información.

Contenidos: Bloques de construcción de programas, herencia, recursividad, listas, arreglos.

Fecha: 10 de abril al 20 de mayo.

La Tabla 15 muestra en detalle el cronograma de trabajo que se va a desarrollar a lo largo de este proceso, en el cual se incluye:

- Fecha de la sesión actual.
- Las acciones que se van a desarrollar al interior del aula.
- Descripción detallada de dichas acciones.
- El tiempo estimado de duración de cada una de ellas.
- Los productos que se tendrán que generar como consecuencia de las acciones antes mencionadas.
- Materiales o recursos didácticos que se van a utilizar.
- Actividades que los estudiantes en sus grupos de trabajo deberán realizar fuera del aula.

Tabla 15. Planificación de la segunda actividad ABP

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
Miércoles 10/04/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Lectura comprensiva por parte de los estudiantes de las observaciones que el profesor ha introducido.	Apunte del profesor. Moodle. Proyector. Computador. Alice. Pizarrón.	Cada grupo debe introducir los cambios que se indicaron en la corrección del informe. Identificación de las necesidades de aprendizaje que hayan detectado los estudiantes en sus grupos de aprendizaje.
	Entrega de las calificaciones de la primera experiencia ABP.	El profesor se reúne alternadamente con cada uno de los grupos, y les entrega una completa retroalimentación con respecto al desempeño exhibido en la primera parte del proyecto. Además, les indica que el próximo miércoles 17 de abril deben tener listas las correcciones que se hicieron al trabajo.	35’	Los estudiantes comienzan a actualizar sus informes de trabajo, en virtud de las recomendaciones dadas por el profesor.		
	Entrega de las bases del trabajo ABP.	Los alumnos, en sus grupos de trabajo leen la especificación del problema y lo discuten entre ellos.	10’	Lectura comprensiva por parte de los estudiantes, para llegar a un consenso general en torno a la manera en cómo se percibe el trabajo grupal.		

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Entrega de las rúbricas para la confección del informe final y para la presentación.	El profesor le entrega a cada alumno las rúbricas que se emplearán para calificar el informe y la presentación del lunes 8 de abril. Además, aclara las dudas que pudieran surgir a partir de cada uno de los ítems considerados en estos documentos.	5’			
	Formulación de preguntas	Los estudiantes elaboran preguntas en relación al desglose de la segunda parte del proyecto, que se materializan a través de una lista con interrogantes que emergen como consecuencia de la lectura al problema planteado.	10’	El profesor explica en grandes rasgos cada una de las etapas que vienen a continuación, para que el proyecto final quede de acuerdo a lo requerido en el escenario de trabajo.		

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Entrega del apunte que trata sobre funciones y estructuras condicionales «*if-else*» que pueden aparecen en Alice. (anexo C, páginas 135 a la 167).	El profesor habilita en la plataforma Moodle el apunte sobre funciones y estructura condicional en Alice.	10’	Apunte publicado en la plataforma Moodle del curso. Lectura comprensiva por parte de los estudiantes con respecto a los contenidos ligados con sentencias condicionales.		
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Viernes 12/04/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	El profesor brinda una retroalimentación que va relacionada con las necesidades de aprendizaje que fueron detectadas por los	Apunte del profesor. Moodle. Proyector.	Cada grupo continúa con la incorporación de los cambios que el profesor le ha recomendado a cada uno.

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Taller grupal sobre funciones y estructuras condicionales.	Los alumnos trabajan en sus respectivos grupos los conceptos y ejemplos planteados en el apunte sobre funciones y estructuras condicionales.	70’	estudiantes. Lectura comprensiva en grupo, en la cual ellos debieran asimilar que las estructuras condicionales pueden aparecer tanto en funciones como en métodos, ya sea de mundo o de clase. Los alumnos asimilan los conceptos de función y de estructura condicional.	Computador. Alice. Pizarrón.	Los alumnos desarrollan un esquema del problema que están abordando, que debe dar cuenta del análisis parcial y mejorado que lleve hasta el momento. En esta instancia, debiera contemplar una extrapolación de cuáles serían algunas funciones que pudieran emerger en los proyectos particulares de los grupos, como consecuencia de la lectura comprensiva hecha en clase.
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Lunes 15/04/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Lectura comprensiva en grupo. Los alumnos asimilan los conceptos de función y de estructura	Apunte del profesor. Moodle. Proyector.	Cada grupo continúa con la incorporación de los cambios que el profesor le ha recomendado a cada uno.

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Taller grupal sobre funciones y estructuras condicionales	El profesor les solicita a los alumnos que empiecen a hacer un bosquejo de las funciones que van a requerir implementar en el programa, y que vislumbren algunas sentencias condicionales que empiecen a aparecer.	35’	condicional. Codificación de las funciones en Alice. Recopilación de material adicional que sea necesaria para ahondar más en torno a lo que se ha presentado.	Computador. Alice. Pizarrón.	Los alumnos analizan la información que hayan recopilado, y en sus grupos trabajan con el material entregado, y determinan cómo enriquecer lo obrado a partir de los nuevos conceptos e ideas que se han incorporado en la clase.
	Revisión de los primeros aprontes.	Cada uno de los grupos de trabajo recibe una retroalimentación de parte del profesor en cuanto a la forma en cómo llevan la implementación de las funciones en sus proyectos.	35’ ,	Retroalimentación que el profesor le brinda a cada grupo en función de lo obrado hasta el momento. Codificación de las funciones en Alice.		
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Miércoles 17/04/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la	5’		Apunte del profesor. Moodle.	Cada grupo incorpora tanto al informe final como en Alice, la cáscara de las funciones

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
		clase en sus grupos de trabajo.			Proyector. Computador. Alice. Pizarrón.	que va a utilizar. A contar de ahora, se entenderá que el informe final corresponde a la modificación del primero, combinado con las estructuras de programación que se están revisando clase a clase.
	Revisión de los cambios que cada grupo introdujo en el primer informe.	El profesor revisa con cada grupo los cambios que introdujeron en sus respectivos informes.	35’			
	Incorporación de funciones y estructuras condicionales al informe.	Cada grupo prosigue con la incorporación de funciones y estructuras condicionales, en función de la retroalimentación dada por el profesor.	25’	Incorporación de las funciones y estructuras condicional al informe final. Codificación de las funciones en Alice.		
	Solicitud de incorporación de funciones y condicionales al informe.	El profesor se reúne con cada jefe de grupo, y le solicita que su grupo incorpore al trabajo final — informe con programa — funciones y estructuras condicionales.	10 ,	Reunión del profesor con cada uno de los jefes de grupo, con el fin de recibir una retroalimentación sobre el estado de avance del proyecto, y para determinar objetivos de trabajo a corto plazo.		
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
Viernes 19/04/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Apunte publicado en la plataforma Moodle del curso.	Apunte del profesor. Moodle. Proyector. Computador. Alice. Pizarrón.	Generación de algún resumen sobre los temas que se han tratado hasta el momento, y que los alumnos reflexionen sobre la importancia que conllevaría para sus proyectos la incorporación de este importante bloque de programación. Los alumnos reestructuran el esquema de trabajo inicial, e incorporan de manera paulatina esta nueva característica, para determinar cómo les puede ser de utilidad para enfrentar de mejor manera el problema planteado.
	Entrega del apunte de bucles (anexo C, páginas 167 a la 201).	El profesor habilita en la plataforma Moodle el apunte que trata sobre bucles.	5’			
	Taller grupal sobre bucles.	Los alumnos trabajan en sus respectivos grupos los conceptos y ejemplos planteados en el apunte sobre bucles.	65’	Lectura comprensiva del apunte, en la cual los estudiantes deben comprender que estos conceptos se pueden transportar al ámbito de funciones y de métodos, tanto de mundo como de clase.		
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
Lunes 22/04/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Los alumnos identifican la presencia de algunos bucles que puedan emerger en sus proyectos, y que afecten tanto a las funciones como a los métodos que hayan definido. El profesor revisa grupo a grupo el estado de avance de los trabajos.	Apunte del profesor. Moodle. Proyector. Computador. Alice. Pizarrón.	Inclusión de funciones y estructuras condicionales tanto al informe como al programa final.
	Taller grupal sobre bucles.	El profesor solicita que los alumnos identifiquen aquellos casos presentes en funciones o en métodos que ameriten el uso de algún bucle.	70’			
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Miércoles 24/04/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Los alumnos desarrollan problemas relacionados con creación de eventos, y determinan cómo poder implementar aquellos	Apunte del profesor. Moodle. Proyector.	Cada grupo incorpora al informe final y al programa los bucles que sean necesarios, para enriquecer y complementar la

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Integración de todos los contenidos vistos con anterioridad.	Los alumnos en sus grupos de trabajo integran los conceptos de bucle con los de estructura de control, y determinan posibles anidamientos de un bloque dentro de otro.	55’	eventos que hayan sido definidos en el informe final de la primera experiencia ABP.	Computador. Alice. Pizarrón.	estructura semántica de cada función, o de cada método. Los grupos de trabajo preparan una primera aproximación a lo que será el trabajo final, en función de todo el material que se ha leído y discutido hasta el momento.
	Solicitud de incorporación de las nuevas estructuras de programación al informe.	El profesor se reúne con cada uno de los jefes de grupo, y le solicita que integren todo lo visto hasta el momento en sus respectivos informes, indicándoles que el lunes 29 de abril se hará la primera revisión formal al respecto.	15’	Reunión entre el profesor con cada uno de los jefes de grupo, con el fin de generar una mutua retroalimentación con respecto al avance parcial que lleven.		
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Viernes 26/04/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Los alumnos presentan las mejoras que hayan hecho al profesor, en una nueva instancia de retroalimentación mutua entre ambas	Apunte del profesor. Moodle. Proyector.	Los alumnos analizan la información que tienen hasta el momento, y la procesan de tal manera que esto les ayude a avanzar con la

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Taller de integración.	Los alumnos incorporan en el producto final funciones, sentencias condicionales y bucles.	45’	partes.	Computador. Alice. Pizarrón.	codificación de la solución final, y las mejoras que haya que agregar al informe.
	Revisión del estado de avance.	El profesor se reúne con cada grupo para retroalimentarlos en cuanto al estado de avance particular que llevan en sus proyectos.	25’			
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Lunes 29/04/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Lectura del material entregado por el profesor.	Apunte del profesor. Moodle. Proyector.	Los alumnos analizan la información que tienen hasta el momento, y la procesan de tal manera que esto les ayude a avanzar con la

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Revisión de la integración de las estructuras de programación ya vistas al informe final.	El profesor revisa con cada grupo la incorporación de funciones, estructuras condicionales y bucles al informe final. Cada jefe de grupo se hace responsable de subir el archivo modificado a la cuenta Moodle del grupo antes que termine la clase.	70’		Computador. Alice. Pizarrón.	codificación de la solución final, y las mejoras que haya que agregar al informe.
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Viernes 03/05/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de	5’		Apunte del profesor. Moodle. Proyector.	Codificación de la solución final, y elaboración del informe.

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
		trabajo.			Computador. Alice. Pizarrón.	
	Taller grupal sobre bucles.	Los alumnos, en sus grupos, leen el material didáctico sobre bucles finitos e infinitos, y trabajan con los conceptos y ejemplos que aparezcan mencionados.	55’			
	Entrega de los resultados de las evaluaciones a cada grupo.	El profesor le entrega y comenta con cada grupo los resultados particulares que obtuvieron de las observaciones realizadas hasta el momento.	15’	Entrega de la evaluación sumativa aplicada a cada grupo para determinar el grado de avance, a partir de lo obrado hasta el día lunes 29 de abril.		
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Lunes 06/05/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’		Apunte del profesor. Moodle. Proyector.	Codificación de la solución final, y elaboración del informe, junto con las mejoras que hayan debido incorporar hasta la

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Entrega del apunte de listas y arreglos (anexo C, páginas 203 a la 243).	El profesor habilita en la plataforma Moodle el apunte sobre listas y arreglos.	5’	Los estudiantes hacen una lectura reflexiva sobre el material entregado que trata de listas y arreglos. Cada grupo determina cómo estas estructuras de datos se puede emplear en beneficio de su propio proyecto.	Computador. Alice. Pizarrón.	fecha.
	Taller grupal sobre listas y arreglos.	Los alumnos leen, con sus grupos de trabajo, el material que trata sobre listas y arreglos, y ellos replantean la posibilidad de buscar información adicional que les permita complementar lo que se les ha entregado.	65’			
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Miércoles 08/05/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’		Apunte del profesor. Moodle. Proyector.	Los grupos de trabajo empiezan a determinar cómo se pueden transportar los conceptos de listas y arreglos al ámbito de su

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Taller grupal sobre listas y arreglos.	Los alumnos leen, con sus grupos de trabajo, el material que trata sobre listas y arreglos, y ellos replantean la posibilidad de buscar información adicional que les permita complementar lo que se les ha entregado.	60’	Desarrollo de los ejemplos propuestos, con el fin de asimilar los conceptos involucrados.	Computador. Alice. Pizarrón.	proyecto particular.
	Reunión de integración.	El profesor se reúne con cada uno de los jefes de grupo, y le explica que el día miércoles 15 de mayo se hará una primera revisión que contemplará la integración de todo lo visto hasta el momento, en cuanto a funciones, estructuras condicionales, bucles, listas y arreglos.	10’	Materialización de la reunión entre el profesor con cada jefe de grupo.		
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima	5’			

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
		sesión.				
Viernes 10/05/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	Re-estructuración de las funciones y métodos que se han definido hasta el momento, que tiene como objetivo incorporar listas y arreglos, según corresponda.	Apunte del profesor. Moodle. Proyector. Computador. Alice. Pizarrón.	Codificación de la solución final, y elaboración del informe, que dé cuenta de todos los avances que se han estado haciendo hasta el momento. Los estudiantes reestructuran el esquema de trabajo, en función de la incorporación de estructuras de datos como listas o arreglos.
	Taller de listas y arreglos.	Cada grupo continúa desarrollando los ejemplos del apunte, y deben contextualizar lo que se ha visto con su proyecto de trabajo.	35’			
	Revisión del estado de avance.	El profesor se reúne con cada grupo para monitorear y apoyar la incorporación de estas importantes estructuras de datos al proyecto final.	35’	Retroalimentación del profesor a cada grupo en función de todos los cambios y mejoras que se hayan incorporado al informe y al programa.		

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Lunes 13/05/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’	El profesor recorre cada grupo, y les entrega algún consejo para mejorar lo que están haciendo hasta el momento.	Apunte del profesor. Moodle. Proyector. Computador. Alice. Pizarrón.	Codificación de la solución final, y elaboración del informe, que dé cuenta de todos los avances que se han estado haciendo hasta el momento. En esta instancia, se requiere de la participación de todos los miembros del equipo de trabajo, a objeto que cada uno tenga claridad con respecto a lo que sucederá el próximo 20 de mayo.
	Integración de todos lo que se ha realizado hasta el momento.	Los grupos trabajan en la implementación del programa final, junto con la generación del informe.	70’			
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Miércoles 15/05/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la	5’	Revisión del trabajo que hasta el momento hayan estado desarrollando los	Apunte del profesor. Moodle.	Codificación de la solución final, y elaboración del informe, que dé cuenta de todos

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
		clase en sus grupos de trabajo.		grupos.	Proyector. Computador. Alice. Pizarrón.	los avances que se han estado haciendo hasta el momento.
	Revisión del estado de avance.	El profesor se reúne con cada grupo para revisar lo que se ha hecho hasta el momento, tanto en informe como en programa.	70’			
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Viernes 17/05/2013	Introducción de las actividades a desarrollar en la clase.	El profesor brinda una síntesis de las actividades que los alumnos desarrollen durante la clase en sus grupos de trabajo.	5’		Apunte del profesor. Moodle. Proyector. Computador. Alice. Pizarrón.	Mejoramiento del trabajo final, en función de las observaciones entregadas por el profesor, tanto del informe como del programa.
	Síntesis de lo realizado.	El profesor brinda una síntesis del proceso que han estado desarrollando los alumnos, y brinda antecedentes que ayuden a enriquecer el contenido del informe.	35’			

Sesión	Etapas, pasos, acciones	Descripción detallada de las acciones	Tiempo	Productos	Materiales requeridos	Actividades/productos fuera del aula
	Entrega de los resultados de las evaluaciones a cada grupo.	El profesor le entrega y comenta con cada grupo los resultados particulares que obtuvieron de las observaciones realizadas hasta el momento.	35’	Entrega de la evaluación sumativa, que da cuenta del estado de avance de cada grupo en su proyecto, a partir de lo mostrado en las clases de integración.		
	Resumen de lo expuesto en la clase.	El profesor cierra la clase dando un resumen de lo que ha sido la jornada del día, y comenta lo que se hará en la próxima sesión.	5’			
Lunes 20/05/2013	Exposición de grupos.	Cada grupo entrega y expone su trabajo.	80’	Informe, programa y exposición de cada grupo.	Moodle. YouTube Proyector. Computador. Alice. Pizarrón.	

Evaluación

Informe

Tabla 16. Rúbrica para evaluar el segundo informe

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Portada	Incluye todos los elementos solicitados en la descripción del trabajo de manera clara y ordenada.	Incluye todos los elementos solicitados, pero los nombres de los integrantes del grupo no vienen ordenados por apellido paterno en forma ascendente, o bien la portada abarca entre cinco y siete de los elementos contemplados, con algunos errores moderados de formato.	La cantidad de elementos incorporados incluye entre dos y cuatro del total de ítems totales, con algunos errores moderados de formato.	No incluye portada, o bien se consideró sólo uno de los ocho elementos considerados, o la información global contenida en la portada no es suficiente.
	10 puntos.	**6 puntos.**	**4 puntos.**	**0 punto.**

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Índice	Incluye índice completo, con un máximo de tres niveles de profundidad, generado de manera automática, y con una adecuada descomposición de los temas.	El índice ha sido construido de manera automática, pero no contiene un desglose apropiado de los temas solicitados en el cuerpo principal del informe.	El informe está generado de forma manual, con algunos errores en la numeración de las páginas, y sin que se aprecie una secuencia lógica en el orden de los puntos solicitados.	El índice no está presente en el informe, o bien ha sido realizado en forma manual con errores en casi todos los números de páginas, sin que se aprecie una secuencia lógica en el orden de los puntos solicitados.
	5 puntos.	**3 puntos.**	**1 punto.**	**0 punto.**
Introducción	La introducción incluye el resumen del trabajo, junto con los objetivos a alcanzar de una manera clara y ordenada.	Incluye resumen y objetivos, con algunos errores moderados que no afectan la legibilidad ni la coherencia interna de lo expuesto.	El resumen del trabajo no considera todos los aspectos involucrados, y los objetivos no son coherentes con lo que se solicita.	No está incluida en el informe, o bien no es clara en cuanto al planteamiento de las ideas, y no especifica los objetivos del trabajo.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**
Contenido / Descripción del Problema	La descripción del problema es completa, y no deja lugar a ambigüedades, ya que es congruente con lo enunciado en la introducción.	La descripción del problema excluye algunos elementos menores que no alteran de manera significativa la formulación del mismo.	La descripción del problema presenta ciertas lagunas argumentativas que alteran de manera notoria su conformación interna.	No está presente, o bien es completamente diferente con lo descrito en la introducción.
	10 puntos.	**6 puntos.**	**3 puntos.**	**0 punto.**

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Contenido / Elaboración del libreto	El libreto o algoritmo general del proyecto mantiene una correspondencia clara con lo enunciado en la descripción del mismo.	El libreto del proyecto presenta algunos errores menores que se pueden superar rápidamente.	El libreto presenta serias incoherencias con respecto a lo formulado en la descripción del proyecto de trabajo.	El libreto no está presente, o bien evidencia graves incongruencias con respecto a la descripción dada, al no contener todos los elementos descritos en el punto anterior.
	25 puntos.	**20 puntos.**	**10 puntos.**	**0 punto.**
Contenido / Clases	Las clases son pertinentes y suficientes para la descripción del proyecto.	Las clases aparecen descritas en cantidad suficiente y se especifican con algunos errores moderados.	Las clases que se han definido no son suficientes para alcanzar a cubrir todos los aspectos del programa. Sin embargo, se aprecia que se trata de omisiones menores que no impiden una correcta lectura de la especificación del proyecto.	Las clases no están definidas, o bien aquellas que están presentes no cubren las exigencias del guión textual que se ha entregado, ya que la carencia de ellas es evidente, y la explicación brindada contiene errores severos en cuanto a forma y fondo.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Contenido / Métodos	Los métodos que se han descrito dan cuenta de un desarrollado sentido de la abstracción, en el sentido de haber escogido aquellos conjuntos de acciones que se van a desarrollar con mayor frecuencia en el programa. Además, queda claramente especificado si éstos serán de nivel de mundo o de objeto, y la documentación entregada es consistente con los distintos bloques de construcción de programas.	Los métodos que se han definido dan cuenta de la capacidad de abstracción del grupo, pero no hace una diferencia entre aquellos que sean de mundo, con respecto a los de nivel de objetos. La documentación que se ha proporcionado contiene moderados errores en cuando al uso de bloques de construcción de programas.	Existen errores graves pero no numerosos en cuanto a la definición de métodos de mundo o de objeto. La documentación entregada presenta omisión de comentarios en el código fuente, y no queda claro el por qué de la elección de una u otra estructura de programación.	Ningún método ha sido definido, o bien lo que se ha presentado contiene graves y reiterados errores en cuanto a la completitud y forma de la documentación.
	20 puntos.	**15 puntos.**	**10 puntos.**	**0 puntos.**
Contenido / Funciones	Las funciones que se han definido son congruentes con la totalidad que se requieren para el proyecto abordado, y su documentación es apropiada y bien explicada, sin errores.	Las funciones mostradas contienen moderados errores en cuanto a documentación.	La mayoría de las funciones contienen una documentación demasiado ambigua y no deja claro la real utilidad de cada una de ellas.	El grupo no definió funciones, o bien la documentación mostrada es inexistente, o con reiteradas y graves fallas u omisiones.
	20 puntos.	**15 puntos.**	**10 puntos.**	**0 puntos.**

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Contenido / Descripción de los eventos	Los eventos que el grupo ha considerado permiten identificar de manera clara y suficiente la manera en cómo se dará el curso de las acciones y de las interacciones propias del proyecto.	Los eventos de programación que se han considerado permiten describir claramente la mayor parte del flujo de acciones del proyecto, con algunas omisiones menores.	Los eventos que se han definido dan cuenta de una cantidad significativa de omisiones de continuidad de las acciones del producto final, que alteran en demasía la esencia del proyecto.	No existe una descripción de los eventos, o bien, lo que se ha definido no permite en absoluto mantener una secuencia clara y ordenada del curso de acciones del producto que se va a implementar.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**
Contenido / Maquetación	Los prototipos que se muestran en el informe expresan con claridad todos los posibles escenarios que se puedan dar durante el desarrollo del programa.	Los prototipos que se muestran presentan errores moderados en cuanto a los diferentes escenarios del proyecto.	La mayoría de los prototipos evidencian grandes problemas de diseño e incongruencia con respecto al planteamiento del proyecto, y que en cuanto a cantidad alcancen un número significativo.	No se muestra una maquetación del trabajo, o bien que los prototipos exhibidos son insuficientes y que en su mayoría evidencian graves incongruencias con el planteamiento del proyecto formulado al inicio de esta parte.
	10 puntos.	**6 puntos.**	**3 puntos.**	**0 punto.**

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Conclusión	La conclusión del trabajo abarca de manera completa cada uno de los puntos solicitados en el enunciado del trabajo.	La conclusión presenta algunas incongruencias leves con respecto a los objetivos del proyecto, y hace un buen tratamiento de los logros y dificultades encontradas.	La conclusión no abarca a todos los objetivos del trabajo, ya que se limita a unos cuantos, y no hay una reflexión grupal en torno a logros y dificultades encontradas.	La conclusión no existe, o bien no contempla objetivos o alguna reflexión sobre logros y dificultades.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**
Bibliografía	La bibliografía es pertinente al desarrollo del trabajo, el sistema de citas respeta el estándar APA, y fue generada de forma automática.	La bibliografía presenta algunas omisiones leves, pero las citas siguen el estándar APA, y fue generada de forma automática.	La bibliografía no es completa, el estándar APA no fue empleado de manera correcta.	La bibliografía no existe, o bien fue hecha de forma manual, las fuentes no están debidamente documentadas y no hay presencia de citas en formato APA.
	5 puntos.	**4 puntos.**	**2 puntos.**	**0 punto.**
Presentación	El informe cumple a cabalidad con cada uno de los aspectos formales solicitados.	El informe presenta algunas faltas moderadas de ortografía y/o de redacción, y preserva una estructura tipográfica uniforme a lo largo de la escritura del mismo.	El informe presenta reiteradas faltas de ortografía y/o de redacción, y en algunos pasajes se aprecia que la estructura tipográfica que se solicita no se respeta.	El informe no presenta una cohesión interna con respecto a la redacción de las ideas, y en varios pasajes se aprecian severas faltas de ortografía que dificultan su lectura.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**

Programa

Tabla 17. Rúbrica para evaluar el programa

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Instructivo de uso.	El instructivo de uso del software explica de manera clara el funcionamiento del mismo, y se expone al inicio del programa. Además cuenta con el logo de la empresa.	El instructivo contiene la mayor parte de los comandos que el usuario deberá introducir, junto con el logo de la empresa.	El instructivo presenta algunas inconsistencias y carencias con respecto a los eventos que el grupo declaró en el informe escrito. Puede contener o no el logo de la empresa.	El instructivo no existe, o bien, lo que se ha presentado no es congruente con la mayoría de los eventos que se detallaron en el informe. Puede contener o no el logo de la empresa.
	10 puntos.	**6 puntos.**	**4 puntos.**	**0 punto.**
Concordancia entre el instructivo y lo que finalmente termina haciendo el programa.	Se aprecia que el funcionamiento del programa es absolutamente congruente con lo que se indicó en el instructivo.	No se pudo ejecutar a plenitud el videojuego, ya que era evidente que algunos comandos de teclado o de ratón no fueron debidamente declarados.	Algunos de los comandos no responden de acuerdo a lo que se especificó en el instructivo inicial del juego.	Prácticamente ninguno de los comandos de teclado o de ratón responden de acuerdo a lo dado en el informe o instructivo.
	10 puntos.	**6 puntos.**	**4 puntos.**	**0 punto.**

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Desarrollo de las acciones del videojuego.	El juego responde de manera coherente con todo lo que se escribió en el informe, en términos de la lógica argumentativa que se utilizó, y de las clases, métodos y funciones que se emplearon para esos fines.	Se observan algunas inconsistencias menores en determinados pasajes del juego, con respecto a todo lo que se indicó en el informe, especialmente en el curso de determinadas acciones del mismo.	Existen severas incongruencias con respecto a lo indicado en el informe, ya que hay eventos o estados del programa que no se dan dentro de la ejecución del mismo.	Lo que se ha entregado es completamente incoherente con respecto a lo que se indicó en el informe, y por lo tanto, el producto de software que se ha entregado no está íntimamente ligado con la documentación que se proporcionó.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**
Tráiler.	El avance que se ha subido a YouTube sintetiza de manera correcta el curso de las acciones del videojuego, y además el tiempo de reproducción se ajusta de manera perfecta con la melodía que se puso de fondo.	El avance deja fuera algunos elementos menores del proyecto, se ha subido a YouTube, y presenta diferencias de tiempo moderadas con respecto a lo que está dado en la pista de audio.	El avance no se ha subido a YouTube, pero al menos mantiene cierta coherencia con lo que se escribió en el informe. La melodía de fondo no está presente, o bien la duración total del clip no es igual con el largo de la pista de audio.	El grupo no ha subido el avance del videojuego a YouTube, y/o carece de melodía de fondo, y/o lo que se muestra poco y nada tiene que ver con lo que se ha indicado en el informe del proyecto.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**

Exposición

Tabla 18. Rúbrica para evaluar la segunda exposición

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Contenido	Demuestra un completo entendimiento del tema, junto a una presentación que los contempla a todos.	Demuestra un buen entendimiento del tema, con algunos errores moderados.	Demuestra un dominio de una escaza cantidad de los temas abordados.	No pareciera tener un dominio de lo que está hablando. Se nota inseguridad a lo largo de toda la presentación.
	10 puntos.	**6 puntos.**	**4 puntos.**	**0 punto.**
Despierta interés por el proyecto.	El planteamiento del proyecto se formula de manera interesante, ya que genera debate y preguntas de la audiencia.	Quien expone contribuye a la generación de preguntas, pero no despierta mayor interés por iniciar algún debate en torno a lo presentado.	La persona que expone no despierta mayor interés entre sus compañeros, con la salvedad de algunas tibias preguntas.	El tema ha sido mal planteado desde el principio, y no genera preguntas ni debates.
	10 puntos.	**6 puntos.**	**4 puntos.**	**0 punto.**
Respuestas grupales.	Todos los miembros del grupo responden de manera asertiva a las preguntas formuladas por el profesor.	Las respuestas dadas por los miembros del grupo presentan algunos errores moderados.	Las respuestas dadas por los integrantes del grupo presentan severas inconsistencias.	La respuesta de algunos miembros del grupo evidencia que no hubo una participación activa dentro del desarrollo del proyecto, que se hizo notoria debido a la inseguridad mostrada al hablar.
	20 puntos.	**15 puntos.**	**5 puntos.**	**0 punto.**

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Presentación en Prezi o en PowerPoint	Buen tamaño letra, buen contraste con el fondo de las diapositivas, diseño de pantalla no recargado, buena selección gráficos, fotos y/o figuras.	El tamaño y contraste de las letras no es lo óptimo, demasiada escritura, imágenes adecuadas pero no suficientes, sobre todo en cuanto a la maquetación.	Tamaño de letras muy reducido, escaza cantidad de figuras.	Demasiada escritura, casi nada de síntesis, figuras poco apropiadas que configuran una presentación poco atractiva para la audiencia.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**
Pronunciación	Habla claramente y distintivamente todo el tiempo, y no tiene mala pronunciación.	Se expresa de manera clara en casi todos los aspectos, con algunos errores moderados de pronunciación.	Más de la mitad del tiempo habla bien, pero sin destacar.	A menudo habla entre dientes, o no se entiende lo que dice la mayor parte del tiempo.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**
Vocabulario	Emplea un vocabulario apropiado para la audiencia, y cuando es necesario explica con claridad todos los términos que se empleen.	Emplea un vocabulario apropiado para dirigirse a la audiencia, pero no define los términos nuevos que vayan surgiendo.	Usa un vocabulario que es limitado para la audiencia, y no hace explicación de los términos propios que emerjan del proyecto.	Usa palabras u oraciones que no son entendidas por la audiencia, el lenguaje empleado es bastante pobre, y tiende a repetir en demasía las mismas palabras.
	15 puntos.	**10 puntos.**	**5 puntos.**	**0 punto.**

Aspecto a evaluar	Excelente	Muy Bueno	Regular	Insuficiente
Volumen de voz	El volumen es lo suficientemente alto para ser escuchado por todos los miembros de la audiencia a través de toda la presentación.	El volumen es lo suficientemente alto para ser escuchado por todos los miembros de la audiencia al menos el 90% del tiempo.	El volumen es lo suficientemente alto para ser escuchado por todos los miembros de la audiencia al menos el 60% del tiempo.	El volumen con frecuencia es muy débil para ser escuchado por todos los miembros de la audiencia.
	10 puntos.	**6 puntos.**	**3 puntos.**	**0 punto.**
Tiempo	El tiempo de exposición varía entre 10 y 15 minutos.	La presentación dura entre 8 y 10 minutos.	La presentación dura entre 5 y 7 minutos.	La presentación dura menos de 5 minutos, o más de 15.
	10 puntos.	**6 puntos.**	**3 puntos.**	**0 punto.**

Auto-evaluación

Para cada una de las aseveraciones que se indican a continuación, marque con una X aquella que mejor refleje lo que usted ha trabajado por su grupo.

Tabla 19. Segunda autoevaluación

Aspecto	Totalmente de acuerdo (4 puntos)	De acuerdo (3 puntos)	En desacuerdo (2 puntos)	Totalmente en desacuerdo (0 punto)
Me siento cómodo en el grupo.				
He tomado con responsabilidad las tareas encomendadas por el grupo.				
Me siento comprometido con el trabajo que estoy realizando por el grupo.				
Creo haber aportado como debiera para mejorar el				

Aspecto	Totalmente de acuerdo (4 puntos)	De acuerdo (3 puntos)	En desacuerdo (2 puntos)	Totalmente en desacuerdo (0 punto)
rendimiento del grupo.				
Creo que mis aportes son tomados en cuenta por los demás integrantes del grupo.				
Aporto ideas que contribuyen a la toma de decisiones importantes dentro del grupo.				
Llego a tiempo para las reuniones de trabajo del grupo.				
Cumplí con los plazos de entrega de cada tarea que me fue encomendada.				
Mantengo buenas relaciones con los demás integrantes del grupo.				
He logrado dar a conocer mis planteamientos con claridad.				
He contribuido a que otros compañeros también participen.				

UNIVERSIDAD DEL BÍO-BÍO
Facultad de Educación y Humanidades
Departamento de Ciencias de la Educación

Profesor Guía:
Dr. Pedro Salcedo Lagos

Cátedra:
Proyecto de Intervención e Innovación Pedagógica

TEXTO DE APOYO PARA LA ENSEÑANZA DE LA PROGRAMACIÓN ORIENTADA A OBJETOS EN ENTORNOS VIRTUALES

Milton Agustín Ramírez Klapp

Anexo C del Informe de Tesis

Chillán, Julio de 2013

Índice General

Índice de Tablas

Índice de Figuras

Prólogo

Estimados alumnos y alumnas.

Este apunte ha sido elaborado para que usted desarrolle las destrezas necesarias para desenvolverse con éxito en el apasionante mundo de la programación computacional orientada a objetos. En este texto podrá encontrar las herramientas necesarias para que pueda afrontar con éxito el desarrollo del pensamiento algorítmico y sistémico, que es tan necesario dentro de la carrera de Ingeniería.

Para lograr un efectivo logro de los aprendizajes propuestos, se le recomienda de manera encarecida que intente resolver los ejercicios — por su cuenta — y proyectos — con otro compañero — que vienen propuestos al final de cada parte de alguna unidad.

En la medida que proceda de la manera indicada en el párrafo anterior, notará que la confianza en sí mismo va a aumentar, junto a lo que usted debiera terminar aprendiendo.

Estimados profesores y profesoras.

Este texto ha sido confeccionado con el objetivo de entregar una mirada alternativa al enfoque clásico con el cual se aborda la enseñanza de la programación orientada a objetos. El entorno de desarrollo que se ocupa en esta oportunidad es el software educativo Alice, que permite introducir al estudiante en los conceptos y técnicas básicas para que desarrolle destrezas necesarias que le permitan afrontar con mejor base los desafíos que vienen en los cursos posteriores.

Este apunte corresponde a una adaptación del libro oficial[1] que se utiliza para apoyar el proceso de enseñanza-aprendizaje de este enfoque computacional, y pretende ser un vínculo entre el estudiante y la herramienta Alice que fue desarrollada en la Universidad de Carnegie Mellon.

En breves palabras, Alice es un software educativo que tiene como propósito apoyar el aprendizaje de la programación a través de un entorno de creación de mundos virtuales en los cuales el alumno recrea animaciones o videojuegos a través de una interfaz basada en arrastrar y soltar.

Este apunte ha sido estructurado en torno a cinco unidades didácticas, donde se cubren los principales aspectos involucrados en la enseñanza inicial de un curso de orientación a objetos. Al final de cada una de ellas se ofrece un resumen, ejercicios propuestos que están diseñados especialmente para ser abordados durante las sesiones de ayudantía, junto con un apartado que entrega algunas orientaciones de carácter educativo al profesor, que se desglosan en hechos concretos que el estudiante debe tener en cuenta para alcanzar los aprendizajes esperados, más una serie de preguntas que están pensadas para ser abordadas en las clases de práctica.

Al término de la quinta unidad, se dan algunas recomendaciones de carácter general con respecto al uso de BlueJ como siguiente recurso didáctico a trabajar, pensando que este texto está contextualizado a lo que se hará durante el curso de «*Tecnologías de la Información*».

[1] Dann, W., Cooper, S., & Pausch, R. (2012). *Learning to Program with Alice*. Pittsburgh: Editorial Pearson.

Los invito a trabajar de manera sistemática en pro de la consecución de los objetivos de aprendizaje que se han trazado en este largo camino del conocimiento.

Milton A. Ramírez Klapp
Autor.

Unidad 1

Introducción a la Programación

Esta unidad comienza con las características generales que tiene la ciencia de los algoritmos, junto a una presentación del entorno de programación Alice, que será la herramienta con la cual vamos a trabajar para aprender los conceptos más importancia de la orientada a objetos.

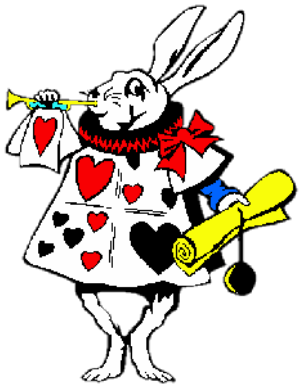
Sentido educativo de la unidad

Aprendizajes esperados.

- Identificar los elementos principales que componen el entorno de trabajo de Alice.
- Diseñar el bosquejo de una historia animada por medio de la metodología «*top-down*».

Contenidos.

- Lenguajes de programación.
- Características del entorno de programación Alice.
- Escenarios de trabajo, equivalencia entre historietas y algoritmos.
- Metodología de diseño «*top-down*».



Primera Parte

Introducción a Alice

1.1 Computación y Lenguajes de Programación

Introducción

Al igual que los idiomas sirven como medio de comunicación entre los seres humanos, existe una amplia variedad de lenguajes que permiten esta forma de expresión entre el hombre y los computadores. Estas herramientas — que se denominan lenguajes de programación — permiten expresarle a la máquina el conjunto de instrucciones que el programador — que es la persona que construye la aplicación — espera que ejecute el computado.

Por otro lado, el software — o programas — es aquello que le da vida al funcionamiento del hardware de cualquier máquina. Lo importante es que las aplicaciones para computador se realizan sobre un lenguaje de programación determinado. De esta manera, el aprendizaje de la programación es clave para poder construir herramientas que permitan hacer que nuestras máquinas se transformen en reproductores de música, en procesadores de texto, en un centro de entretención, etc.

Hay muchas razones para aprender a programar un computador: para algunas personas, la programación constituye en una fuente de entretención que les permite disfrutarla como un fin en sí misma. Sin embargo, para la mayoría de la gente, la escritura de programas informáticos constituye un medio para llegar a un fin, ya que el computador es una herramienta útil en todos los aspectos de la vida del hombre contemporáneo: desde que nacemos[2] hasta el día que partimos[3] estaremos siendo objeto de algún software. Entremedio, los avances de las ciencias de la computación le permiten al hombre vivir más tiempo.

Por ejemplo, la investigación médica se apoya con programas para obtener resultados a partir de estudios que se hagan. También está el caso de la industria automotriz: los vehículos que usamos a diario están controlados por computadores

[2] En algunas localidades, a los recién nacidos se les coloca un chip alrededor del tobillo, que contiene un rastreador para monitorear en todo momento su ubicación.
[3] Los dispositivos que se emplean en los hospitales para medir los signos vitales son computadores con un diseño diferente a cómo los conocemos, y que emplean un programa para esos fines.

que permiten activar — entre otras funciones — las bolsas de aire en caso de accidente, o mantener al conductor en conocimiento de los valores que marquen los diferentes indicadores de su máquina para saber en qué momento revisar el líquido de frenos, cuándo cargar combustible, etc. Otros usos que tienen los computadores dentro de nuestro diario vivir:

- Existencia de protocolos de comunicación que están presentes en la telefonía celular y en el acceso a Internet.
- Investigaciones sobre fenómenos de la naturaleza, por medio de estudio de los patrones migratorios de los animales, entre otros.
- Exploración del espacio, gracias a las simulaciones que se pueden hacer en instalaciones especialmente equipadas para ello.
- En industria del entretenimiento: en el cine, los efectos especiales sólo son posibles gracias a los computadores.
- En el campo de los videojuegos, su uso resulta más que evidente, ya que de lo contrario sería imposible poder construirlos.

La mayor parte de las personas que escriben el software para estos proyectos son programadores profesionales, que probablemente hayan tenido alguna especialización en Ciencias de la Computación en la universidad. Pero incluso es posible que sujetos que no hayan tenido estudios formales en esta área puedan perfectamente desarrollar sus propias soluciones, ya que muchas aplicaciones contemporáneas — como las hojas de cálculo o programas para construir sitios web — permiten a los usuarios finales extender el conjunto de prestaciones que ofrecen, por medio de herramientas llamadas «*macros*» que facilitan bastante estas labores.

Algo importante que se quiere alcanzar en este curso es que usted pueda desarrollar una nueva forma de pensar por medio del aprendizaje de la lógica de la programación orientada a objetos, ya que este campo le va a ayudar a solucionar problemas de maneras diferentes y válidas.

Presentación de Alice

En buena parte de este curso vamos a trabajar con una herramienta que se llama Alice, que hace posible escribir programas computacionales de una manera completamente diferente a como se hace de forma convencional, en el sentido que ahora el estudiante tendrá la posibilidad de ser el director de alguna historia, donde puede tener la responsabilidad de coordinar el flujo de acciones que permiten hacer que su relato cobre sentido, o que pueda estar al mando de la creación de un videojuego interactivo, en el cual tenga que estructurar el libreto de trabajo, modelar a los personajes, definir los escenarios, el flujo de las acciones que formen parte de él, etc.

Elementos básicos de la programación computacional

Un programa de computador un conjunto de instrucciones que le indican al computador qué hacer. Por supuesto, existen múltiples opciones para indicarle al computador lo que tenga que hacer; de hecho, los programadores suelen emplear palabras de uso cotidiano[4] para indicar las órdenes que la máquina habrá de ejecutar.

[4] En el ámbito de las Ciencias de la Computación, esto se conoce como «*lenguaje natural*».

Una de las tantas intenciones que tiene este apunte es que usted pierda el temor hacia la programación computacional, y que al finalizar el curso comprenda que una solución de software se compone de ideas bastante simples, que iremos revisando a lo largo de este viaje. Pensemos en algo tan cotidiano como ponernos las zapatillas en la mañana: ¿cuáles son las operaciones que debemos realizar, y en qué orden?:

1. Poner el calcetín izquierdo.
2. Poner el calcetín derecho.
3. Colocar la zapatilla izquierda.
4. Colocar la zapatilla derecha.

Es posible que usted a lo mejor haya pensado en un ordenamiento diferente para las acciones que se indican arriba, y que también nos van a conducir al resultado esperado. Bueno, esa es una de las primeras ideas que debe tener presente:

Un problema no necesariamente debe tener una solución única.

Algunas nociones de programación secuencial

Lo que se acaba de mostrar en el ejemplo anterior constituye algo que en las Ciencias de la Computación se conoce como **procesamiento secuencial**, que en términos sencillos significa indicarle a una máquina — o incluso a un ser humano — cómo realizar determinadas acciones, siguiendo un orden, o una secuencia — de ahí el nombre — determinada.

Existen también las denominadas **instrucciones condicionales**, que son aquellas que se llevarán a cabo siempre y cuando se cumpla alguna condición. A modo de ejemplo: «***si*** *está lloviendo,* ***entonces*** *saldré con paraguas*». Esto significa que el uso del paraguas estará sujeto a las condiciones meteorológicas del momento. ¿Alguna vez ha razonado de esa manera?, ¿Podría dar otro ejemplo de razonamiento condicional que utilice en su vida diaria?

Por otra parte, tenemos los **comportamientos repetitivos**, que también son comunes en lo cotidiano. Por ejemplo:

- «*Dar 20 vueltas alrededor de la manzana*».
- «*Mientras queden galletas en el plato, podemos seguir comiendo*»

Esta clase de repeticiones, se conoce en Ciencias de la Computación como **bucle, iteración o recursión.** ¿Puede dar algún otro ejemplo de bucles que usted ejecute a diario? Otra idea que es materia de estudio en esta asignatura corresponde a la técnica de **reduccionismo** o **metodología top-down**, que consiste en descomponer algún problema en trozos o unidades más pequeñas — llamados subproblemas — que permitan simplificar y desglosar cada una de sus etapas de resolución.[5] Y le apuesto que en más de una oportunidad lo ha usado en su vida. Por ejemplo, suponga el problema de limpiar una casa:

¿Cómo lo haría?

En primer lugar, podría partir por la cocina, baño, y terminar con los dormitorios. Note que la limpieza parcial de cada parte de la casa constituye de por sí un problema. Por lo tanto, la limpieza se habrá completado una vez que hayamos abordado cocina, baño y dormitorios, habremos completado la «*gran tarea*», que pasó

[5] Esto es algo que también se conoce en la cultura popular como «*dividir para conquistar*».

a ser el problema a resolver. ¿Puede dar algún ejemplo de realización de una tarea que involucre su descomposición en problemas menores?

También aprenderemos una idea muy importante que se conoce como **llamadas a funciones**, que tiene como objetivo hacer preguntas sobre «*algo*». Por ejemplo: determinar cuánto pesa una persona el día de hoy, para saber con cuántos kilos va a comenzar un programa de acondicionamiento físico. Este tema va muy ligado con lo de instrucciones condicionales, que se mencionó con anterioridad.

Es importante que usted note que parte de la esencia de la programación radica en las ideas que se acaban de nombrar, y que un programa computacional corresponde a una combinación de ellas (no necesariamente todas). También habrá notado que al final de cada párrafo se le pedía que diera algún ejemplo de aplicación de estos conceptos en el contexto de su vida cotidiana. Este gran paralelismo que podemos establecer entre ambos «*mundos*» — el de los computadores y el real — es lo que inspira la creación de este apunte que le enseñará algunas técnicas de diseño de programas que se apoyen en casos concretos de la vida real, con el fin que usted pueda adquirir de mejor manera los aprendizajes que se esperan al final de cada una de las unidades didácticas que componen este documento.

Entonces, la dificultad de programar radica en qué tan compleja sea la especificación del problema que debamos abordar. En este curso, usted aprenderá algunos trucos que le puedan ayudar a sortear con éxito estas dificultades, junto con la planificación de cómo escribir programas antes que usted pruebe que funcionen correctamente.

De hecho, cómo aprender a pensar a la hora de diseñar un programa es probablemente la parte más valiosa de este aprendizaje. Lo que aprenderemos en este curso corresponde a los principios de la programación orientada a objetos.

1.2 Conceptos de Alice

Como hemos mencionado en la introducción, Alice es un entorno que nos va a permitir crear mundos virtuales en el computador, y poblarlos con algunos personajes, para luego dar paso a escenas que puedan conducir a una animación, o bien a la creación de una película interactiva. En esta parte comenzaremos dando una introducción a los mundos virtuales en Alice, junto a algunas técnicas que ayudarán a crear una animación inicial.

Concepto: Mundo virtual

Los video juegos y simuladores pueden ser de dos o tres dimensiones — 2D o 3D —, y es posible que en más de una oportunidad haya tenido que usar alguno de ellos — como por ejemplo, en algún curso de conducción. Los pilotos de aeronaves utilizan simuladores de vuelo, porque eso es parte de su formación. La ventaja de las simulaciones es evidente: cuando el avión de combate se estrella a manos de un piloto novato, ninguna parte del aparato se va a llenar de humo: a lo más aparecerá algún mensaje acompañado de un sonido.

Un videojuego o simulación en 3-D se llama un *mundo virtual*. Un mundo virtual aporta un sentido de la realidad al simulador y aumenta su eficacia. Para ver la diferencia entre 2D y 3D, compare las imágenes en las Figuras 1 y 2. La imagen de la **¡Error! No se encuentra el origen de la referencia.** muestra el bosquejo de una

scena cualquiera dibujada a mano en versiones delantera y trasera. Es evidente que la estructura es en 2D debido a que tiene anchura y altura, pero carece de profundidad.

Figura 1. Maquetación en 2D de una escena.

En la Figura 2, las tomas de la cámara frontal y posterior muestran a una tortuga y a una liebre que se están preparando para salir a correr. La tortuga y la liebre son objetos en un mundo virtual en 3D, porque podemos encontrar:

- Anchura;
- Altura;
- Profundidad.

En este caso, donde la cámara toma diferentes ángulos mostrando los objetos que le dan un sentido más real.

Figura 2. Representación 3D de una escena.

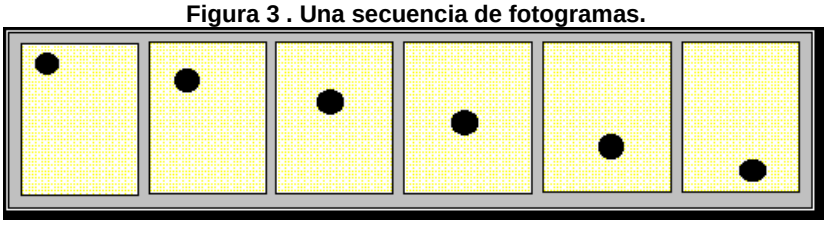
Cómo se crea una animación en Alice

En Alice, usted puede crear mundos virtuales y animaciones moviendo los objetos en el mundo, de la misma manera que los objetos se mueven en un simulador de vuelo o un juego de video. Muchas de las mismas técnicas que se utilizan en Alice para dar la ilusión de movimiento, son las que se emplean en estudios profesionales de animación, como el caso de Disney o Pixar.

La animación es una fantasía de la visión: en otras palabras, corresponde a un trabajo conjunto entre el cineasta y un grupo de artistas gráficos que unen cientos de miles de marcos fotográficos para poder recrear todas las acciones que se deban

llevar a cabo en una escena determinada. La escena es dibujada con objetos los cuales van cambiando de posición, a medida que se va desarrollando la trama de la animación. Por ejemplo, en la Figura 3 se ilustra una secuencia de fotogramas —- de izquierda a derecha —, que muestran una pelota:

Figura 3 . Una secuencia de fotogramas.



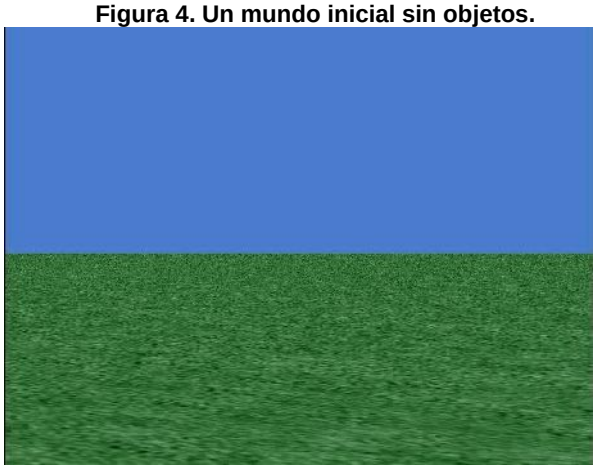
En la producción de una animación, los marcos son fotografiados en secuencia en un carrete de película o capturado por una cámara de vídeo digital. La película se ejecuta a través de un proyector y se ven en un monitor a una velocidad más rápida de lo que el ojo humano pueda detectar. Por lo tanto, el cerebro es engañado para percibir la bola cómo la pelota de la imagen anterior cae a través del aire. En lugar de crear una sola imagen, muchas imágenes se crean y se muestran en secuencia rápida, creando una ilusión de movimiento.

En este sentido, Alice crea un efecto similar en la pantalla del computador. Lo bueno para usted es que no tiene que preocuparse demasiado de la parte artística, ya que esta herramienta se va a encargar de crear la secuencia completa de fotogramas: usted sólo tendrá que preocuparse de tener claro cuáles serán las acciones o las instrucciones que se deberán ejecutar, para darle vida a sus proyectos de animación. Por lo tanto, usted pasará a ser el director de la película o del videojuego.

Un mundo virtual en Alice

En Alice, un mundo virtual comienza con una escena inicial compuesta de un cielo azul, y con el verde que representa al pasto que está en el suelo, tal como se muestra en la Figura 4.[6]

Figura 4. Un mundo inicial sin objetos.



[6] Esa es la opción que viene «*por defecto*» cuando uno se enfrenta a la tarea de hacer una animación. Por supuesto, que es posible cambiar esto desde el menú inicial del programa.

La parte divertida de usar Alice consiste en usar la imaginación para (re)crear nuevos mundos con ideas innovadoras para hacer animaciones. Los objetos se agregan con el objetivo de definir quiénes serán los personajes que van a intervenir ahí, como por ejemplo: personas, animales, naves espaciales, etc. Los desarrolladores de Alice han construido una amplia variedad de modelos 3D que podemos emplear directamente en nuestro proyecto. Para agregar algún objetivo al mundo, basta con presionar en el botón **Añadir Objetos**.

Modelos 3D de objetos

Una gran cantidad de modelos 3D se pueden emplear para ser usados como objetos en los mundos animados que construyamos. La instalación de Alice incluye una galería que contiene una selección de esta clase de modelos 3D, mientras que otros se pueden encontrar en la página web http://www.alice.org. Para ver los objetos presentes en alguna galería, basta con hacer clic en la carpeta Galería, como se muestra en la imagen que acompaña a este párrafo. Es importante destacar que Alice no es un programa para hacer dibujos en 3D, y que debido a esto existen las galerías de recursos gráficos que están disponibles tanto en el directorio local como en el repositorio de Internet.

Todos los objetos que se crean en el mundo virtual de Alice tienen una orientación tridimensional: una característica común que comparten todos ellos, es que un objeto tiene «*noción*» de lo que significa arriba o abajo (a lo largo del eje vertical) en relación a sí mismo. También, el objeto «*entiende*» el significado de **izquierda** y **derecha** (en el eje horizontal*),* y también qué dirección es **hacia delante** y **hacia atrás** (a lo largo del eje profundidad), tal como se observa en la Figura 5. Esto equivale a seis grados de orientación en el espacio tridimensional del mundo virtual. Es importante notar que **izquierda** y **derecha** son posiciones relativas con respecto a la patinadora, y no desde la perspectiva de la cámara. El lugar donde se intersectan las tres línea de colores amarillo, rojo y azul, se denomina **centro del objeto**.

Figura 5. Seis posibles direcciones para un objeto en Alice.

1.3 Consejos y Técnicas

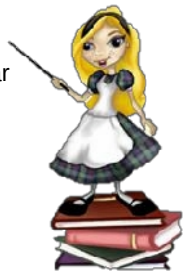
Alice ofrece una rica variedad de animaciones que se pueden hacer dentro de este entorno. Aunque no está dentro de los objetivos del curso cubrir todas las capacidades especiales de Alice, sería bueno mostrar algunas, para lograr que usted se familiarice con algunas de ellas. Al final de cada unidad, se presentarán algunos consejos y técnicas que le serán de gran ayuda para desarrollar con éxito sus proyectos finales.

Es importante destacar que estas herramientas especiales no son esenciales para aprender a programar, sino que están orientadas a brindarle una mejor presentación a sus trabajos: por ahora nos interesa lograr una buena «*puesta a punto*».

Tamaño de la ventana

Cuando se presiona el botón de reproducción, el mundo — del cual hablaremos con más detalle en las otras unidades didácticas — se pone en ejecución en una ventana que emerge para mostrar el flujo de acciones que contendrá su animación, la que se puede redimensionar en función de sus necesidades de visualización. Para esto, basta con arrastrar alguna de las esquinas de la ventana, hasta llegar al tamaño deseado, tal como aparece en la Figura 6. Después de cambiar el tamaño, Alice recuerda el último valor seleccionado. Considere además que entre más grande sea el tamaño de la ventana, el proceso de *renderización*[7] tiende a hacerse cada vez más lento. Por lo tanto: entre más pequeña sea la ventana de trabajo, mayor rapidez vamos a tener en nuestras

[7] Renderizar es un concepto que utiliza en el ámbito del diseño gráfico tridimensional, que consiste en generar una secuencia animada a partir de un conjunto de imágenes fijas o de otras secuencias de películas ya existentes, que puedan o no contener sonido.



ejecuciones.

Figura 6. Redimensionando la ventana durante la ejecución de un programa en Alice.

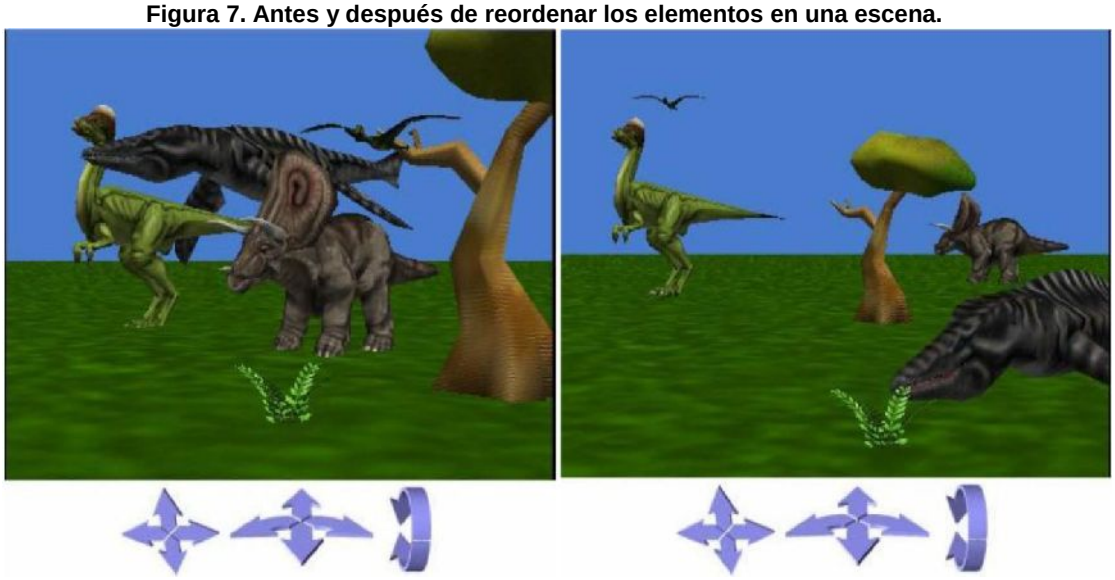
Organización de objetos

Cuando se configura un nuevo mundo, se añade un objeto a la escena, y Alice lo ubica en el centro del mundo. A medida que se requiere aumentar la cantidad de elementos, es necesario reacomodarlos con el ratón para que queden ubicados en las posiciones que deseemos.

Este procedimiento funciona igual de bien en mundos donde algunos objetos deban ser colocados relativamente cerca el uno del otro. Pero, cuando muchos de ellos se quieren ubicar dentro de la escena, su posterior ubicación puede llegar a ser complicado, en términos del reducido espacio de trabajo que se tiene para maniobrarlos. Otro elemento que le agrega cierta dificultad a esto es el hecho que los objetos agregados en algún momento se tendrán que mover, y para eso el uso de la cámara es fundamental, porque cuando uno — o algunos — se desplazan, es necesario que ésta sea capaz de seguir cada uno de sus pasos. De esta manera, la visibilidad pasa a ser un aspecto importante que debe ser resuelto.

Se recomienda que antes de organizar la estructura final de la escena, agregar cada uno de los objetos a ella, sin importar en principio el eventual desorden en que pudieran quedar. A la izquierda de la Figura 7 se observa cómo quedaría un mundo en el cual se agregan objetos por primera vez. En la derecha, cómo lucen una vez que se han ordenado según las preferencias del diseñador.

Figura 7. Antes y después de reordenar los elementos en una escena.



Moviendo subpartes de objetos

En el editor de escena, el movimiento del ratón permite mover un objeto completo. Sin embargo, existen opciones que permiten lograr que algunas subpartes de un objeto se puedan mover, tal como se muestra en la imagen adjunta. Cuando la opción de afectar a las subpartes está activada, se puede emplear el dispositivo señalador para manipular las subpartes — como cabeza, tronco, piernas, manos, etc. — que se desee, en lugar de usar a todo el objeto en sí. Debe tener habilitada la opción **afectar a las subpartes**.

Por ejemplo, podemos tener una escena en el hielo donde tengamos a un pingüino, como se ilustra en la imagen adjunta:

Esta opción es muy útil cuando algunas subpartes deban ser drásticamente reconfiguradas en función de los propósitos del diseñador. En este ejemplo, el pingüino

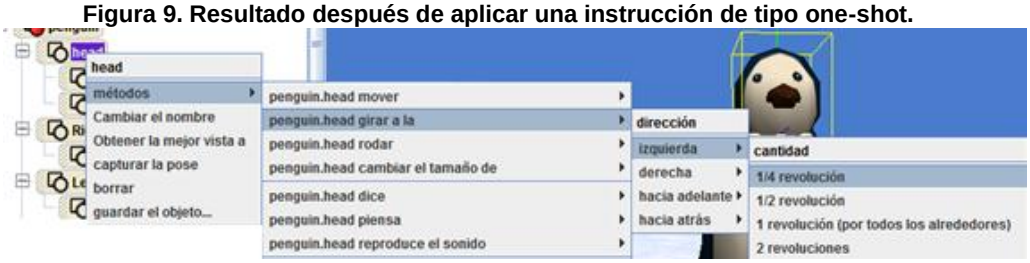
tiene que quedar en posición de emprender el vuelo. El resultado final se puede apreciar en la Figura 8:

Figura 8. Movimiento de las partes de un objeto para acomodar una escena inicial.

Movimientos de tipo «*one shot*»

Hemos visto que el control del ratón para posicionar objetos es bastante bueno para ubicarlos dentro de la escena, y que el usuario puede mover algunas de sus subpartes para adaptarlo mejor a sus necesidades. Sin embargo, existe otra variante de movimientos que vienen incorporados en el motor de Alice que simplificarían de sobremanera esta tarea, en el sentido que incorporan aquellas acciones que tienen una mayor probabilidad de ser empleados, y que sólo basta con seleccionarlos para que queden activados: este grupo de movimientos son los que se denominan de tipo **one-shot**, que sólo requieren de los valores exactos para poder ejecutarse. Siguiendo con el ejemplo del pingüino, le hemos pedido que su cabeza haga un giro igual a un cuarto de revolución hacia la izquierda, lo que se puede apreciar en la Figura 9:

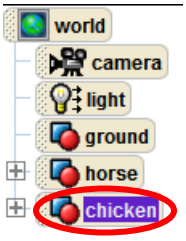
Figura 9. Resultado después de aplicar una instrucción de tipo one-shot.



El resultado que se obtiene es el que se muestra a continuación:

Vehículo

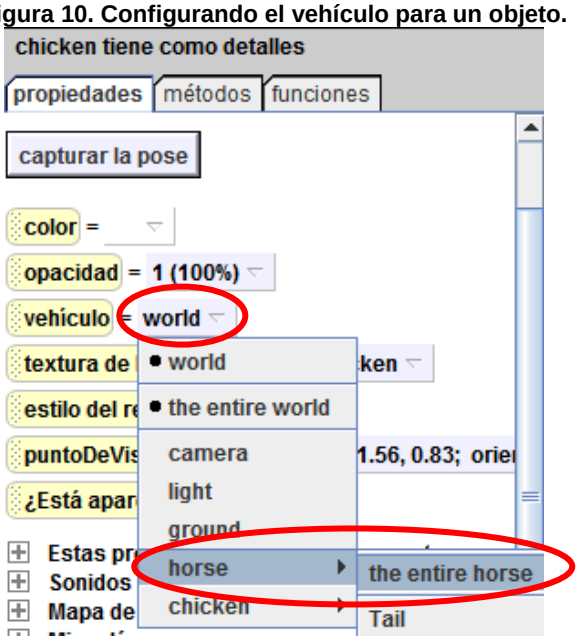
En algunas animaciones, usted puede necesitar que un objeto fuera el vehículo de otro simular el efecto de un caballo que transporta a otro animal, o de una persona que esté en un automóvil o en algún buque. Alice proporciona una propiedad especial llamado vehículo que facilita este tipo de movimiento para dos objetos. A modo de ilustración de esta propiedad, considere un circo donde un pollo va en la parte trasera de un caballo, como se observa en la imagen que acompaña a este párrafo.



Para sincronizar el movimiento, haremos que el caballo sea un **vehículo** para el pollo, de tal manera que **cuando el caballo se mueva, el pollo también**. En primer lugar, nos dirigimos al **árbol de objetos**, y seleccionamos el elemento que queramos alterar, que en este caso corresponde al pollo

A continuación, vamos a la ficha **propiedades**, y buscamos la que se llama **vehículo**, y modificamos su valor por el del caballo, tal como se muestra en la Figura 10.

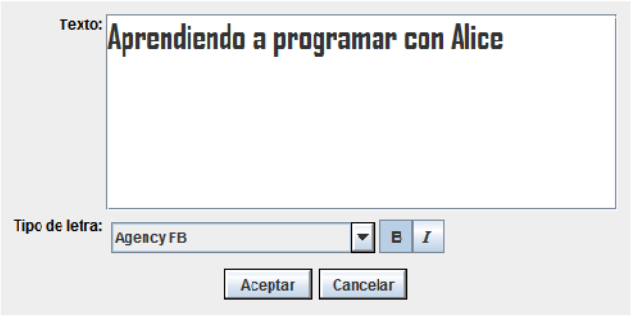
Figura 10. Configurando el vehículo para un objeto.



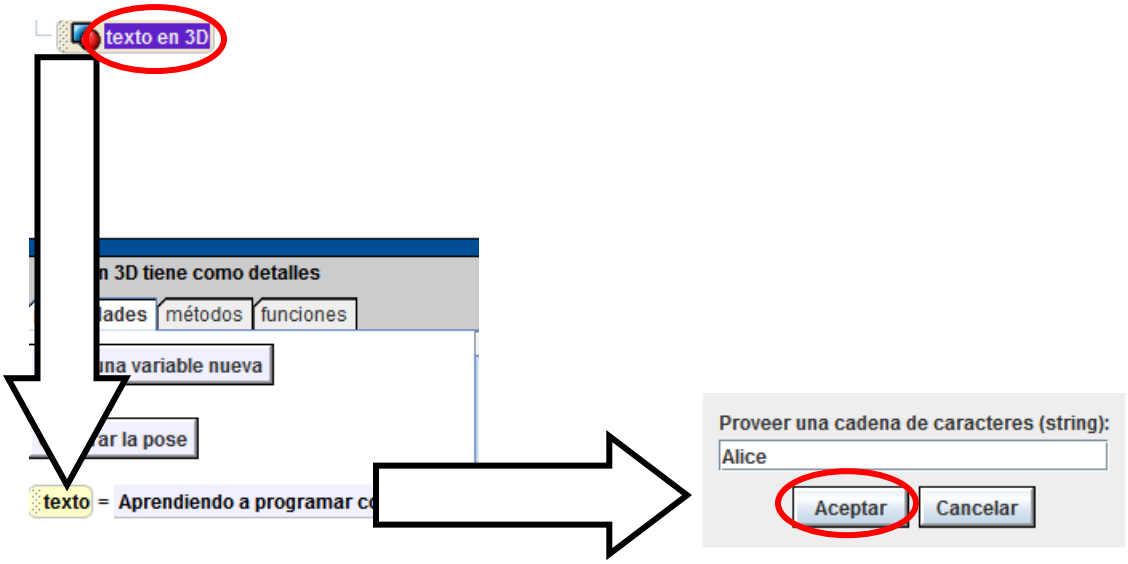
Note que cuando se requiere que **todo un objeto** sea vehículo de otro, es necesario indicarle a Alice la opción **the entire <nombre del objeto>**, que en este caso corresponde al caballo. De esta manera, si usted mueve el animal, el pollo lo hará con él.

Texto en 3D

La galería de objetos ofrece la posibilidad de agregar texto en 3D. Para ello, haga clic en **Crear un texto en 3D**, que se encuentra dentro de la galería local. Después de hacer este paso, el cuadro de diálogo que se abre le permitirá seleccionar fuente — que es el tipo de letra —, negrita, cursiva y el resto de las opciones para terminar de introducir el texto. Una vez que se ha presionado el botón Aceptar, el texto aparecerá en dentro del mundo principal:



En caso de ser necesario, usted puede modificar el contenido de ese texto sin necesidad de tener que eliminarlo y volver a introducirlo. Para eso, basa con ir a las **propiedades**, y cambiar directamente el valor que tenía.



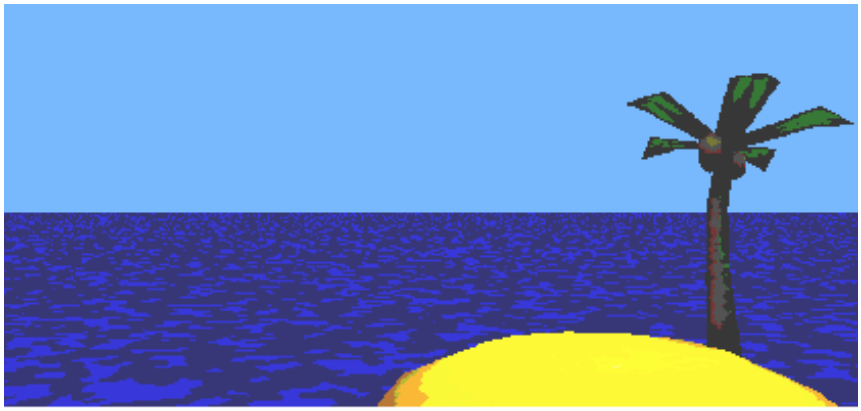
Ejercicios

El objetivo de esta primera serie de ejercicios es aprender a usar la interfaz de Alice para crear un nuevo mundo virtual. Se reitera la importante de completar el tutorial que viene adjunto en el apéndice de este texto, como una forma de llegar a buen puerto con la realización de estos ejercicios.

Consejo. Es una buena idea guardar el mundo de trabajo cada vez que sea posible, para evitar posibles pérdidas de lo que se ha construido, en caso de hacer algún desperfecto en el computador o en el sistema eléctrico. Para hacer esto, basta con ir al menú **Archivo | Guardar el mundo con este nombre**. En el cuadro emergente guardar archivo, diríjase al lugar donde quiera guardar del disco duro donde quiera guardar su trabajo, introduzca el nombre deseado, y luego presione el botón **Guardar**.

1 Mundo de isla.

Este ejercicio sirve para practicar la construcción de una escena de fondo y agregar objetos en ella. Comencemos por cambiar el color de fondo en azul. Agregue una isla objeto, y la palmera, de tal manera que su trabajo tenga una apariencia como el que se muestra a continuación:

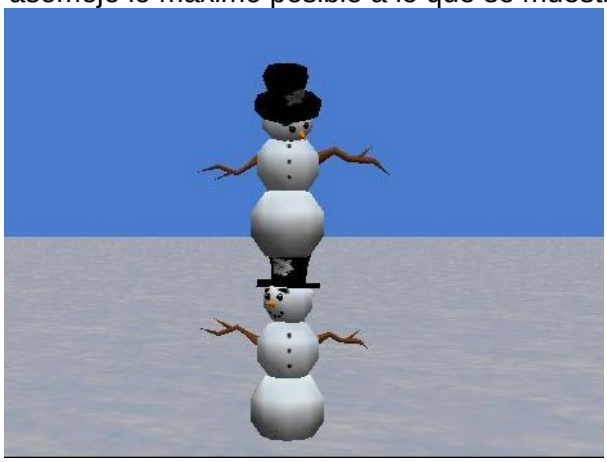


Ahora, agregue algunos peces a la esce7na, de tal manera que quede la impresión que estuviera nadando. Luego, utilice los controles de la cámara para alejar la imagen, de modo que la isla y el pez se puedan ver como en la imagen que se muestra a continuación:

Nota: Para quitar un objeto de la escena, coloque el cursor del ratón sobre el nombre del objeto en el árbol de objetos, haga clic con el botón derecho, y seleccione la opción **eliminar** que aparece dentro del menú contextual. Otra forma útil para eliminar un objeto que se acaba de agregar a un mundo es hacer clic en el botón **deshacer**, que se puede emplear tantas veces como sea necesario, en caso que hayan acciones de las cuales el usuario se haya arrepentido de incluir en su proyecto.

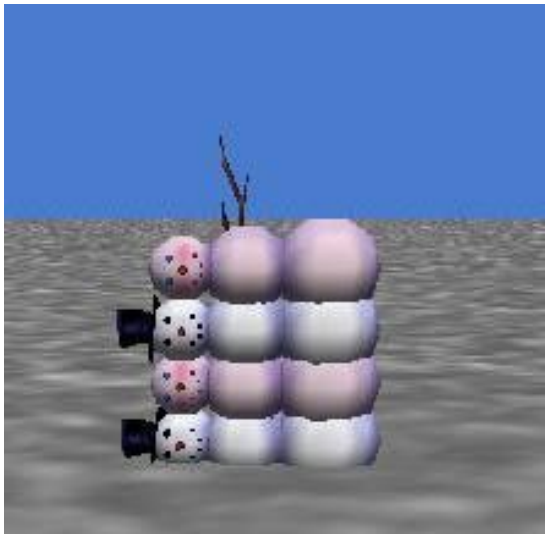
2 Invierno.

Abra un nuevo mundo de Alice y añada dos objetos muñeco de nieve a la escena, de tal manera que su trabajo se asemeje lo máximo posible a lo que se muestra a continuación:



3 Pila de nieve.

Construya un «*muro*» con dos muñecos de nieve y dos muñecas de nieve, inclinados a los largo, uno encima del otro, tal como se ilustra en la figura adjunta (note que ellos van alternados):

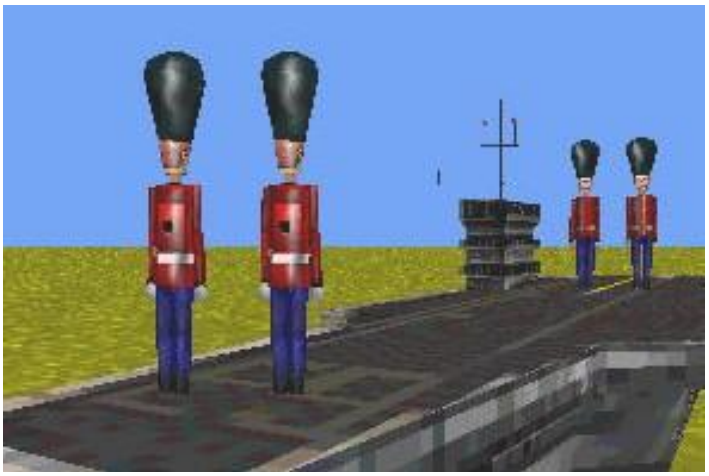


4 Fiesta del té.

En homenaje a Lewis Carroll — que fue el autor del cuento «*Alicia en el país de las maravillas*»—, recree una Fiesta del Té donde participen Alicia y el Conejo blanco. Además de estos personajes, la escena deberá incluir una mesa y dos sillas pintadas, una tetera, crema, tostadora y mini-galletas de trigo, de color amarillo.

5 Soldados de juguete.

Cree una escena que contenga un portaaviones y cuatro soldados de juguete para un nuevo mundo, de tal manera que estos últimos aparezcan alineados, como si fueran a participar de alguna ceremonia formal. Lo que se quiere lograr aparece reflejado en esta imagen:



Segunda Parte

Diseño e Implementación de Programas

En esta segunda parte comenzamos una introducción hacia el mundo de la programación. Escribir un programa para animar objetos 3D en un mundo virtual corresponde a detallar todas las acciones que estos elementos puedan hacer sobre el escenario de trabajo. Desde un punto de vista práctico, la escritura de un programa conduce a solucionar problemas que se pueden presentar en la vida cotidiana: en primer lugar, se nos presenta la descripción de alguna situación que haya que resolver, para luego determinar cuáles serán los pasos que haya que hacer para brindar alguna solución. De ahí, viene la etapa de resolución, que en nuestro caso va a corresponder a la etapa de escritura del programa usando el **editor de código de Alice**.

Como no es la idea seguir un enfoque tradicional para enseñar a programar, de momento diremos que la escritura de un programa consistirá en estructurar el esquema de una animación que se deberá llevar a cabo en un determinado *escenario de trabajo*, que representará en buenas cuentas la especificación de los requerimientos que deberá seguir la solución que programemos.

De esta manera, el programa que se haya construido va a ser equivalente a un guión de trabajo que deberá corresponder a la implementación del escenario; esto es lo que en Ciencias de la Computación se conoce como **código fuente**, y las animaciones que le darán vida a este libreto corresponderán a cada una de las facetas que componga el guión propiamente tal.

En los primeros párrafos se tratará el tema de los escenarios y guiones gráficos, que formarán parte de la metodología que emplearemos para la elaboración de programas. Los guiones gráficos fueron elegidos porque son la herramienta de diseño utilizado por los animadores profesionales que participan en estudios cinematográficos como Pixar o DreamWorks. Su uso se justifica para que usted se sienta como todo un director de cine ☺

Los guiones gráficos fueron elegidos porque representan una estructura paso a paso de cómo se debe dar el flujo de acciones del programa. Las líneas de texto de un guión gráfico son similares al *pseudocódigo*, que corresponde al esqueleto de todo lo que tiene que hacer el producto que terminemos fabricando.

Luego, se exponen los fundamentos básicos de la creación de un programa sencillo en Alice. Una de las características más notables que tiene esta herramienta es que los diferentes bloques de programación se «*escriben*» usando una funcionalidad

de la interacción hombre-máquina conocida en Ciencias de la Computación como *arrastrar y soltar* — o «*drag and drop*» en inglés.

Lo bueno de esto es que nos podemos centrar exclusivamente en los pasos de la solución, porque Alice elimina de forma automática cualquier problema de sintaxis[8] que pudiera haber, junto con la generación automática de los fotogramas que componen la animación final. Es común que en una animación, algunas acciones deban ser ejecutadas en secuencia, y otras en forma simultánea, lo que significa que el código que se vaya a realizar debe dar cuenta de este tipo de situaciones.

1.4 Escenarios y Diseño de Historias

La creación de un programa en computador que anima objetos en un mundo virtual es un proceso que contempla dos pasos:

- El primero corresponde al **diseño**, que es la planificación de todo lo que hay que hacer **En esta parte sólo abordaremos el diseño.**
- El segundo es la **implementación**, o escritura del programa de animación.

El diseño es una parte importante para llegar a construir programas de animación, ya que requiere dominar una habilidad que se denomina *planificar*. Los programas que se muestran en los ejemplos mostrados a continuación son bastante claros e ilustrativos, que pueden ayudar a desarrollar — o pulir en muchos casos — la habilidad de planificación antes mencionada, cuyo trabajo es absolutamente necesario, sobre todo cuando haya que elaborar programas de mayor dificultad y complejidad. Un diseño consta de dos componentes:

- **Escenario.** Es un planteamiento del problema que describe en términos globales el planteamiento del problema, y lo que se espera que el programa sea capaz de hacer. Sin ir más lejos, las caricaturas animadas comienzan su desarrollo mediante la creación de un escenario preparado por escritores profesionales.

- **Guión gráfico.** Corresponde a la secuencia de escenas por las que deberá transitar la animación que estemos construyendo. Cada escena deberá contener:
  - Número que la identifique del resto.
  - Descripción gráfica (de ahí el nombre de guión gráfico) de lo que debiera suceder.
  - Texto que digan los personajes u **objetos** que participen de ella.
  - Sonido que debiera acompañar.

Los elementos propios del guión gráfico se discutirán dentro de poco. Por el momento, seguiremos hablando del escenario.

A veces, un escenario se conoce como «*la historia*» que se debe relatar, la lección que debe enseñar, o la simulación por computador que se va a hacer para hacer alguna demostración práctica de algún fenómeno. De esta manera, una animación se puede descomponer en muchos escenarios, en cada uno de los cuales será necesario contar con un guión gráfico para describir la secuencia de las escenas que lo compongan.

Tal como se indicó en la parte anterior, un mundo nuevo en Alice consta de un cielo azul y de una hierba verde que simula el suelo. Al crear una escena inicial, los

[8] En este contexto, la sintaxis se refiere a las reglas que debe seguir la escritura de algún programa, tanto en la parte de estructura de las sentencias como de los signos de puntuación que deban emplearse.

objetos que participen de ella se agregan y colocan en el mundo. Sin embargo, antes que los objetos se agreguen y se ubiquen en la escena, es necesario responder tres preguntas esenciales:

1. **¿Cuál es la historia que se va a contar?**
2. **¿Qué objetos se necesitan? Los objetos incluyen personajes que desempeñan funciones directivas en la historia, así como otros objetos de fondo, como islas, montañas y lagos (¿se parece en algo a lo que aparece en la primera tanda de ejercicios dados al final de la primera parte?).**
3. **¿Cuáles son las acciones que se tienen que llevar a cabo?**

Un escenario ofrece todos los detalles necesarios para la creación de la escena inicial y luego la planificación el guión gráfico para la animación.

Ejemplo de escenario

Supongamos que usted se encuentra en el hemisferio norte, en plena época de Navidad. Un día, se queda contemplando el calor que sale de la chimenea que se ha instalado en el hostal donde está alojando. De pronto, se queda profundamente dormido. Esto es lo que soñó:

> *Varios muñecos de nieve están al aire libre, en un paisaje nevado. Un DJ está tocando la canción «let it snow» de Frank Sinatra. Un muñeco está tratando de conocer a una muñeca de nieve, que a su vez está conversando con un grupo de amigas. Él le «hace ojos» a ella para tratar de conseguir su atención, pero por desgracia, ella no está interesada en bailar con él. De hecho, lo ignora con frialdad, y sigue hablando con sus amigos.*

En este escenario, tenemos respuestas a las preguntas que se plantean en el párrafo anterior. En primer lugar, relata la triste historia de un fallido intento de coqueteo de un muñeco de nieve hacia una muñeca en un baile que se desarrolla en un paisaje nevado, al ritmo de la canción «*let it snow*». Los objetos son muñecos o monos de nieve, y el escenario de fondo representa una escena de invierno típica del hemisferio norte. Las acciones corresponden al muñeco haciéndole «*ojitos*» a la muñeca de nieve, los muñecos bailando, y la muñeca aludida haciendo un gesto de indiferencia hacia su pretendiente.

Guión gráfico visual

Un *guión gráfico visual* se compone de un escenario que muestra las secuencias de acciones por medio de escenas, y que indica lo que debe ocurrir en cada transición. Cada dibujo representa un *estado* de la animación, que corresponde a una fotografía o a una instantánea de la escena, donde cada una de ellas abarca a objetos que están en determinadas posiciones, colores, tamaños, y poses. De esta manera, cuando cualquiera de estas características se modifica, se producen cambios en la animación, lo que produce una nueva transición, y por ende, una nueva escena. Cada una de las instantáneas se numera en secuencia y se etiqueta con la información necesaria.

Plantilla genérica para un guión gráfico

Una plantilla genérica escena se muestra en la Tabla 1. Cada instantánea es etiquetada con un **número de escena**, y contiene una **ilustración** de todos los objetos que participan en ella donde los objetos en la escena. La **descripción** indica la acción que se debe producir, junto a los objetos que tomarán parte de ella. El campo de sonido se utiliza para dar a conocer cuáles son los archivos de audio que se van a requerir para darle vida a la escena. El campo **texto** indica cuál será el diálogo que se deba producir, en caso de ser necesario.

Tabla 1. Plantilla para construir alguna escena en forma gráfica.

Número de escena: En este espacio en blanco se inserta alguna imagen que permita describir con claridad lo que se quiera representar en la escena. **Descripción:** **Sonido:** **Texto:**

Ejemplo de guión gráfico

Para ilustrar la creación de un guión gráfico, el escenario que se muestra a continuación describe la escena en la cual el mono o muñeco de nieve le guiñe el ojo a la muñeca, mientras suena de fondo la canción «*let it snow*». En la Tabla 2, los círculos fueron creados para representar a los monos de nieve, mientras que las líneas diagonales pretenden simbolizar a las montañas en la distancia. Las líneas onduladas grises se ocupan para representar la superficie cubierta de nieve. Lo importante de este ejemplo es hacer ver que no es necesario ser un experto dibujante para poder dar a entender nuestras ideas o conceptos mentales ☺

Tabla 2. Una representación gráfica inicial para el relato de la fiesta de la nieve.

Número de escena. 2

Descripción. Los ojos del muñeco de nieve se mueven hacia arriba y hacia abajo.

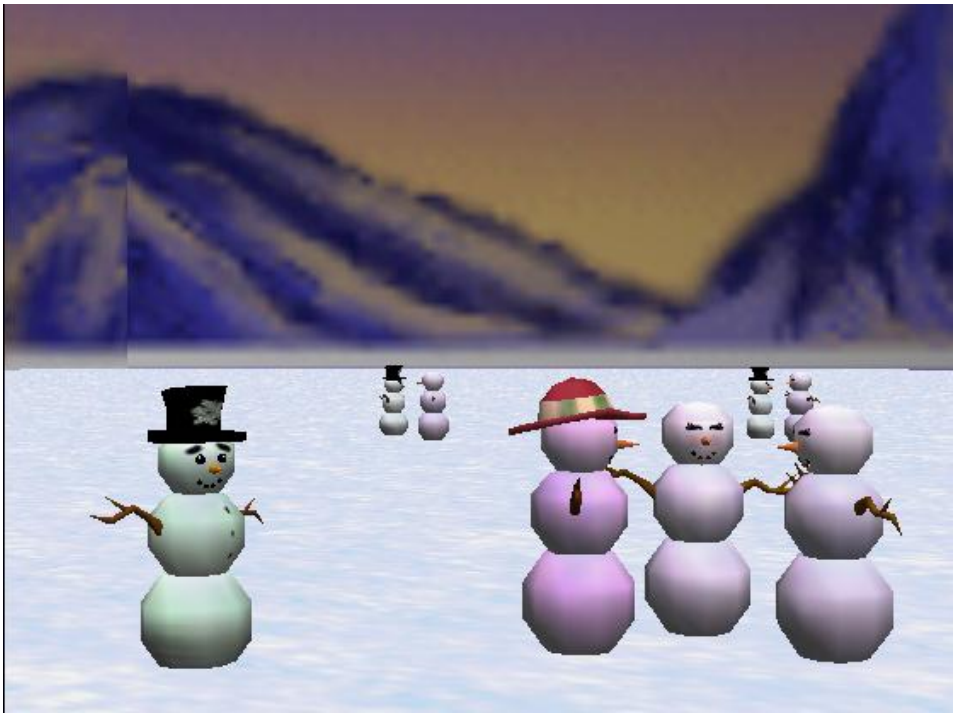
Sonido. Let it snow, interpretada por Frank Sinatra.

Texto. (no hay)

El mismo ejemplo, pero usando otra técnica

Una segunda técnica que podemos emplear consiste en la utilización del editor de escena de Alice para agregar los objetos que vayan a tomar parte de ella, junto a la posición que deban adoptar. Como cada escena debe ser creada, lo que haremos es ir generando sucesivas capturas de pantalla que permitan reflejar los distintos cambios de estado por los cuales vaya pasando el flujo de acciones de nuestra historia o aplicación. Los bocetos que se muestran a continuación reflejan lo que hemos comentado hasta el momento.

Número de escena: 1

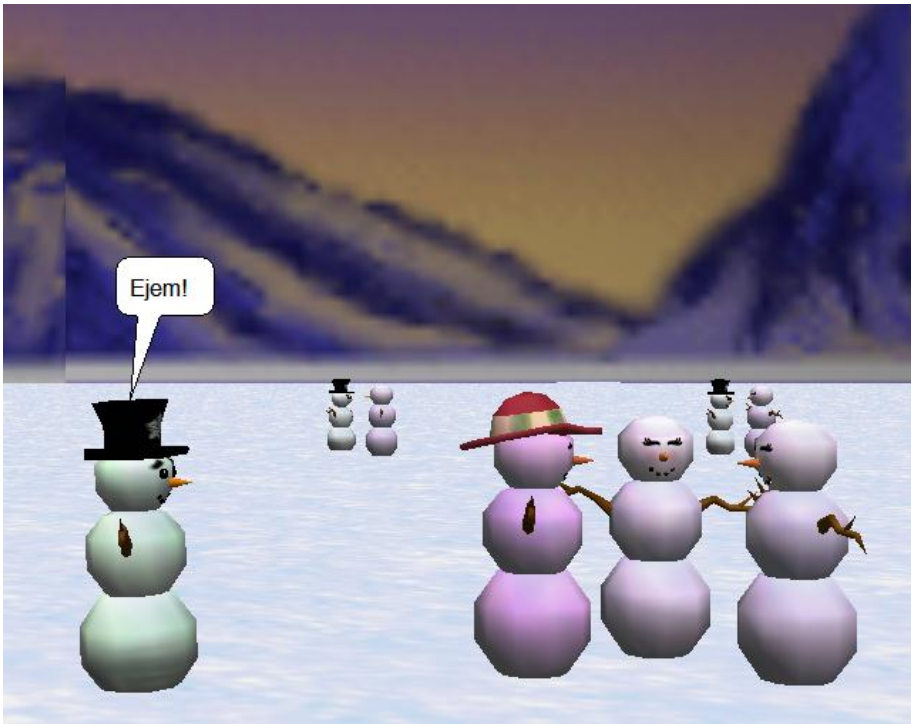


Descripción: Esta corresponde a la escena inicial, en la cual el muñeco de nieve está interesado en conocer a la muñeca que está más a la izquierda.

Sonido: Let it snow, interpretada por Frank Sinatra.

Texto: (no hay)

Número de escena. 2

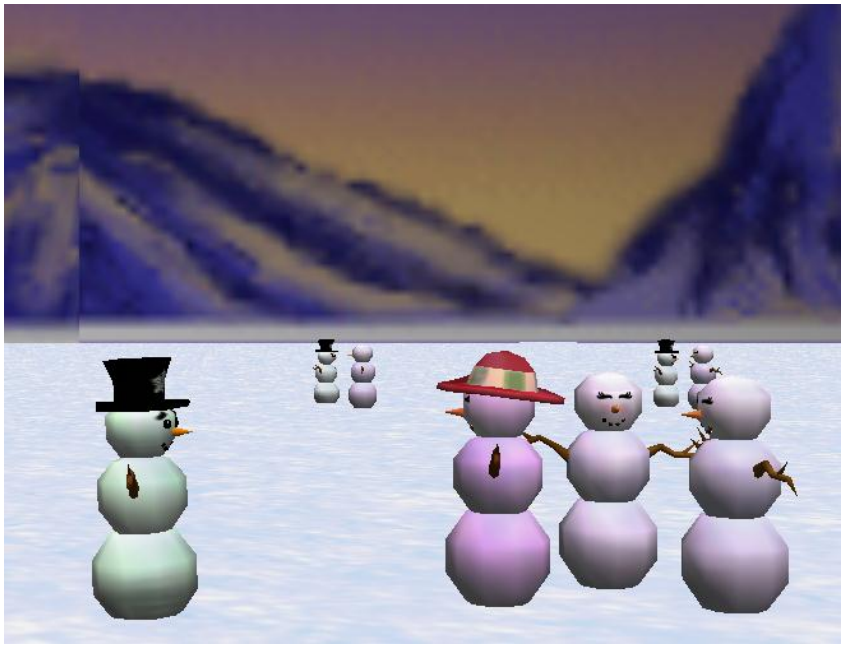


Descripción. El muñeco de nieve trata de captar la atención de la muñeca de nieve.

Sonido. Let it snow, interpretada por Frank Sinatra.

Texto. El muñeco dice "Ejem!"

Número de escena. 3



Descripción. La muñeca de nieve se da vuelta para ver quién la está cortejando.

Sonido. Let it snow, interpretada por Frank Sinatra.

Texto. (no hay)

Guión textual

Un guión textual corresponde a uno gráfico, pero sin imágenes. Esta clase de herramientas resulta ser de gran utilidad en la práctica, ya que permite ahorrar gran cantidad de tiempo por concepto de generación de dibujos computacionales que luego tengan que ser traspasados a alguna tabla. Un posible[9] guión textual para el relato de los muñecos de nieve se muestra en la Tabla 3. Un punto importante que debe ser mencionado es que **un guión textual puede abarcar más de una escena de un guión visual**, permitiéndonos con ello tener en pocos párrafos una idea global de lo que se quiere hacer.

Tabla 3. Guión textual para la interacción entre los muñecos de nieve.

Hacer en orden:
1. El muñeco de nieve apunta a la muñeca de nieve
2. Hacer los siguientes pasos *simultáneamente*:

[9] Es bueno recalcar que los problemas y ejemplos planteados en este apunte deben estar sujetos a una única respuesta correcta. Conviene explorar más de una alternativa para poder evaluar entre muchas posibles soluciones que sea válidas.

```
            a) El muñeco de nieve llama a la muñeca de nieve.
            b) El muñeco de nieve le guiñe el ojo a la muñeca de
      nieve
      3. La muñeca de nieva voltea la cabeza para saber quién la
está llamando … y así sucesivamente
```

Las líneas de texto proporcionan una lista ordenada de las acciones que se llevarán a cabo en la animación. Tenga en cuenta que el texto tiene dos líneas que se muestran en cursiva: ellas permiten organizar las acciones, en cuanto a si deban ser ejecutadas en forma secuencial o simultánea. La sangría se utiliza para hacer que el guión sea fácil de leer y de entender.

En la terminología que se emplea en las Ciencias de la Computación, un guión textual recibe el nombre de **algoritmo**, que corresponde a una lista de acciones que se deben llevar a cabo para realizar una tarea o resolver un problema. Recuerde que aún estamos en la etapa de diseño; y que luego revisaremos la de implementación.

Consejo. Una vez que un guión haya sido elaborado, siempre es bueno leerlo con el objetivo de determinar cuáles podrían ser los cambios que podamos introducir para hacer que el trabajo realizado mejore en términos cualitativos. Una forma de evaluar la calidad de lo que hemos hecho, es responder a las siguientes preguntas:

1. ¿El flujo de acciones que transcurre escena tras escena, es tal como se menciona en la historia?
2. ¿Es necesario agregar alguna transición para pasar de una escena a la siguiente?
3. ¿Hay alguna parte esencial de la historia que estemos pasando por alto?

1.5 Un primer programa

En la sección anterior, hemos aprendido a diseñar una animación con un escenario y un guión gráfico. Ahora, estamos listos para ver cómo puede llegar a ser escrito, de tal manera que revisemos cómo se comporta al momento de ejecutarse en nuestra máquina. En este sentido, diremos que una **aplicación** corresponde al producto de la escritura del programa, y que debe ser congruente con el **diseño** que se haya realizado, y que se materializa por medio de los guiones que hemos construido: gráfico o textual.

¿Qué es un programa?

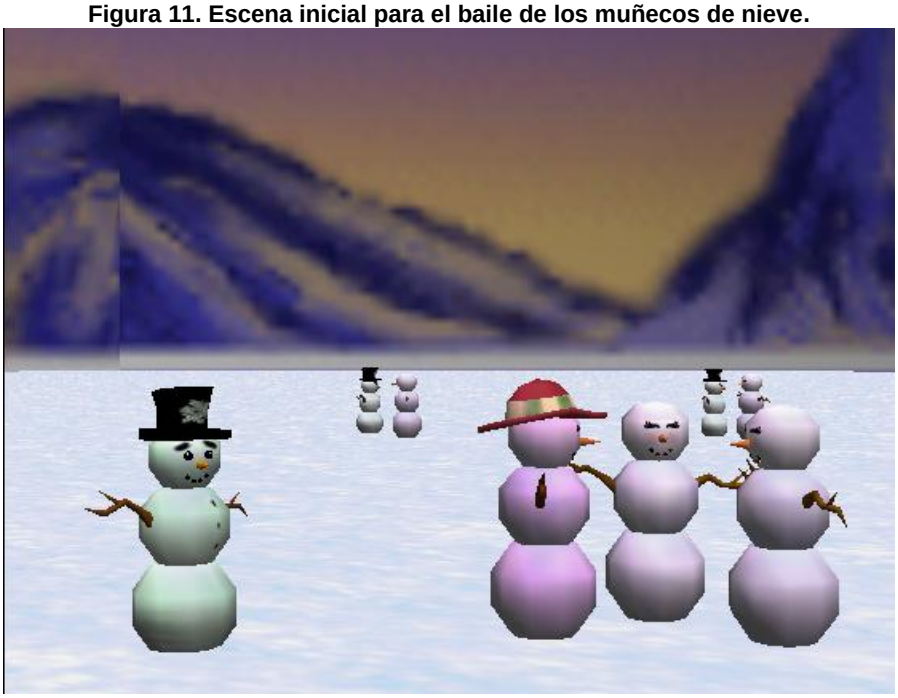
Un *programa* es una lista de acciones o de instrucciones que debe efectuar el computador para que entregue los resultados que deseemos. En el contexto de Alice, pensaremos que un programa corresponde al guión de alguna obra teatral.

Un guión teatral nos cuenta una historia, describe las acciones que se toman, y las palabras que deben pronunciar los actores en algún momento en el escenario. De manera similar, un programa o aplicación en Alice establece los personajes — u objetos —, acciones, texto y sonido que deben combinarse de manera apropiada dentro de los mundos virtuales que construyamos.

Construcción de una escena inicial

Empecemos con la creación de una escena a partir del ejemplo que revisamos en la sección anterior, donde el muñeco de nieve estaba interesado en bailar con la muñeca de nieve. Él le hace ojos, pero ella no se muestra muy interesada en danzar con él, lo que hace que se aleje de su presencia. El primer paso consiste en crear la escena inicial. Entonces, en un mundo nuevo agregaremos los muñecos y muñecas de nieve, junto con el paisaje invernal de fondo y el suelo de color blanco. En la Figura 11 se muestra una recreación de esta escena.

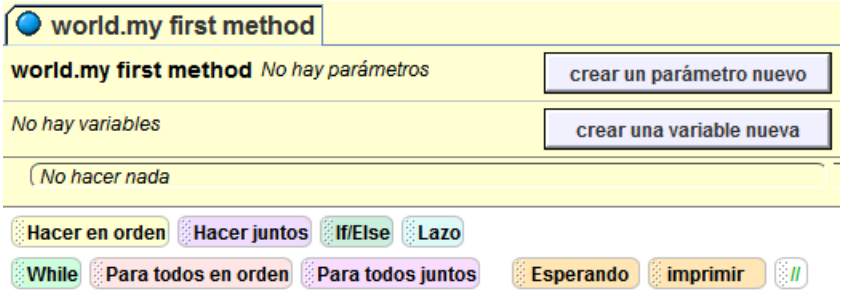
Figura 11. Escena inicial para el baile de los muñecos de nieve.



Una vez que hemos confeccionado la escena inicial, las instrucciones que finalmente le darán vida al programa deben trasladarse hasta el panel de edición de código, que corresponde al rectángulo amarillo que aparece en la parte inferior derecha de la ventana principal de Alice, tal como se muestra en la Figura 12.

Cuando el programa — o una parte — se haya escrito, será necesario ejecutar la animación tantas veces sea necesario hasta que el resultado obtenido nos deje satisfechos. Para ello, es importante tener presente que siempre es recomendable guardar lo que hayamos hecho en algún archivo, de tal manera que si deseamos proseguir nuestro trabajo en algún momento posterior, podamos abrir y ejecutar lo obrado hasta el momento.

Figura 12. El editor del código del programa se encuentra en el extremo inferior de la pantalla



Método «*my fisrt method*»

Tal como se ve en la Figura 12, la ficha para el área de edición tiene la etiqueta **World.my first method**. Es preciso señalar en este punto[10] que un método corresponde a un segmento de código del programa que realiza una tarea específica. Alice utiliza automáticamente el nombre **World.my first method** para referirse al primer conjunto de tareas que se espera que haga la aplicación, lo que no significa que usted no pueda renombrarlo como le parezca más apropiado.

De momento vamos a trabajar con la etiqueta **World.my first method**. En este ejemplo tan particular de la fiesta de la nieve, podemos escribir todas las acciones dentro de **World.my first method**. También es bueno destacar que cuando se presiona el botón ejecutar, el método **World.my first method será el primero que se ponga en marcha, a no ser que se indique lo contrario.**

Cuáles son las instrucciones requeridas

Como se dijo en párrafos anteriores, el editor de código es la herramienta de Alice en la cual se van a posicionar las instrucciones que posibilitarán la puesta en marcha la animación del baile de los muñecos de nieve. Ahora bien, el guión que se haya diseñado — ya sea gráfico o `textual` – es lo que nos ayudará a determinar cuáles serán las instrucciones que se vayan a utilizar, y en qué orden deban ser invocadas, ya que estos mini libretos ayudan bastante a planificar las acciones que se deban llevar a cabo. A continuación, se ofrece una versión completa del guión textual del ejemplo animado, que se presenta en la Tabla 4:

Tabla 4 . Guión textual tentativo para el problema de la fiesta de la nieve.

```
Hacer en orden:
      1. El hombre de nieve mira a la muñeca de nieve.
      Hacer juntos:
            a) El hombre de nieve llama a la muñeca de nieve
            b) El hombre de nieve mira embobado a la muñeca
      de nieve
```

[10] En la próxima unidad abordaremos los temas relacionados con métodos con mayor profundidad. Por ahora, quedémonos con las ideas que se han expuesto hasta el momento.

```
        3. La muñeca de nieve se voltea para ver quién la ha
    estado llamando
        Hacer juntos:
            a) La muñeca de nieve se sonroja, haciendo que
        su cabeza quede de color rojo.
            b) La muñeca de nieve se voltea y sigue hablando
        con sus amigas
        4. El efecto del sonrojamiento ha terminado, y la
    cabeza de la muñeca de nieve queda de color blanco.
```

El programa probablemente debería consistir de varias instrucciones que gatillen acciones como *mover* o girar. Parece razonable comenzar haciendo que el hombre de nieve mire a la muñeca de nieve. Una instrucción de *apuntar hacia* se puede solicitar para simular que el muñeco estuviera mirando a la mujer de nieve.

Luego, una instrucción *decir* puede ser usada para que el hombre se dirija a la mujer. Es importante recalcar que todas las acciones se deben hacer en una secuencia específica, para que el relato global tenga un sentido narrativo claro y coherente.

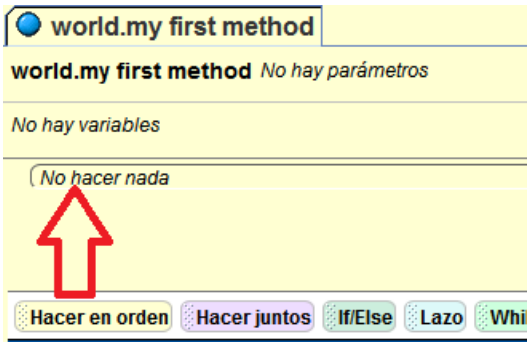
Acciones secuenciales y simultáneas

Las acciones **secuenciales** son todas aquellas que se ejecutan **una después de la otra**, mientras que las otras exigen que todo lo que encierren se materialice de forma **paralela**. Resulta evidente por ejemplo que el muñeco deba quedar mirando fijamente a la mujer antes de acercarse a conversar con ella: a lo mejor para nosotros resulta muy claro que así tenga que ser, pero a Alice hay que *hacérselo ver* de manera explícita. Lo mismo ocurre cuando estamos en la escena en la cual el muñeco de nieve la llama y al mismo tiempo la queda mirando con expresión de embobamiento.

Hacer en orden

La forma de indicarle a Alice que las instrucciones deban ir en un orden estrictamente secuencial es usando el comando **Hacer en orden** que luego hay que arrastrar hacia el editor de código, tal como se muestra en la Figura 13.

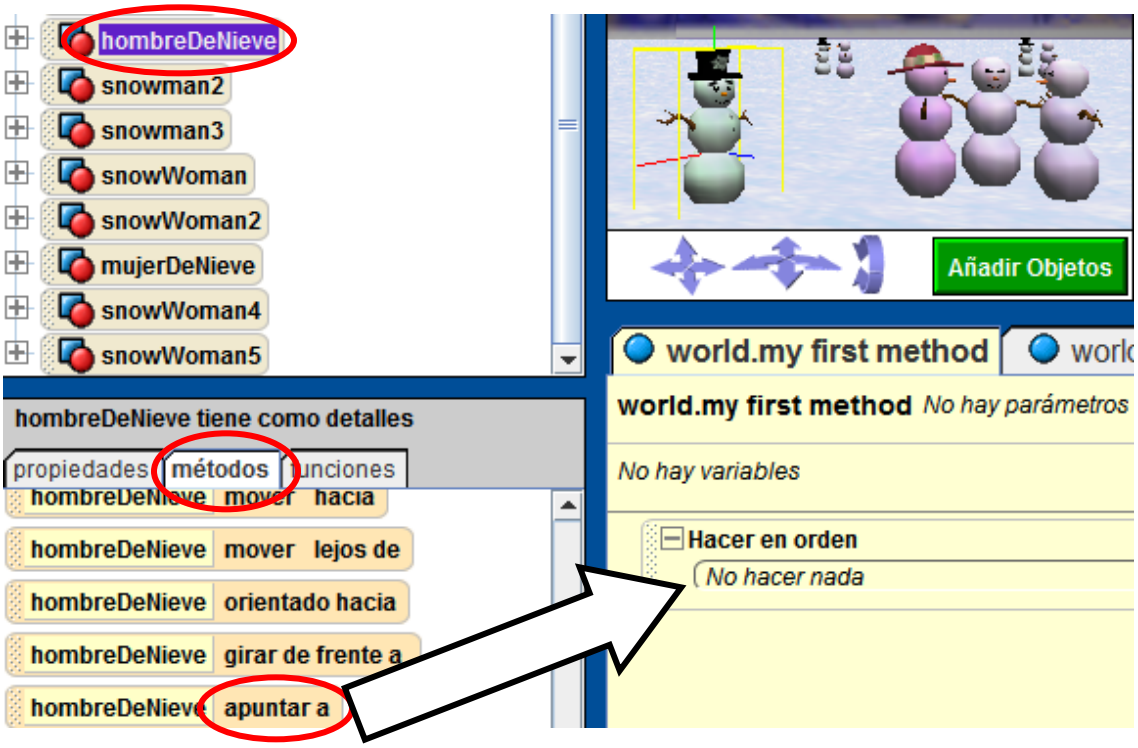
Figura 13. Agregando un bloque *Hacer en orden* al editor de código.



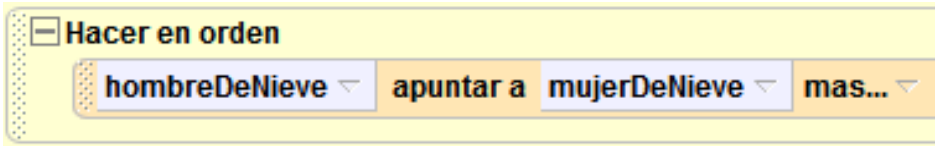
Una vez que se ha arrastrado el control *hacer en orden,* las instrucciones que vengan a continuación deberán ser colocadas en el editor de código. La primera de

ellas indica que el muñeco de nieve debe quedar mirando a la mujer de nieve. Para esto, el muñeco de nieve se selecciona en el árbol de objetos, que está ubicado en el extremo superior izquierdo de la ventana de Alice. A continuación, en la ficha «*métodos*», se arrastra el comando «*apuntar a*», y se marca a la muñeca de nieve que será el objetivo. Todo esto se muestra en la Figura 14.

Figura 14. Agregando una instrucción de un objeto hacia el editor de código.



Como se había comentado en el párrafo anterior, la instrucción *apuntar a* requiere de un **parámetro**[11], que en este caso corresponde a saber cuál será el objeto del mundo que debiera ser apuntado por el mono de nieve. En este ejemplo, la muñeca de nieve es la que tiene que ser seleccionada, tal y como se muestra a continuación:



[11] El concepto de parámetro lo trataremos con más detención en las segunda unidad. De forma sintética, corresponde a un elemento de información que debe suministrarse a Alice para ejecutar alguna acción que lo requiera.

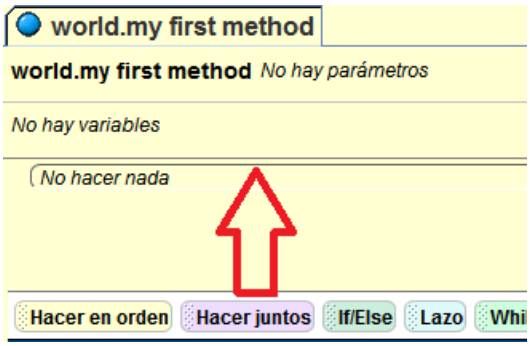
Hacer juntos

El siguiente paso en el guión gráfico requiere que dos cosas se ejecuten a la vez: el muñeco diciendo «*Ejem!*» a la muñeca de nieve, y que él suba y baje sus ojos. Luego, el bloque **Hacer juntos** se arrastra hacia adentro de *hacer en orden*, tal como se muestra en la Figura 15.

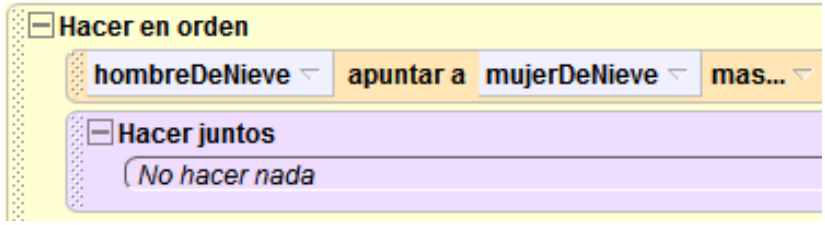
Es importante que usted tenga presente que el hecho de **anidar**[12] el bloque *hacer juntos* dentro de *hacer en orden* es con el fin de hacer esta animación lo más fidedigna posible con el esquema de trabajo. Como se indicaba en un párrafo anterior, recuerde que los problemas de programación no necesariamente admiten una única solución válida.

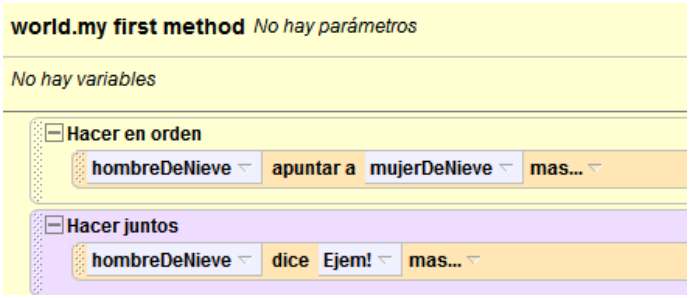
También es bueno tener en cuenta que estos dos bloques de programación pueden trabajar por separado, y se pueden combinar de muchas maneras diferentes.

Figura 15. Insertando un bloque *Hacer juntos* dentro de un bloque *Hacer en orden.*



El resultado debe aparecer tal como se ilustra a continuación. Ahora, estamos en condiciones de arrastrar hacia el bloque **hacer juntos** las instrucciones necesarias para que el muñeco diga «*Ejem!*», y para que «*coquetee*» con la mujer de nieve subiendo y bajando los ojos.



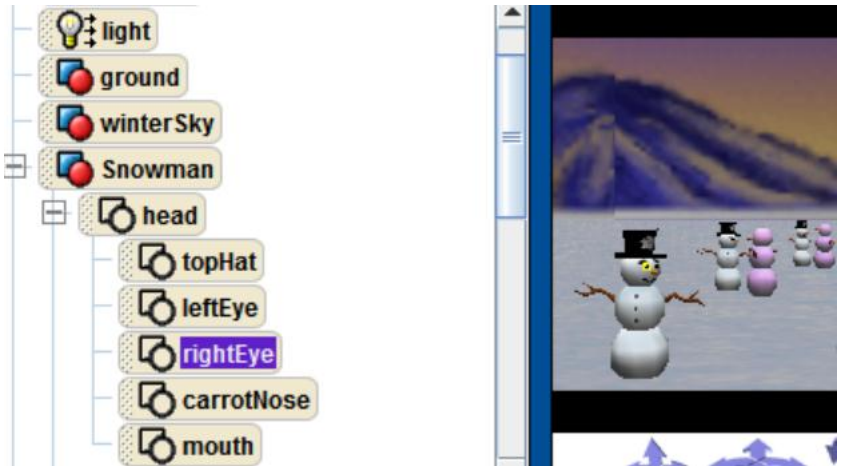


La acción de hablar se implementa arrastrando el comando que se llama «*dice*» hacia el editor de código, y escribiendo la cadena de texto que se quiere decir. En este caso, hay que pedir que sea «*Ejem!*».

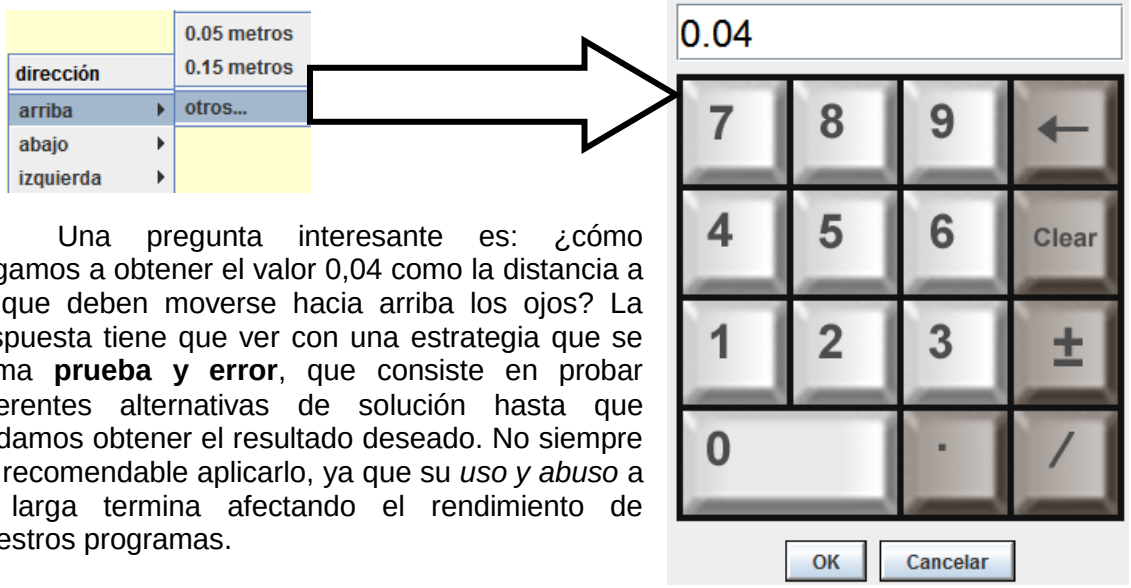
[12] Anidar significa incluir alguna estructura de programación dentro de otra. Este concepto lo estudiaremos con más detalle en las siguientes unidades.

Las instrucciones para que el muñeco mueva los ojos son un poco más complicadas, en el sentido que no se generan de manera directa. Para esto, vamos al árbol de objetos, con el fin de acceder a las piezas que componen el objeto muñeco de nieve. A continuación, haga clic con el ratón sobre la pestaña asociada con la cabeza del personaje, para poder acceder a los ojos. Todo esto se muestra en la Figura 16.

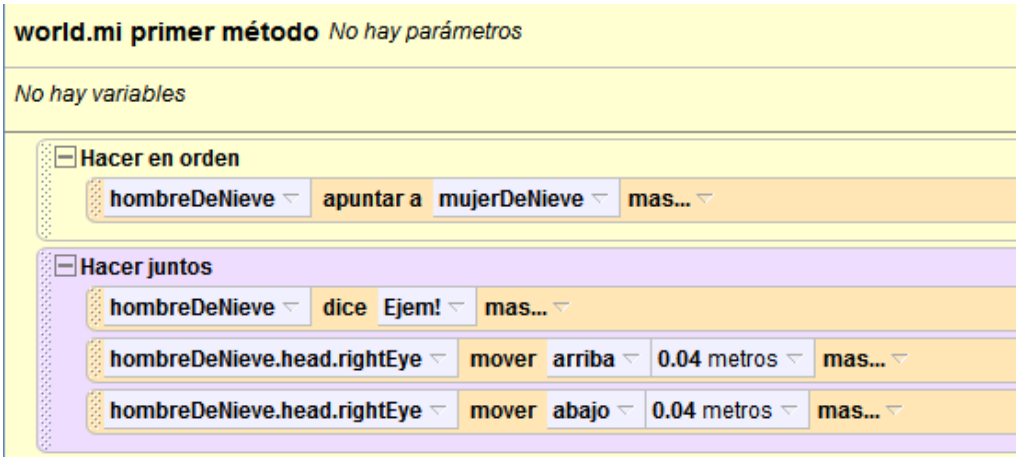
Figura 16. Accediendo al ojo derecho del muñeco de nieve



Ahora es posible arrastrar las instrucciones hacia el editor para mover el ojo derecho del hombre de nieve 0,04 metros hacia arriba, y 0,04 metros hacia abajo, lo que genera el código que se muestra a continuación. Tenga en cuenta que la dirección del movimiento es necesaria, así como la distancia, que se introduce seleccionando la opción **otros**, y luego escribiendo el valor 0,04 en la calculadora.



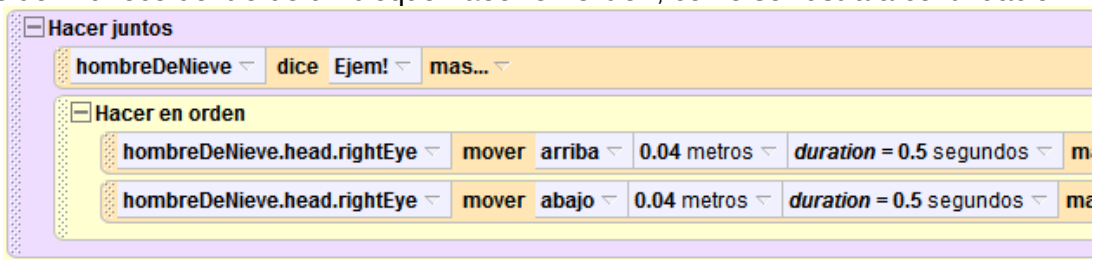
Una pregunta interesante es: ¿cómo llegamos a obtener el valor 0,04 como la distancia a la que deben moverse hacia arriba los ojos? La respuesta tiene que ver con una estrategia que se llama **prueba y error**, que consiste en probar diferentes alternativas de solución hasta que podamos obtener el resultado deseado. No siempre es recomendable aplicarlo, ya que su *uso y abuso* a la larga termina afectando el rendimiento de nuestros programas.



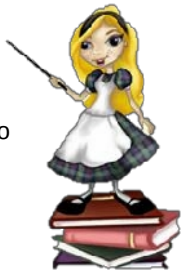
Probar y depurar

Cuando se han escrito varias líneas de código, es necesario probar lo que hasta el momento hemos realizado. Para esto, hay que presionar sobre el botón **Ejecutar**. Luego de esto, el muñeco apunta hacia la mujer de nieve, y él le dice «*Ejem!*». Sin embargo, su ojo derecho no se mueve hacia arriba y hacia abajo.[13]

Esto significa que se ha producido un error en el programa, que en la jerga computacional se conocen con el término anglosajón **bug**. El problema que se ha producido es que al muñeco se le ha pedido que ejecute instrucciones para mover los ojos hacia arriba y hacia abajo *al mismo tiempo*. Lo que ha ocurrido es que estas instrucciones se **cancelan mutuamente**, como si estuviéramos sumando y restando un mismo valor numérico. Para solucionar este problema, podemos colocar el movimiento de los ojos del muñeco dentro de un bloque **hacer en orden**, como se ilustra a continuación.



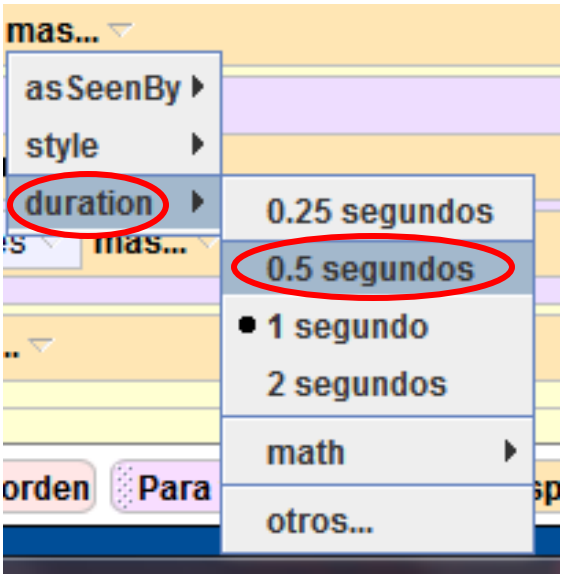
Ahora, al ejecutar este programa, los ojos del muñeco se mueven hacia arriba y hacia abajo. Una observación útil es que las animaciones en Alice requieren — de forma predeterminada — un segundo para que se ejecuten. Normalmente, dentro de un bloque **hacer juntos**, es conveniente que todas las instrucciones tomen la misma cantidad de tiempo. Como el saludo de «*hola*» que hace el muñeco tarda un segundo, es conveniente que las instrucciones que permiten el movimiento de los ojos hacia arriba y hacia abajo también requieran un segundo. Por lo tanto, el movimiento de los ojos hacia arriba y hacia abajo en total va a tomar un segundo, lo que significa que cada una de ellas por separado



[13] Lo importante es que usted comprenda que de momento estamos trabajando a nivel de ojo derecho. Una vez que haya terminado el análisis, podrá notar que el razonamiento se puede extender perfectamente al ojo izquierdo, y así lograr el «*coqueteo visual*» deseado.

debiera durar medio segundo. Para modificar la duración de una animación instrucción, hacemos clic en **más**, y luego cambiar el valor a 0,5, como se muestra en la Figura 17:

Figura 17 . Modificando la duración de una instrucción.

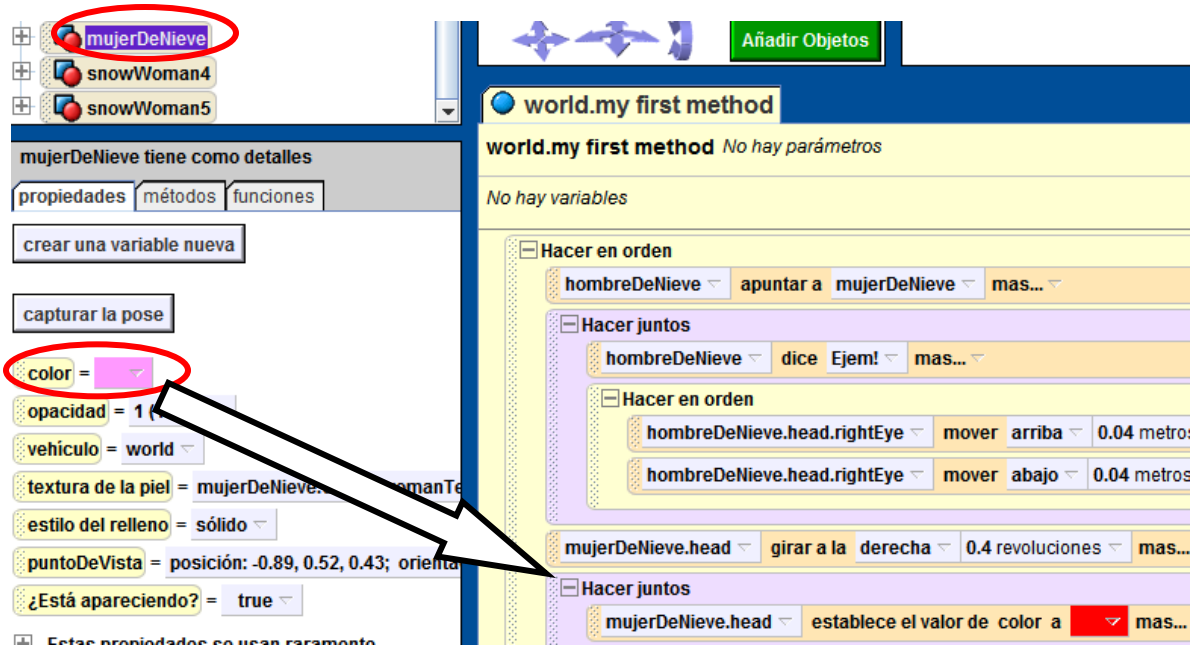


A pesar de los cambios que se han introducido, es necesario completar los últimos pasos que se habían descrito en el guión de trabajo. Ahora hay que implementar el momento en el cual la mujer de nieve la gira la cabeza. En la práctica, esto significa que a la cabeza le podemos pedir que dé 0,4 revolución hacia la derecha, tal como se muestra a continuación.[14]

El cuarto paso consiste en que la mujer se sonroje al momento de voltear su cabeza hacia atrás mientras está con sus amigos. De esta manera, la inclusión de otro bloque del tipo **hacer juntos** será necesaria. A diferencia de la acción anterior, el cambio de color de la cabeza de la muñeca se debe hacer directamente en el establecimiento del nuevo **valor** que tendrá la **propiedad color**. Para esto, seleccione la muñeca de nieve, y luego la cabeza desde el árbol de objetos. Finalmente, se modifica el color desde la ficha **propiedades**. Una vez hecho esto, arrastre el atributo de propiedad hacia el bloque **hacer juntos** del editor de código, tal como se sugiere en la Figura 18. Cambiando el color de un objeto..

[14] Invocando al método de ensayo y error, se adoptó la determinación de girar en media revolución hacia la derecha la cabeza del personaje.

Figura 18. Cambiando el color de un objeto.



El código resultante se muestra a continuación. Finalmente, el último paso de la animación — o la última secuencia de acciones, si le quiere llamar así — consisten en que la cabeza de la mujer de nieve se ponga de color blanco. El procedimiento es similar al caso anterior, cuando ésta era de color rojo. El código final es el siguiente:

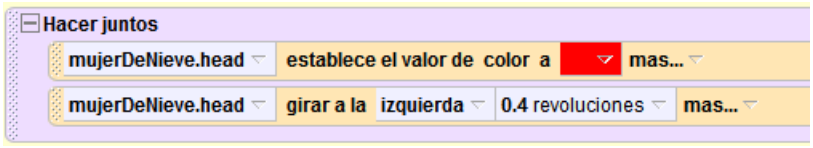
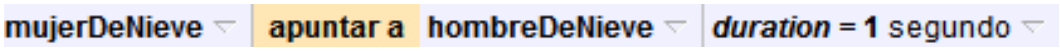


Figura 19. Implementación de la fiesta de la nieve.

1.6 Consejos y Técnicas

Estilo

En algún momento se comentó que al aplicar una instrucción es necesario hacer uso de una opción que se denomina **más**, tal y como se muestra a continuación:



Esto conduce a un menú emergente. Una de las tantas alternativas que ofrece Alice es el **estilo**, que le permite al programador especificar el modo en que se va a hacer la transición de una animación hacia la siguiente. Lo que muestra el **cuadro de diálogo** son opciones como empezar y terminar suavemente, transición abrupta, empezar suavemente con final abrupto, o comenzar abruptamente.

Ahora bien: ¿cuál de ellas es la más apropiada? La respuesta viene dada por el nivel de **uniformidad** que se le quiera dar a la animación que se esté realizando., o por aquella que mejor se adapte a lo que requiera el libreto de trabajo.

Sonido

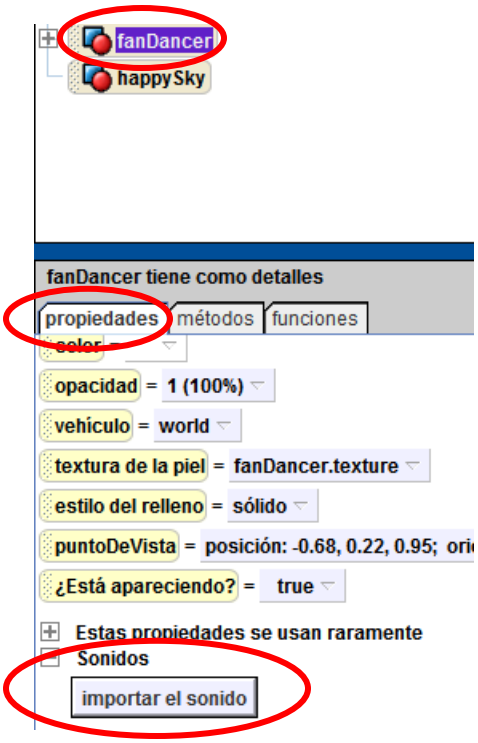
Con Alice, es posible insertar archivos de sonido para enriquecer los mundos virtuales que vayamos construyendo, y soporta exclusivamente los formatos wav y mp3. El uso del sonido no es un requisito para la mayoría las animaciones, pero sin dudas que permite imprimirle mayores dosis de realismo.

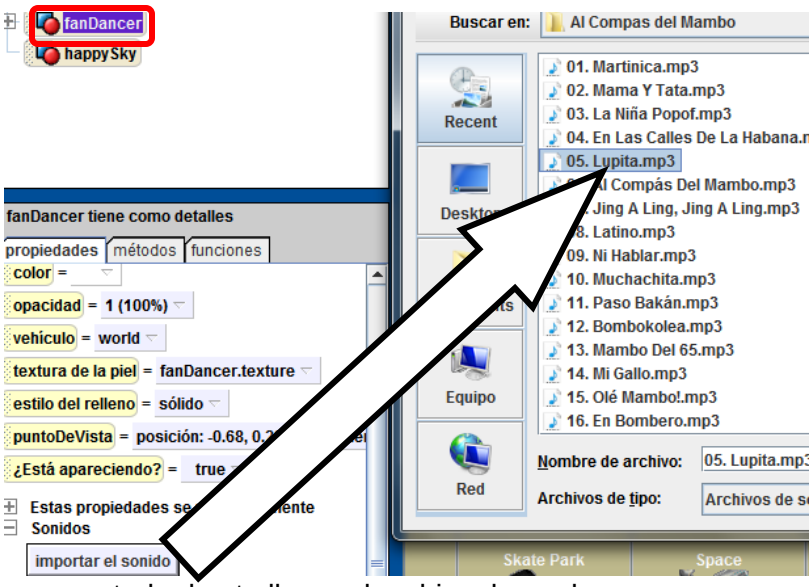
Sin ir más lejos, en el ejemplo de la fiesta de la nieve, hemos dejado de lado que de fondo se debe escuchar la canción «*let it snow*» de Frank Sinatra. En otros casos, como en el de la Figura 20, la incorporación de sonido se hace necesaria, para comprender que la bailarina está danzando al compás de una melodía que sale desde una radio que no se alcanza a apreciar con claridad en la imagen.

Figura 20. Tipo de escena donde es necesaria la inclusión de sonido.

Un sonido puede ser importado en la lista de propiedades para todo el mundo o para un objeto determinado, como se ilustra en la imagen adjunta. En este mundo, el sonido se destina únicamente para acompañar a la bailarina. En las imágenes que se muestran a continuación se exhibe la manera en cómo importar el sonido, tanto para el mundo como para algún objeto en particular. La Figura 21 muestra la primera parte del proceso de elección de una pista de audio:

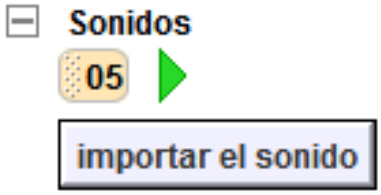
Figura 21. Importando un archivo de audio.

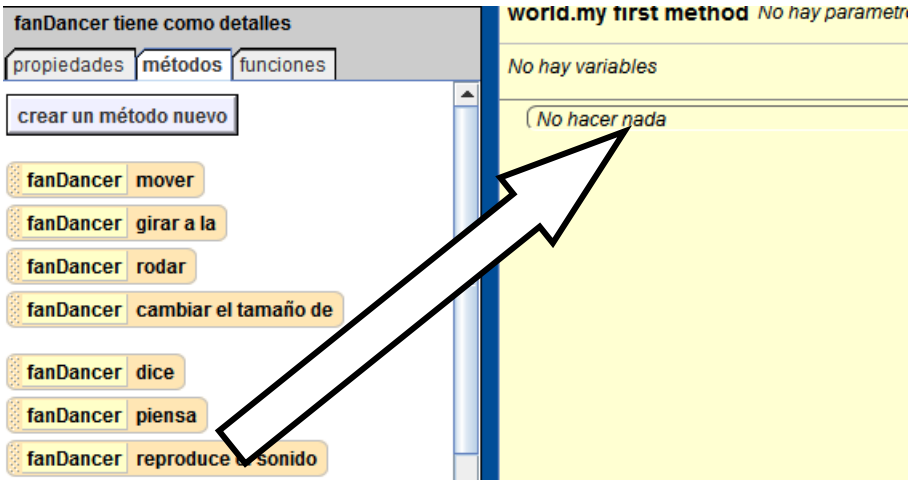




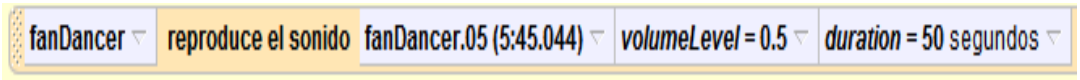
Para importar el archivo de sonido, haga clic en el botón **importar el sonido** que está en el submenú **Sonidos** de la pestaña **propiedades.** A continuación, aparecerá un cuadro de selección de archivos, que le permitirá acceder al directorio donde usted haya guardado las pistas de audio que desee emplear en la animación. Tenga en cuenta que puede ser necesario navegar a través de varias carpetas en el computador hasta llegar al archivo deseado**.**

Una vez que está listo, podemos apreciar que el archivo está disponible dentro del grupo de sonidos del objeto. Para reproducir el sonido como parte de una animación, seleccione el método **reproducir el sonido**, y luego arrástrelo hacia el área de código, en la zona donde desee que se ejecute el clip. Es posible asignar otros parámetros asociados con el sonido, como por ejemplo: volumen o duración, que se pueden encontrar directamente en la opción **más**.





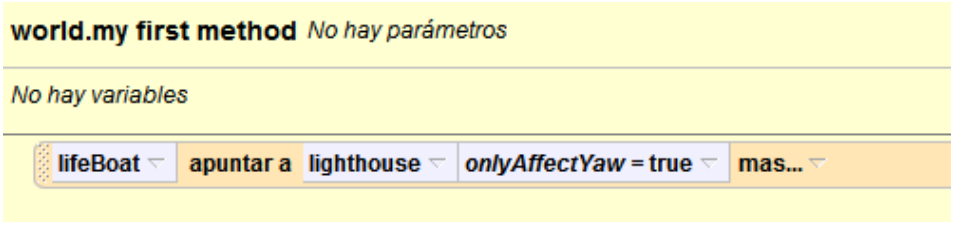
Una vez que se ha arrastrado la instrucción hacia el editor, podemos configurar la reproducción del archivo de sonido con los otros **parámetros** que nos ofrece Alice, y que se aprecian en la imagen adjunta:



«*Apuntar a*» y «*onlyAffectYaw*»

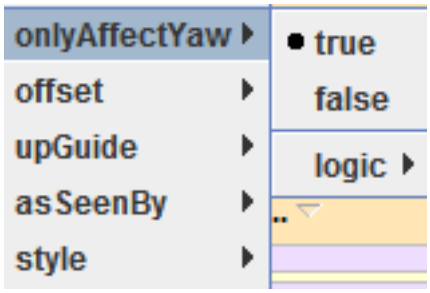
Una acción común en las animaciones consiste en mover un objeto hacia otro. El primer paso es hacer que el objeto que se va a mover haga un giro y apuntar hacia el su destino. Por ejemplo, en esta imagen los remeros que están en el bote pueden remar hacia el faro de la isla.

La instrucción **apuntar a** parece ser la más apropiada para girar el bote hacia el faro.



Como usted puede apreciar, esta sentencia provoca que el bote apunte hacia el faro, pero genera el efecto que estuviera a punto de hundirse en un extremo, tal como se observa en esta imagen:

La inclinación del bote salvavidas es porque está dirigido hacia el centro del faro, que es más elevado. Para evitar este efecto, Alice proporciona el comando *onlyAffectYaw* para complementar la orden que viene después de aplicar el método *apuntar a*. Para utilizar esta característica, haga clic en **más**, y haga que **onlyAffectYaw** quede marcado como **true**, tal como se muestra a continuación:



Portapapeles

El portapapeles es una característica que nos ayuda a hacer más amable la programación en Alice, ya que permite guardar temporalmente el contenido de una instrucción, o de un bloque de sentencias para poder usarlos más tarde. Opera de la misma forma que el clásico *copiar y pegar* de Windows, con la diferencia que en Alice no se pueden emplear las combinaciones de tecla:

- *Control+C* para copiar; o

- *Control+V* para pegar.

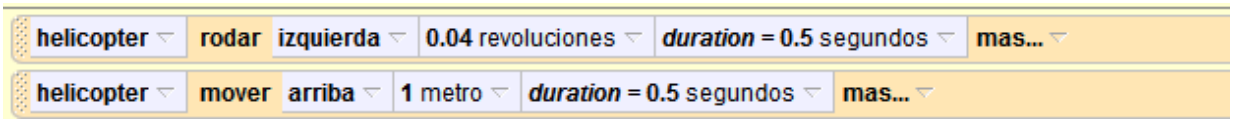
Para copiar las instrucciones de un lugar a otro, haga clic con el botón izquierdo sobre la instrucción o en algún bloque **hacer en orden** o **hacer juntos**, y luego arrástrela hacia el portapapeles.

Una vez que estén ahí, basta con arrastrarla hacia el lugar del código fuente donde quiera colocarla. En caso que sea factible hacerlo, aparecerá un signo más, y el portapapeles quedará de color blanco.

«As seen by»

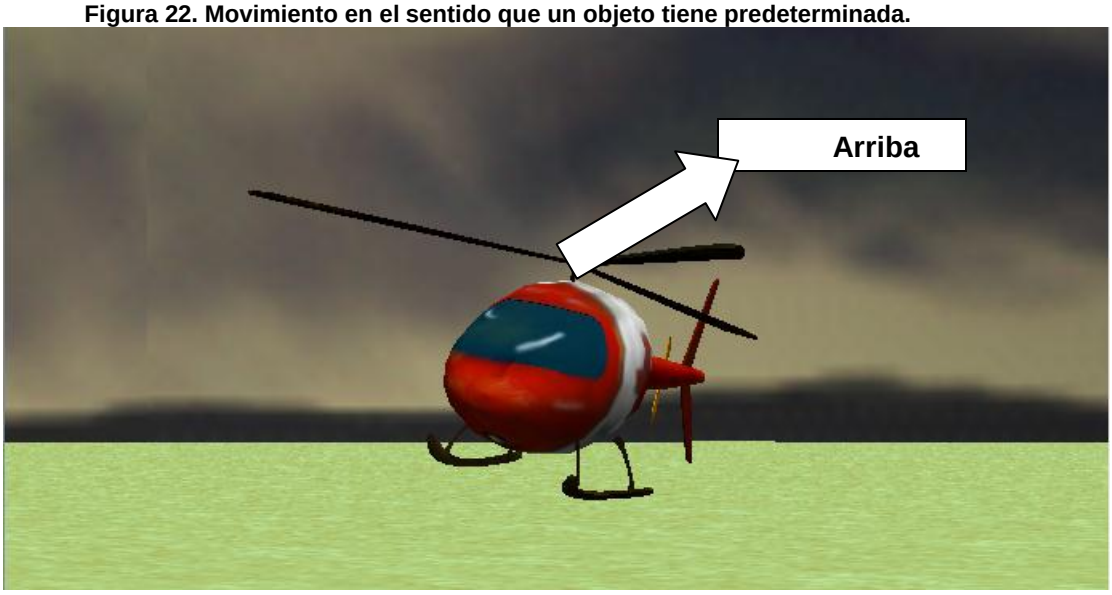
Como hemos visto hasta el momento, cada uno de los objetos de Alice tiene su propio sentido de la orientación, que es lo que se espera del comportamiento de un entorno 3D de animación. Pero, algunas veces, este sentido puede conducir a resultado bastantes particulares. Supongamos por ejemplo que tenemos un helicóptero como el de esta imagen:

Lo que vamos a hacer es hacer que el aparato ruede hacia la izquierda, y que luego se mueva hacia arriba, tal como ilustra esta porción de código:

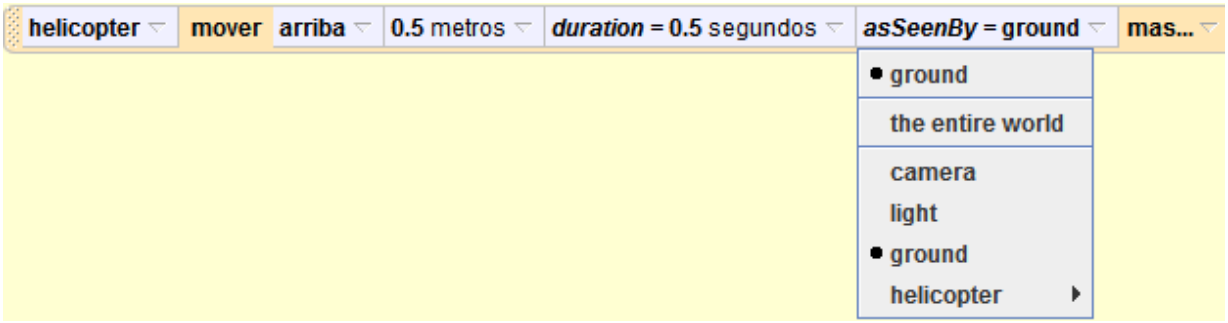


Al ejecutar la animación, podemos ver que el resultado final no es el que teníamos planeado en nuestra mente. Cuando el helicóptero se mueve hacia arriba, lo hace desde su propio sentido de la orientación, como se ve en la Figura 22:

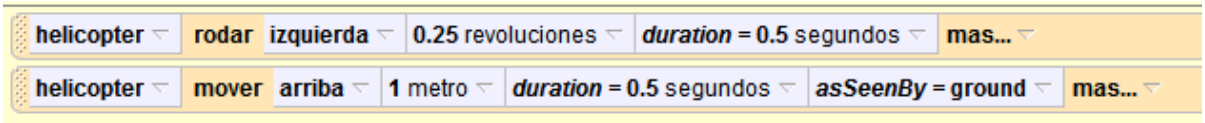
Figura 22. Movimiento en el sentido que un objeto tiene predeterminada.



A lo mejor, lo que queremos es que el movimiento sea con respecto a la tierra, y no desde la perspectiva del helicóptero en sí misma. Una manera de hacer más fácil el efecto de lograr que un objeto se mueva en relación a *cómo lo ve otro objeto* es por medio de la instrucción **asSeenBy**. En este caso, lo que haremos será crear este efecto desde la perspectiva del suelo — o *ground* en idioma de Alice:



El código que resulta después de aplicar los cambios, es el que sigue:



Ejercicios

1 Creación de un escenario.

En varios ejercicios de este apunte, se le dará un escenario que formará parte del problema que tenga que resolver. Pero al mismo tiempo, es probable que quiera ser creativo con algún proyecto en el cual quiera desarrollar una animación por cuenta propia. Este ejercicio va en esa línea. Recordemos que un escenario es una descripción general de la escena y de la animación que se va a desarrollar. Hasta el momento, hemos trabajado con un escenario que contiene a la fiesta de la nieve donde un muñeco intenta bailar con una muñeca.

Acá se presentan algunas ideas para ayudarle a pensar en el diseño de su propia animación:

1. Pensar en las cosas que recientemente ha leído o visto en la televisión.
2. Cómo sería un día de ensueño.
3. Intercambios de ideas que pueda haber tenido con un amigo.

Considere lo siguiente:

- *¿Quién es el público objetivo?*
- *¿Qué es lo que queremos decir a la audiencia a través de nuestra animación?*
- *¿Qué quiere usted despertar en la audiencia: alguna lección en particular?*
- *¿Le gustaría crear algún juego que le permita al usuario controlar el flujo de acciones de la animación?*

Asegúrese de que el escenario incluya información suficiente que le permita responder a las preguntas enunciadas en la sección anterior.

2 Crear un libreto a partir de un escenario dado.

Cree un guión para cada una de las siguientes situaciones:

1. Un escenario para un juego de niños: Alice, el conejo blanco, y el gato de Cheshire están disfrutando de un juego de sillas musicales, en el contexto de la fiesta

del té. En un momento cualquiera, alguno de los personajes grita «*cambio*», y todos deben tratar de sentarse en alguna de las sillas disponibles. Se elimina una silla, y si algún personaje queda de pie, automáticamente se elimina del juego. El último que logre sentarse en la última silla disponible será el ganador.

2. Un escenario para un juego de video: un avión está volviendo a la cubierta de un portaaviones después de una misión de entrenamiento. El avión hace un semicírculo alrededor de la cubierta para obtener una posición de aterrizaje, para luego descender hasta la superficie del portaaviones. Después, como el avión continúa en movimiento, el piloto debe ajustar de manera gradual los controles para aterrizar de mejor manera. Una vez que el avión se haya acomodado en la superficie, se detiene.

Nota. Cada vez que usted complete parte de una animación, asegúrese de guardar lo que haya hecho hasta el momento. Trate de ser siempre ordenado con los nombres que le coloque a cada archivo, para que después sea más fácil identificarlo.

3 Pez en círculos.

Cree un mundo de isla con un pez en el agua. Ajuste el punto de vista de la cámara para que tenga una escena similar a la que se muestra en esta imagen:

Utilice el editor de código de programa para escribir una animación en la cual el pez nade en círculos alrededor de la parte delantera de la isla. La idea es que después de dar cinco vueltas completas, el pez salte del agua y luego se vuelva a sumergir.

4 El pingüino y la galleta.

Cree una escena inicial como la que se muestra en la siguiente imagen:

Escribir un programa en el cual el pingüino se acerque a la silla con la galleta, la tome con las manos, la coma, y que al final mueva sus alas — las veces que usted quiera — en señal de agradecimiento por el regalo que alguien le dejó.

5 Puerta abierta.

Cree un mundo en el cual Alice esté en frente de una puerta cerrada. A continuación, ella debe girar hacia la puerta, mover la manilla, abrirla, y caminar a través de ella. Finalmente, la cámara tiene que quedar apuntando a su cara.

6 Diversión magnética.

Cree un mundo en el cual Alice tenga un imán en su mano derecha. Elija cinco objetos metálicos diferentes a su elección, siendo el vehículo uno de los que tiene que seleccionar. Cada objeto tiene que ser apuntado con el imán, y luego moverse en dirección a Alice. Como el vehículo es demasiado grande, al momento de apuntar, ella deberá ser la que se mueva hacia el automóvil, impulsada por el imán. Mientras esto suceda, ella deberá decir «*Hey!!*».

Resumen

Esta unidad presenta los conceptos fundamentales de la programación en Alice, sin que hayamos entrado en muchas profundidades teóricas. Comenzamos con un ejemplo de diseño de guión, que se puede materializar en términos gráficos o textuales. Las herramientas de diseño incluyen un escenario — descripción de la historia, juego o simulación — y un guión gráfico. Un escenario indica las características que tengan los objetos en un momento determinado, junto con las acciones que deban realizar. Un guión gráfico divide un escenario en una secuencia de escenas que en conjunto proporcionan un sentido al ordenamiento que deban seguir las acciones. Algunas estas acciones se llevarán a cabo en secuencia (uno después del otro) y otras en forma simultánea. Una vez que se ha confeccionado, el guión gráfico se emplea como una guía que facilita el tránsito hacia la escritura final del programa.

Conceptos importantes vistos en la unidad

- Un escenario es una forma de plantear un problema, en términos de cómo se debe implementar la animación global en función de lo que sea necesario resolver.
- Un guión puede ser gráfico o textual.
- Un guión gráfico es una secuencia de hechos que puede ser confeccionado a mano o por bocetos computacionales que tiene como propósito dividir un escenario en una secuencia de escenas principales.
  - Cada dibujo del boceto representa un *estado* de la animación, o una *instantánea* de ella, donde podemos encontrar la posición, color, tamaño y otras propiedades puntuales de los objetos que intervengan en ella.
- Un guión gráfico textual es una lista de «*cosas por hacer*», que proporciona una lista de los pasos algorítmicos que describen las secuencias — **hacer en orden** — y/o las acciones simultáneas — **hacer juntos.**
- El programa consta de líneas de código que especifica las acciones a realizar con los objetos.
  - El espacio para escribir el código recibe el nombre de editor.

Orientaciones Pedagógicas

Para conseguir los aprendizajes esperados, es necesario que los estudiantes tengan claridad con respecto a los siguientes puntos:

- Un programa corresponde a un conjunto de instrucciones que le indica al computador lo que tenga que hacer.
- La documentación es una parte fundamental que rige la correcta planificación de diseño de un programa.
- Un programa computacional se puede construir en base a un listado bastante acotado de conceptos.

- Alice es un lenguaje de orientado a objetos.
- Los programas de Alice se recrean a través de mundo tridimensionales.
- Cada objeto en Alice admite seis posibles direcciones de movimiento.
- La creación de un programa computacional es un proceso que lo enfocaremos en torno a cuatro pasos:
  - Leer el escenario de trabajo.
  - Hacer el diseño, por medio de un libreto gráfico o uno textual.
  - Implementar lo que se ha diseñado.
  - Hacer pruebas al funcionamiento.

Para que estos entendimientos se puedan llevar a cabo de manera exitosa, se recomienda que los estudiantes desarrollen en las sesiones de práctica no sólo los problemas propuestos al final de la unidad, sino también que reflexionen en torno a estas preguntas, que se pueden distribuir en el tiempo que el profesor estime conveniente:

- ¿Cuáles son los tipos de instrucciones que están disponibles para crear un programa de computador?
- ¿En qué consiste la técnica de descomposición de problemas que se usa con bastante frecuencia en el ámbito de las Ciencias de la Computación?
- ¿Cómo determina el alumno cuál es la mejor manera para descomponer un problema?
- ¿Cómo se le puede explicar a alguien que la solución que uno propone requiere de ciertas precondiciones, para que al final pueda entregar los resultados esperados?
- ¿Cómo determina la elección de los valores de prueba que puede utilizar para demostrar la correctitud de su solución algorítmica?
- ¿Cómo se emplean las representaciones 3D que ofrece Alice para poder recrear mundos y construir programas con ellos?
- ¿Cuáles son las direcciones en que se puede mover algún objeto en Alice?

Unidad 2

Presentación de la Programación Orientada a Objetos

En esta unidad revisaremos algunos de los conceptos más importantes que existen dentro de la programación orientada a objetos, junto con incentivar el desarrollo del pensamiento sistémico por medio de las nociones de métodos, parámetros y herencia.

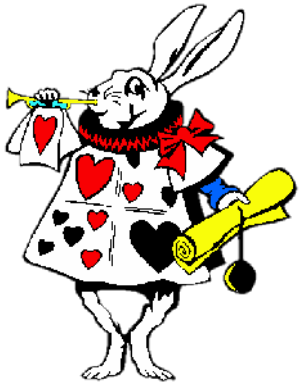
Sentido educativo de la unidad

Aprendizajes esperados.

- Identificar y aplicar el concepto de método dentro del contexto de alguna aplicación computacional.
- Reconocer y aplicar los conceptos de parámetro y argumento.
- Identificar la diferencia conceptual entre método y función (o pregunta).
- Construir eventos que involucren métodos y parámetros.

Contenidos.

- Métodos.
- Parámetros.
- Argumentos.
- Eventos.



Primera Parte

Métodos y Parámetros

En esta unidad veremos los conceptos básicos de la programación orientada a objetos, que se instrumentalizarán por medio de las nociones de métodos, parámetros, herencia y eventos.

En términos muy resumidos, diremos que un método — que fue mencionado de manera muy tibia en la unidad anterior — corresponde a un conjunto de acciones que los objetos del mundo llevarán a cabo cuando solicitamos que sea ejecutado. En tanto, un parámetro corresponde a una característica de la orientación a objetos que permite enviar información a un método para que se lleve a cabo de acuerdo al valor que contenga.

Los mundos creados como ejemplos y ejercicios en la primera unidad son pequeñas animaciones que han sido levantadas empleando los recursos disponibles que están presentes en los mundos virtuales. Como es natural, es posible que los proyectos de trabajo aumenten considerablemente de tamaño a medida que las exigencias sean cada vez mayores.

Cuando esto ocurre, será necesario utilizar métodos, ya que ellos permiten hacer una **abstracción** de una tarea global como una colección organizada de todas las mini tareas que la compongan. Por ejemplo: se puede pensar en un método llamado *llamarAtención,* que se contemple acciones como que el mono de nieve gire hacia su objetivo, la mire, la salude y le pregunte si quiere bailar con ella. Hemos visto que esa muñeca de nieve es única, y por lo tanto, no ha sido necesario individualizarla de otras. Sin embargo, es posible que en la fiesta de la nieve existan muchas muñecas, y que el hombre quiera a una en particular. En ese caso, si conocemos el **identificador** de la señorita, será posible que podamos invocar al método *llamarAtención*, pasándole como **parámetro** el nombre de la muñeca con la que quiera bailar. Hasta el momento, interesa que vaya estableciendo una conexión mental entre los conceptos que vayamos destacando con negrillas. En Alice, los métodos pueden ser de dos tipos:

- **De personajes**, que se definen como una acción que afecte sólo a un objeto del mundo. En Ciencias de la Computación, los métodos de nivel de personajes se les denominan *métodos de nivel de clase*.
- **De mundo**, que envuelven acciones que afectan a más de un tipo de objeto. En lenguajes de programación como Java o C++, los métodos de

mundo se invocan directamente desde el método *main*, que es el primero que se ejecuta cuando el usuario activa la ejecución del programa.

Partiremos viendo cómo crear y ejecutar nuestros propios métodos de nivel de mundo. Para poder ejecutarlo, es necesario que éste sea llamado o *invocado*. Los comentarios se utilizan para documentar el código. Después veremos el concepto de *parámetro*, que es un recurso empleado para enviar los valores que un método requiera para trabajar. Los valores que se envíen pueden ser de varios tipos diferentes: número, objeto, valor de alguna propiedad como el color, largo, etc.

2.1 Métodos que están a nivel del mundo

Cada tipo de objeto de la galería de Alice tiene un repertorio de acciones primitivas que puede realizar, tales como:

- Mover.
- Girar.
- Rodar.
- Apuntar a, etc.

Entonces, podríamos decir que al finalizar la primera unidad, habíamos escrito nuestro primer programa como una colección de instrucciones primitivas, que involucraban tanto al hombre como a la mujer de nieve. **En esta sección, vamos a aprender a escribir nuestros propios métodos de nivel de mundo**.

Métodos

Un **método** es una secuencia ordenada de instrucciones que debe ejecutar cuando sea invocado. Hasta el momento, sabemos que en Alice las instrucciones se llevan a cabo por objetos que participan del mundo virtual que estemos construyendo. También podemos decir que los objetos pueden llevar a cabo ciertas acciones de manera individual, sin que el comportamiento de los otros elementos se vea afectado. Pero también está el otro escenario: cuando los objetos puedan estar interactuando con otros objetos. Cuando esto ocurre, los métodos se pueden ver como una prescripción que hace un objeto en relación a la forma en cómo otros se deban comportar.

Aquellos métodos que definan el comportamiento individual de un objeto, se denominan **métodos de nivel de objeto** — o de personaje, considerando el contexto de trabajo de Alice, o bien de *clase*, de acuerdo a la jerga empleada por otros lenguajes de programación.

Aquellos métodos cuyas acciones asociadas tengan un impacto que afecten a más de un objeto, se considerarán **métodos de mundo**. En esta sección del apunte nos vamos a centrar exclusivamente en los métodos de nivel de mundo, dejando para la próxima aquellos relacionados con los de objeto o personaje.

Cuando usted creó sus propias animaciones en los ejercicios de la unidad anterior, es probable que haya evaluado la posibilidad de ir diseñando escenarios de trabajo cada vez más complicados, que involucren nuevos giros a la trama, o que éstas se puedan transformar en algún videojuego o simulación interactiva. Por tanto, a medida que vamos complejizando nuestros requerimientos, el código final también se verá

afectado, sobre todo en la cantidad de líneas que sean necesarias de escribir. Pero no sólo en estos mundos virtuales ocurre esto, sino que en la realidad es eso efectivamente ocurre: los sistemas operativos pueden llegar a tener cientos de miles de líneas de código.

Entonces, es natural que uno se pregunte: ¿cómo puede un programador tratar con esas enormes cantidades de código? Una de las formas consiste en dividir un programa de gran tamaño en varios pequeños *módulos*, donde cada uno realice una parte específica de la funcionalidad del programa en general: ¿se acuerda del concepto de «*metodología top-down*» que vimos al principio de la primera unidad?

En la programación orientada a objetos, los módulos se utilizan para definir las **clases de objetos**, junto con los métodos asociados. Por lo tanto, los métodos juegan un papel importante en la estructuración del código del programa, porque le permiten al programador pensar en un conjunto ordenado de acciones como si fuera una sola. En Ciencias de la Computación, esto es lo que denomina **abstracción**.[15] Por otra parte, cada método puede ser probado para que el programador se asegure que funcione correctamente. Además, acá opera un principio básico y elemental:

Encontrar un error en unas pocas líneas de código es mucho más fácil que hallarlo en cientos de miles.

Por lo tanto, no sólo los métodos ayudan a que un programa sea más fácil de diseñar, sino que además simplifica de sobremanera las tareas de pruebas y de depuración.

Ejemplo

En nuestro primer programa, un muñeco de nieve está tratando de bailar con una mujer de nieve, pero apenas hizo un intento para ver si sus planes resultaban. Tal vez sea más realista que el hombre no renuncie tan fácilmente, y que intente captar la atención de ella en más de una oportunidad. Por lo tanto, su comportamiento involucrará acciones como:

- Decirle «*Ejem!*» a la señorita al mismo tiempo que le mueve los ojos.
- Hacer que su cabeza apunte hacia la de ella.

En términos de la animación, significa que la secuencia de pasos anterior se tenga que repetir varias veces:

¿5, 10, 15, 20, 50?

De forma intuitiva, uno podría pensar en aplicar la técnica de copiar y pegar las instrucciones en el editor, pero esta tarea se hace tediosa si es que este proceso se tuviera que llevar a cabo en reiteradas oportunidades. ¿Por qué cree que eso es así?

Entonces: ¿no sería mejor que estas instrucciones se puedan agrupar, para que en conjunto trabajen como si fueran un método? Lo bueno de esto es que una vez que el método se haya definido, a Alice se le puede pedir que lo ejecute la cantidad de veces que sea necesario, sin tener que copiar las instrucciones una y otra vez en el editor.

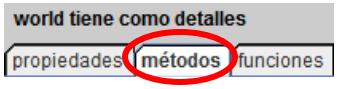
[15] El concepto de abstracción se explicará de manera más sencilla a través del ejemplo que desarrollaremos a continuación.

Creando el primer método

Vamos a escribir un método denominado **llamarAtencion**. Este método implica tanto al hombre de nieve como a la mujer, por lo que pasa a ser uno de tipo *mundo*, de acuerdo a su definición, porque su definición impacta a los objetos mujer y hombre de nieve.

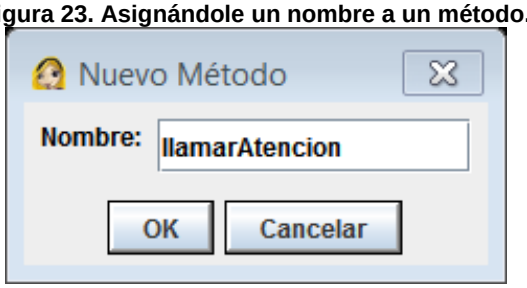
Si bien las instrucciones que utilizaremos son similares a las de la unidad anterior, la animación será construida desde cero, empleando un nuevo mundo junto a una nueva escena inicial. El propósito de esto es para ilustrarle a usted el proceso completo de escribir su propio método.

Para esto, vamos a ir al árbol de objetos del mundo, y a continuación, seleccionamos la ficha de **métodos** del panel de detalles.

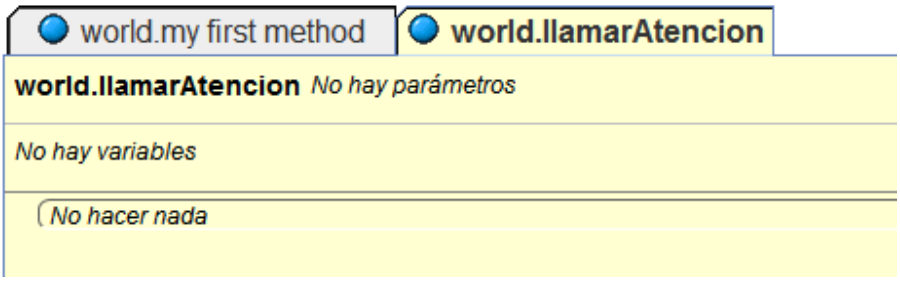


Después, hay que presionar sobre el botón **create a new method.** Una vez hecho esto, aparecerá un **cuadro emergente de texto** en el cual se pedirá el ingreso del nuevo método, momento en el cual escribiremos **llamarAtención**, tal como lo ilustra la Figura 23:

Figura 23. Asignándole un nombre a un método.



A continuación, se abrirá una *pestaña* en el editor de código, y Alice mostrará el espacio habilitado, en señal de estar preparada para la escritura de las sentencias propias del método.



En caso que usted deba abandonar el proyecto de trabajo, puede reanudarlo más tarde, yendo a la pestaña **métodos**, y haciendo doble clic en el botón **modificar** que está al lado de **llamarAtencion**.

Comentario

Si el botón de reproducción se aprieta en este momento, la animación no se ejecutará, debido a que el método **llamarAtención** no ha sido llamado a la acción, lo que en términos de Ciencias de la Computación significa que el éste no ha sido **invocado**.

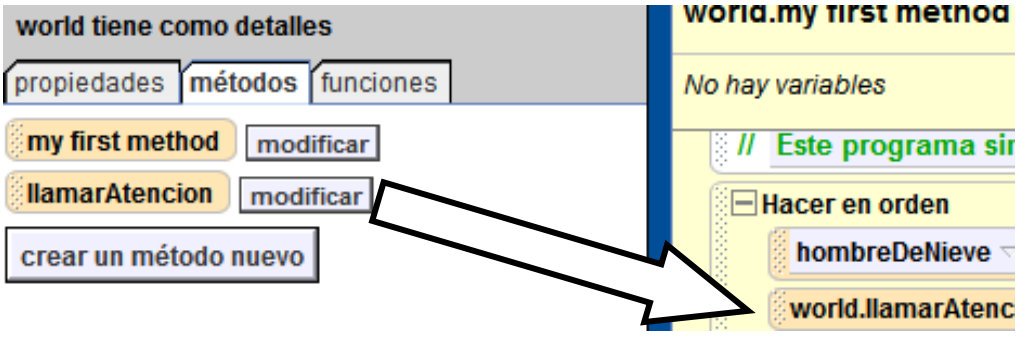
Invocación de métodos

Así que, ¿cómo podemos llamar (invocar) nuestro nuevo método? Recordemos que cuando se hacía clic en el botón de reproducción, Alice automáticamente ejecuta el **método my first method**. Podemos ver por qué sucede esto al examinar cuidadosamente los eventos del editor, que se encuentra en la parte superior derecha de la interfaz de Alice, como se ve en la Figura 24. La instrucción en este editor le dice a Alice cuando el mundo se inicia, se ejecuta el primer método *World.my first method*. No pusimos esta instrucción aquí, sino que la interfaz de Alice se programa automáticamente de este modo. Así, cuando el usuario hace clic en el botón reproducir, el mundo comienza su ejecución, y el método arranca de inmediato.

Figura 24. Por defecto, cuando el mundo arranca, se ejecuta *my first method.*

Vamos a aprovechar este recuento. Todo lo que tenemos que hacer es arrastrar el método **llamarAtencion** que está incluido en la ficha de métodos del panel de detalles en **my first method**, tal como se ilustra en la Figura 25. Ahora, cada vez que el botón de reproducción se presione, **my first method** será ejecutado, y el método **llamarAtencion** será invocado.

Figura 25. Invocando a un método.



Por supuesto, queremos que otras acciones se añadan a lo que hemos estado construyendo. De esta manera, las nuevas instrucciones se han agregado al primer método World.my first method, tal como se muestra en la imagen adjunta. Usted podrá notar que este programa es algo diferente del código del primer programa escrito a fines de la primera unidad. En esta nueva versión, el muñeco de nieve trata de llamar la atención de la mujer de nieve dos veces, acercándose a ella después de la primera vez.

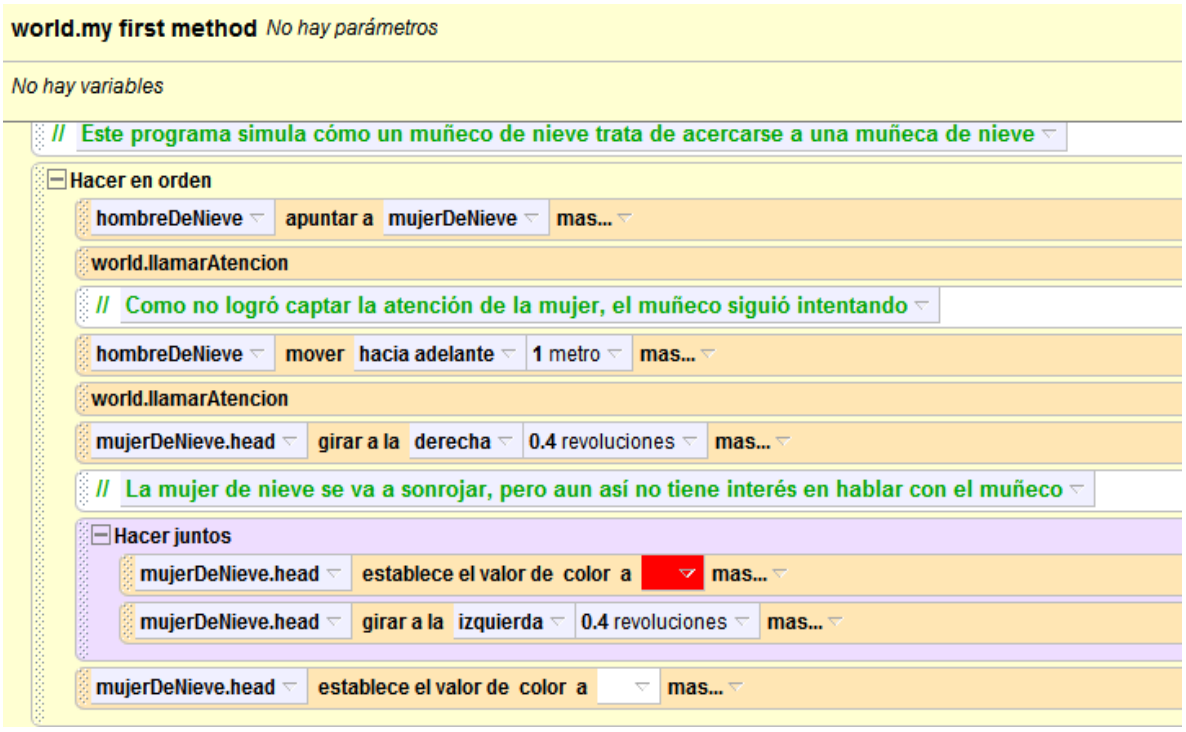
Comentarios

Ahora que hemos escrito nuestro propio método, es hora de mirar un componente útil para los programas, que son los comentarios. Partamos diciendo que los comentarios no son instrucciones ejecutables: esto significa que Alice puede ignorar los comentarios cuando se ejecuta un programa. Sin embargo, se uso es bastante recomendado, porque **facilita la legibilidad del código** que se está escribiendo en el editor.

Los comentarios ayudan al lector a entender lo que hace un programa, y son particularmente útiles cuando otra persona tiene que leer el código del programa.[16] Los comentarios de Alice se crean arrastrando el cuadrado con barras en diagonal de color verde. Una vez que se ha insertado, se da a conocer una descripción de lo que un determinado **bloque de código** vaya a realizar. Al respecto, la Figura 26 ilustra lo que se ha comentado.

[16] Si hay algo que es muy difícil de lograr, es poder llegar a comprender la manera en cómo otras personas razonan.

Figura 26. Ejemplo de código comentado.



En los métodos, se recomienda incluir comentarios al inicio de un método, para apoyar la explicación de lo que hace. El primer comentario de la Figura 26 proporciona una declaración general acerca de lo que hace el programa:

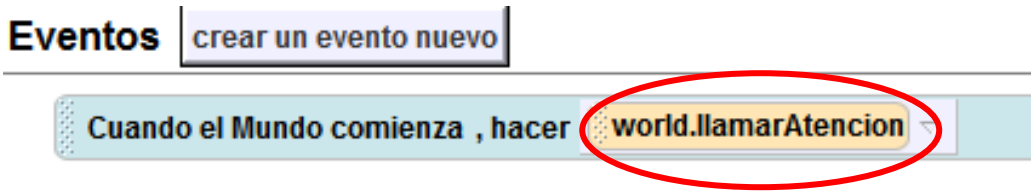
- Simula las acciones de un muñeco de nieve tratando de satisfacer una mujer de nieve.
- El segundo comentario se describe las líneas de código que anima el muñeco de nieve tratando de llamar la atención de la mujer de la nieve.
- El tercer comentario documenta las acciones que responde la muñeca de nieve.

Nota técnica

Aunque el editor de eventos lo veamos con mayor detención en la sección 2.7, vale la pena mencionar que esta parte de Alice se puede usar para modificar el método con el que arranque nuestro mundo virtual.

La modificación de esta instrucción le permite llamar a un método nuevo en lugar del primer método World.my first method cuando el usuario haga clic en el botón reproducir. Para modificar la instrucción en el editor de eventos, haga clic en la imagen a la derecha del primer método, tal como se aprecia en la lista desplegable que aparece en la Figura 27. Ahora, cuando el mundo empieza, el método se ejecutará **llamarAtencion** en lugar de método **my first method**.

Figura 27. Alterando la manera en cómo deba empezar la ejecución de un programa.



Ejercicios

Recordatorio. Asegúrese de añadir comentarios a los métodos para documentar lo que el método hace y qué acciones se llevan a cabo por las secciones de código dentro del método.

1 Kanga confundido.

Cuando el canguro Kanga salió a buscar algo para el desayuno en pleno campo, se encontró con un señal de tránsito bastante confusa. Kanga mira fijamente el signo durante unos segundos, y a continuación empieza a saltar hacia la izquierda y se voltea hacia el signo, salta hacia la derecha, gira hacia el signo y luego a la izquierda, a la derecha, y así sucesivamente.

Cree una simulación que implemente esta cómica situación. Debe escribir los métodos:

- ***hopLeft***, en el cual Kanga gire a la izquierda una pequeña cantidad, sale, y luego gire hacia el signo.
- ***hopRight***, en el cual Kanga gire a la derecha una pequeña cantidad, salte, y luego gire hacia el signo.

Con cada salto, Kanga debería hacer algún progreso para ver el signo. En el *primer método World.my first method*, invoque a los métodos *hopRight* y *hopLeft* dos veces, para hacer que Kanga tome un camino en zig-zag hacia el signo.

2 Galope y salto.

Kelly ha entrado en un espectáculo ecuestre como un saltador aficionado. Ella está un poco nerviosa por la competencia, así que ha decido tomarse un tiempo para practicar antes de las pruebas. Cree una escena inicial con un caballo y un jinete frente a una valla, tal como se muestra a continuación.

Escriba dos métodos:

- ***galope***, en el cual tanto el jinete cabalgan un paso hacia adelante. En este caso, las patas delanteras deben levantarse y después bajarse, al igual como tiene que ocurrir con las patas traseras. Todo esto se tiene que hacer mientras el caballo avanza hacia adelante.
- ***salto***, donde el caballo y el jinete deberán saltar la valla. El método salto debe ser similar, pero el caballo debe moverse hacia arriba lo suficiente como para pasar por arriba de la valla.

Pruebe cada método para asegurarse de que funciona como se espera. Usted tendrá que ajustar la distancia para que más realista.

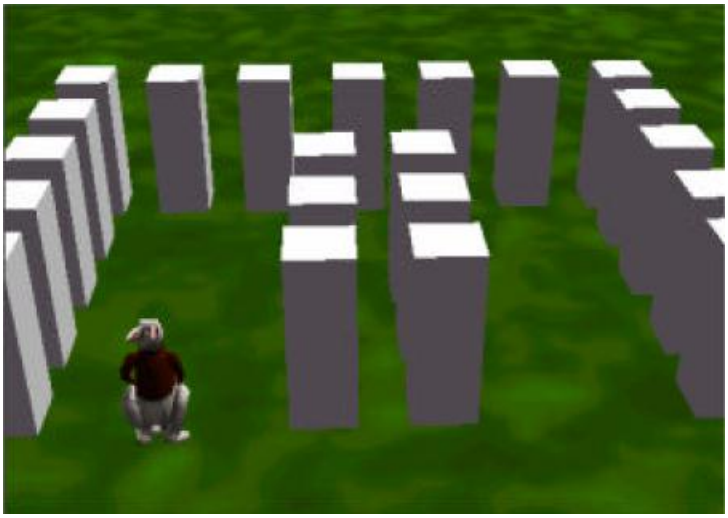
Sugerencia. Si hace que el caballo sea el **vehículo** para Kelly, usted sólo tendrá que escribir una instrucción para mover el caballo, y ella se dejará llevar por la trayectoria del animal.

Cuando usted piense que los métodos galope y salto están trabajando correctamente, escriba las instrucciones en el método **my first method** para que llame al método **galope** tantas veces como sea necesario para mover al caballo y al jinete cerca de la valla, y que luego invoque al método **saltar**.

Utilice el método de prueba y error para saber cuántas veces el método *galope* debe ser llamados para hacer que esta animación resulte ser lo más realista posible.

3 Conejo en un laberinto.

Construya un mundo con el objeto WhiteRabbit al borde de un laberinto, que se crea, usando bloques de la carpeta Formas. Diseñe métodos para hacer que el conejo camine hacia adelante y hacia atrás. También crear métodos para girar a la izquierda y a la derecha. En *World.my first method*, ubique estos cuatro métodos en orden, para que sirvan de guía para que el conejo avance con éxito a través del laberinto.



2.2 Parámetros

Es evidente a partir de los ejemplos y ejercicios de la sección anterior, que un programa puede hacerse con varios métodos. Cada método es un bloque de instrucciones, diseñado para realizar una tarea específica cuando se le solicite. Es posible que los objetos que participen puedan realizar acciones en más de un método de nivel de mundo.

Podemos apreciar que algún tipo de comunicación podría presentarse entre los métodos. En esta sección, nos fijamos en los parámetros. Los parámetros se utilizan para la comunicación entre los métodos.

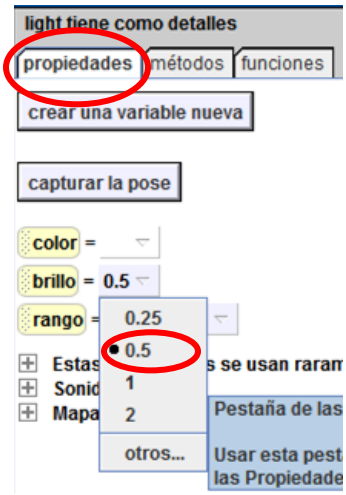
Ejemplo

Vamos a utilizar un mundo de ejemplo para ilustrar el concepto de parámetro que se enunció en el párrafo anterior. Para un concierto de primavera, nuestro comité de entretención ha contratado a un grupo de música popular, que se llama la banda **Bichos**. Nuestro trabajo consiste en crear una animación para anunciar el concierto. En la animación, cada miembro de la banda quiere mostrar sus habilidades musicales en una breve actuación en solitario.

Lo primero que vamos a hacer será la puesta en escena de lo que hemos comentado hasta el momento. En este sentido, la imagen que acompaña este párrafo muestra la escena inicial de la animación, donde el mundo está compuesto de una abeja reina, mantis, abeja trabajadora, hormiga — que forman parte del grupo animales — junto con el bajo, saxo, batería y guitarra, que componen los instrumentos musicales.

Una luz se utiliza para resaltar a algún miembro de la banda cuando le llegue el turno de actuar en solitario, de tal manera que el resto de los integrantes se vean menos brillantes. De esta manera, para lograr el efecto de atenuación, la luz se debe seleccionar del árbol de objetos, y luego alterar la propiedad del brillo: el valor 1 dejarlo en 0,5, como se ve en la Figura 28:

Figura 28. Atenuación del brillo de la luz a la mitad.



Guión textual con un parámetro

El argumento de esta animación es que cada miembro de la banda se destaca por medio del foco, y luego realizará una actuación en solitario, que consiste en tocar alguno de los instrumentos que ya se mencionaron.

Ahora vamos a crear un guión para centrar la atención y el movimiento de cada miembro de la banda durante su actuación. En principio vamos a trabajar con cuatro guiones textuales: uno por cada integrante, para que luego usted pueda apreciar la ganancia de la utilización de parámetros. La Tabla 5 da cuenta de ello:

Tabla 5 . Guiones textuales para cada miembro de la banda.

Hacer en orden: 1. El foco apunta a la **abeja reina.** 2. La **abeja reina** se mueve y toca el instrumento. 3. El foco apunta a la multitud.	**Hacer en orden:** 1. El foco apunta a la **mantis.** 2. La **mantis** se mueve y toca el instrumento. 3. El foco apunta a la multitud.
Hacer en orden: 1. El foco apunta a la **abeja trabajadora.** 2. La **abeja trabajadora** se mueve y toca el instrumento. 3. El foco apunta a la multitud.	**Hacer en orden:** 1. El foco apunta a la **hormiga.** 2. La **hormiga** se mueve y toca el instrumento. 3. El foco apunta a la multitud.

Por supuesto, podríamos escribir cuatro métodos: uno para cada guión, pero es evidente que los cuatro guiones son prácticamente iguales. La principal diferencia está en el nombre del integrante de la banda. Acá es cuando los parámetros entran a jugar. Vamos a colapsar — o factorizar — los cuatro guiones en un solo libreto de trabajo, y usamos un parámetro para comunicar cuál es el miembro de la banda que se presentará a continuación. El nombre del parámetro es **miembroDeLaBanda:**

Tabla 6. Fusión de varios algoritmos en uno solo.

```
Parámetro: miembroDeLaBanda
Hacer en orden:
      1. El foco apunta a miembroDeLaBanda.
      2. miembroDeLaBanda se mueve y toca su instrumento.
      3. El foco apunta a la multitud.
```

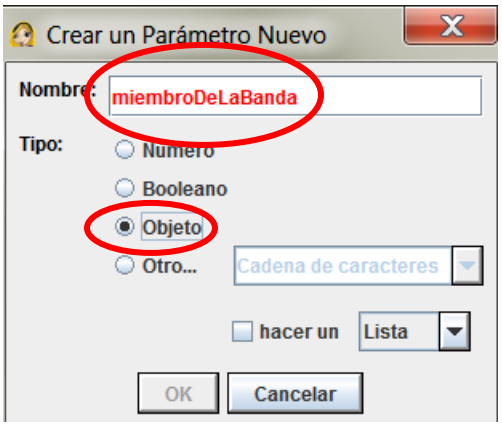
El nombre de parámetro **miembroDeLaBanda** es completamente arbitrario: al computador «le *da lo mismo*» cómo se llame, pero a nosotros como programadores no, porque entre más legible sea el nombre que le demos, nuestra documentación será cada vez más legible y entendible.

El parámetro **miembroDeLaBanda** está tomando el lugar del nombre del objeto específico que va a realizar la actuación. De esta manera, el valor puntual que tome el parámetro, va a determinar el comportamiento que deba adoptar el método que haya creado.

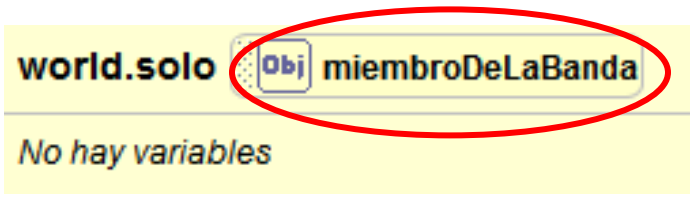
Usando un parámetro de objeto

Para implementar la animación, un nuevo método, llamado **solo**, se ha creado. Cuando se ha presionado el botón **create new parameter**, aparecerá un cuadro de diálogo como se muestra en la Figura 29, luego de lo cual se introduce el nombre del parámetro, y el tipo al que corresponda. En este ejemplo, el nombre del parámetro es **miembroDeLaBanda**, y su tipo es **Objeto**.

Figura 29. Creación de un parámetro.



Una vez finalizado el proceso, el nombre de parámetro aparece en la parte superior izquierda del panel del método, como se muestra a continuación.



Note que las características aparecen inmediatamente antes del nombre del parámetro, lo que significa que cualquier método que llame a **solo** deberá suministrar un valor, que en este caso corresponde al nombre del objeto.

Utilizando un parámetro tipo Objeto

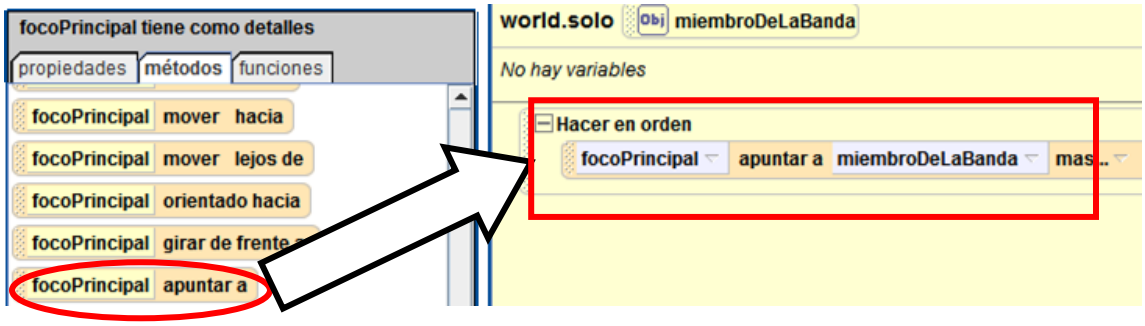
En este ejemplo, dos formas diferentes de usar un parámetro tipo Objeto serán ilustradas: (1) para especificar que el parámetro será el objetivo de una acción determinada; y (2) para especificar que el objeto cuyo valor corresponde al del parámetro sea que el implemente una acción.

(1) El parámetro será el objetivo de una determinada acción. El primer paso en el guión textual anterior era que el foco apuntara a uno de los integrantes de la banda. Este punto de mira está seleccionado en el árbol de objetos, y su método **apuntar** *a* se arrastra hacia el editor. A continuación, seleccionamos la lista de los posibles

objetivos que tendrá el método **apuntar a**.

Notemos en este caso, que lo que andamos buscando no es algo que vaya a aparecer dentro del árbol de objetos, sino que Alice lo considera como si fuera una **expresión**, ya que **miembroDeLaBanda** es un parámetro. La Figura 30 muestra el resultado final de la operación:

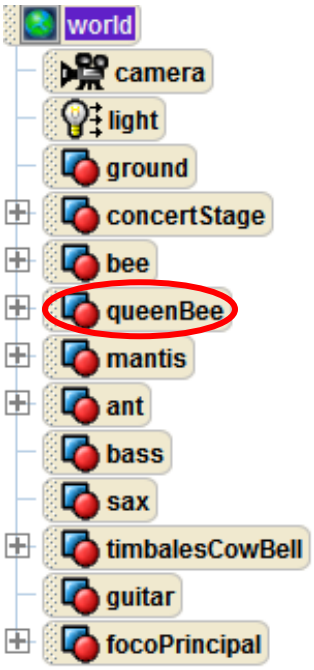
Figura 30. Un parámetro es el destinatario de una acción.



(2) El parámetro realiza una acción. El segundo paso en el guión gráfico es que «*el miembroDeLaBanda se mueve y toca el instrumento*». A partir de esta definición, queda claro que el **miembroDeLaBanda** al que se refiere la instrucción anterior es el parámetro que tendrá que ejecutar la acción de moverse y tocar el instrumento.

Está claro que, en esta acción, el **miembroDeLaBanda** realiza la acción, y no pasa a ser destinatario de ella: esta es la segunda forma de cómo se pasa un parámetro a un método. Cuando no conocíamos el concepto de parámetro, podíamos haber ido al árbol de objetos y haber elegido al integrante de la banda que iba a hacer su presentación en solitario.

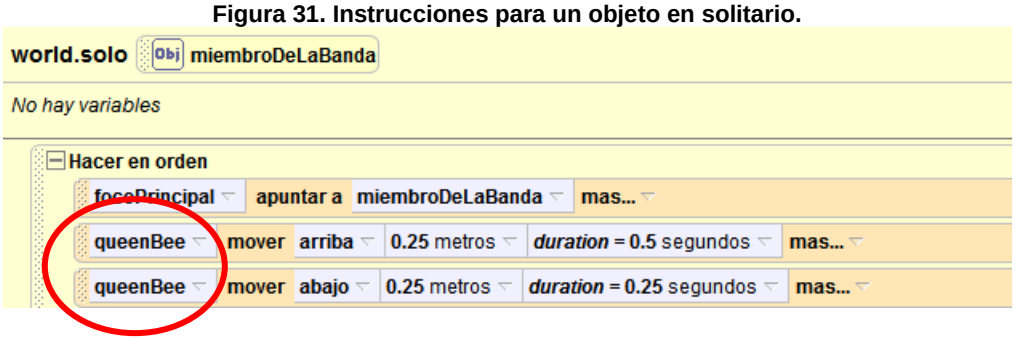
Ahora bien, como queremos optimizar el proceso, debemos saber que **miembroDeLaBanda** no es algo que no está el árbol de objetos, como se puede apreciar en la imagen adjunta. Entonces, viene la pregunta:



¿Cómo se incluye el parámetro en el código?

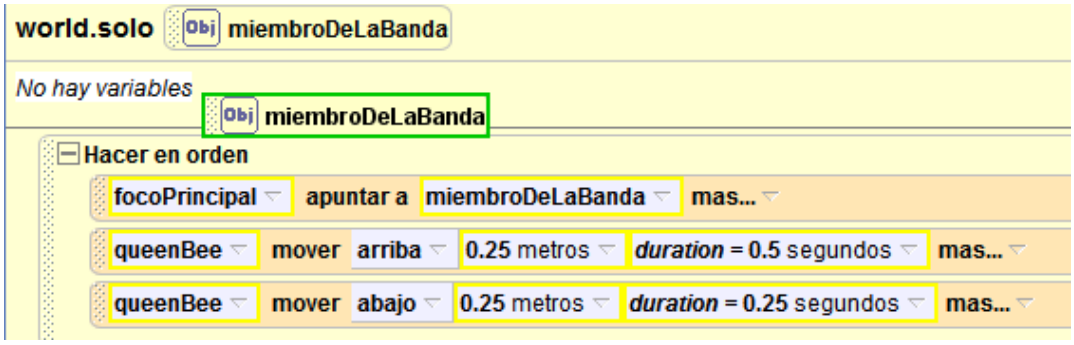
Vamos a trabajar con dos etapas: la primera consiste en seleccionar arbitrariamente cualquiera de los objetos —abeja, mantis, hormiga, abeja reina —, y crear las instrucciones asociadas. En este caso, elegimos a la abeja reina, y le asignamos las instrucciones, tal como se muestra en la Figura 31:

Figura 31. Instrucciones para un objeto en solitario.

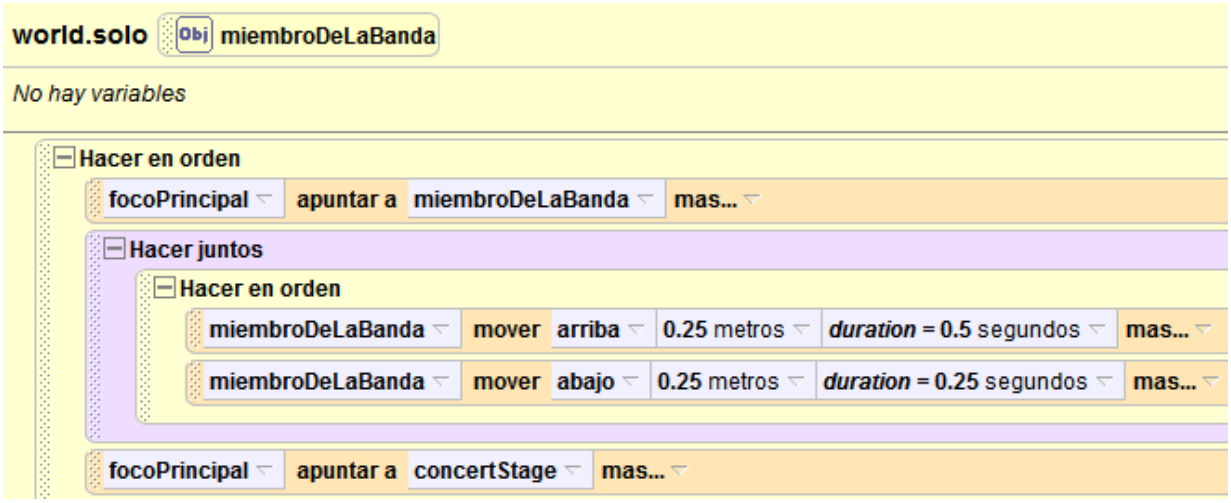


El segundo paso consiste en alterar las instrucciones para que en lugar de la **abeja reina** aparezca el nombre del parámetro **miembroDeLaBanda**, para permitir que éste actúe en lugar de ella. La Figura 32 ilustra el proceso.

Figura 32. Arrastrando un parámetro hacia su ubicación en el código.



Todas las posiciones que aparecen enmarcadas de color amarillo corresponden a **posibles destinos válidos para arrastrar el parámetro**. El código final luce así:



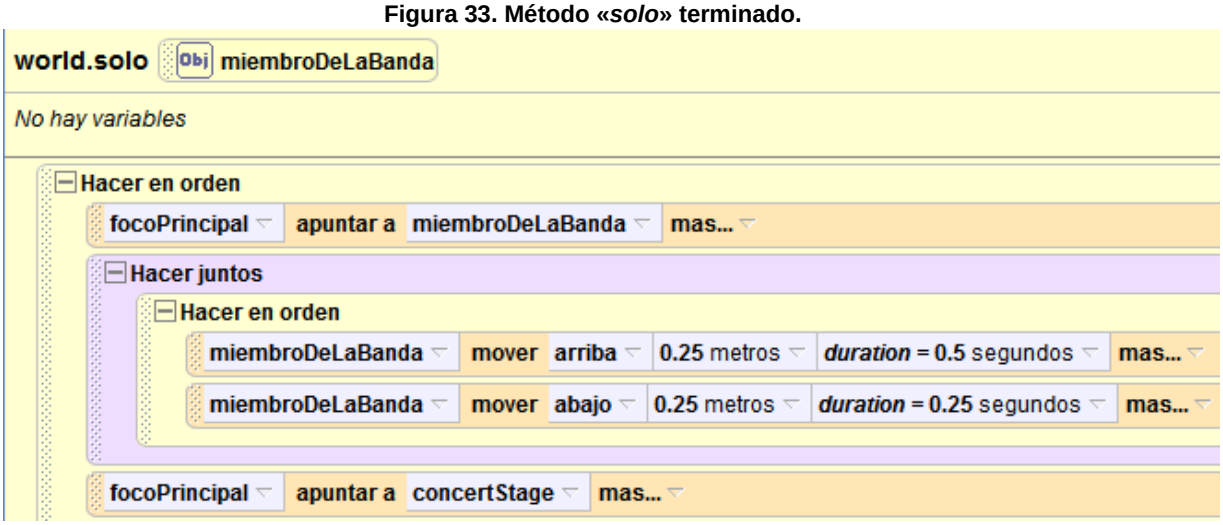
La importancia del tipo de parámetro

Vuelva a revisar la el aspecto que presentan las instrucciones que aparecen en la Figura 32, cuando un recuadro de color verde rodea al espacio donde se va a colocar algo. El color es una ayuda que nos proporciona Alice para que la selección de algún elemento sea la correcta en el espacio donde deseemos colocarlo.

En este ejemplo, un miembro de la banda se va a utilizar como un marcador de posición para un objeto en una instrucción de movimiento. Pero, **miembroDeLaBanda** no debe utilizarse como un marcador de posición para la distancia (0,25 metros) o duración (0,5 segundo), ya que estos son valores numéricos, no objetos.[17]

El último paso en el guión es la atención del público, lo que se materializa a través de una instrucción que obliga a que el foco apunte a la gente. En esta instrucción, **concert_stage.crowd** pasa a ser el destino para la acción ***apuntar a*.** El código resultante se muestra en la Figura 33.

Figura 33. Método «*solo*» terminado.



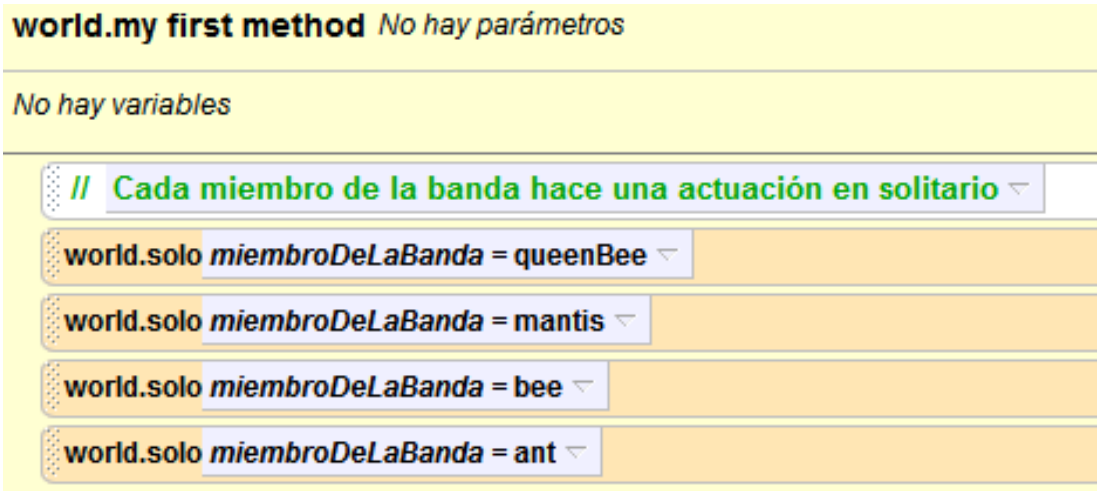
Prueba con argumentos

Para probar el método **solo**, debemos llamarlo desde **my first method**, de tal manera que cuando ***solo*** se arrastra, la interfaz de Alice pedirá la selección de algún objeto de tipo **miembroDeLaBanda** para que el método opere como se espera.

Para estar seguro que el método ***solo*** funciona para cada miembro de la Banda, se escribieron cuatro declaraciones, como se observa en la Figura 34. En la primera llamada, el método solo será llevado a cabo con el integrante abeja reina, mientras que en la segunda el valor será igual a mantis; la tercera se llama abeja obrera, y finalmente hormiga.

[17] Es algo que de forma coloquial se expresa como: «*peras con peras, y manzanas con manzanas*».

Figura 34. Un método invocando a otro método, con distintos parámetros.



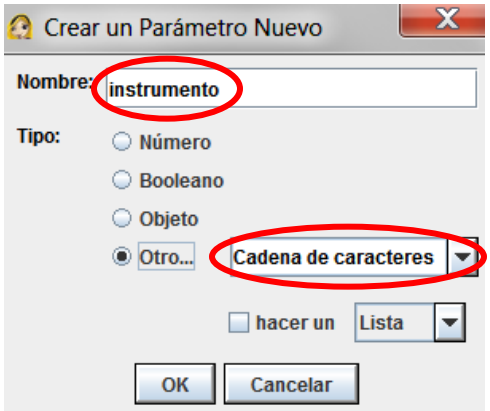
En Ciencias de la Computación, el valor de un método llamado se conoce como **argumento**. En este ejemplo, la abeja reina, mantis, abeja obrera, y hormiga pasan a ser el argumento que se va a utilizar para cada una de las llamadas. De esta manera, el método sólo es algo *genérico* que puede ser utilizado con diferentes argumentos para llevar a cabo la misma tarea con distintos objetos.

Varios parámetros

Usted notará que con el código anterior no se ha completado la animación. En cada una de las actuaciones, el miembro de la banda no sólo debe moverse, sino también debe tocar un instrumento musical.[18]

Una sola instrucción se necesita en el método para reproducir un sonido pero cada instrumento musical debe tener un audio diferente. En términos de lo que se ha diseñado, el sonido del bajo debe ser para la abeja reina, un saxofón para la mantis religiosa, batería para la abeja obrera, y la guitarra para la hormiga. Evidentemente, más de un parámetro sería útil en esta situación. Para ilustrar el uso de múltiples parámetros, podemos añadir otro que sea de distinta naturaleza.

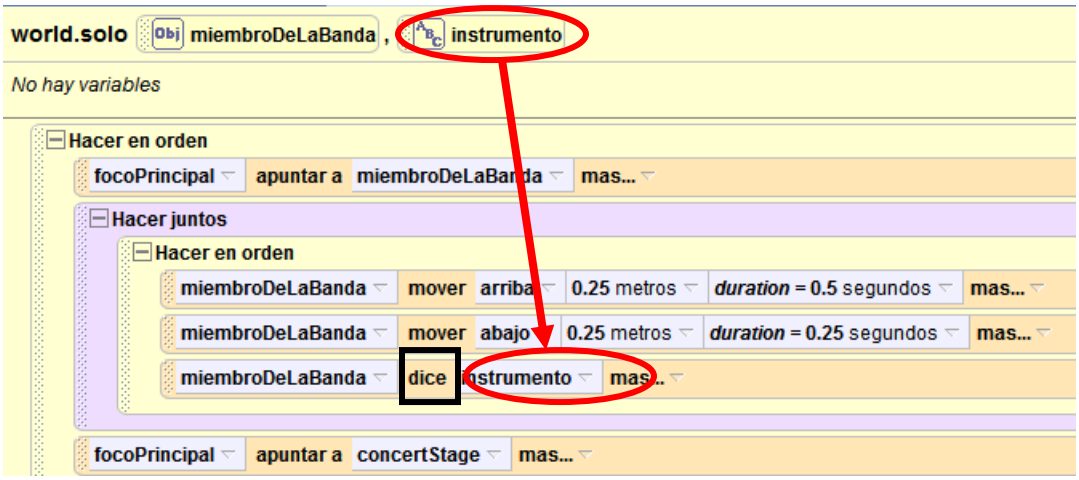
El tipo de este parámetro será de **texto, o cadena de caracteres**, como se muestra en la imagen adjunta, que tendrá la misión de indicar cómo se llama el instrumento que cada integrante vaya a tocar. El nombre será igual a **instrumento**, que formará parte de la lista de parámetros del método **solo**.



El método **solo** lo completamos pidiendo que mientras el integrante de la banda se presente en solitario, diga cómo se llama el instrumento que está tocando, a través del

[18] Por razones obvias, se recomienda tener encendido los altoparlantes del computador para poder ejecutar las pruebas sonoras que vendrán a continuación. En caso que tuviera algún problema , puede reemplazar la ejecución de algún instrumento por algún globo de texto.

método **dice**. El método **solo** se muestra a continuación:

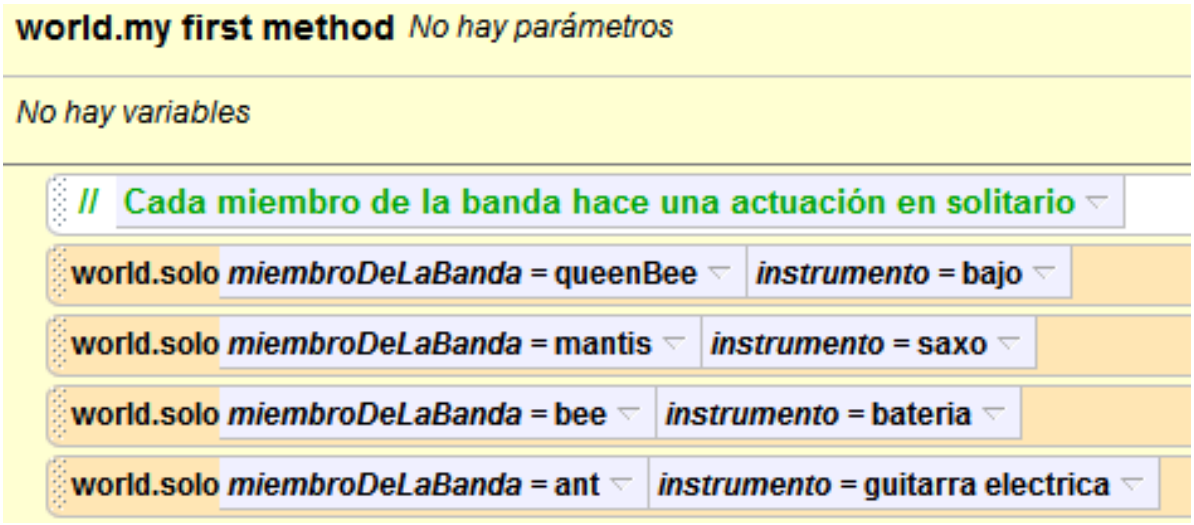


Por lo tanto, cuando se llame al método **solo** desde **my first method**, será necesario pasar dos argumentos:

1. Un objeto, que corresponde al bicho que le toque actuar en ese momento.
2. Un texto, que corresponde al nombre instrumento que el bicho tenga que interpretar.

Una posible invocación del método **solo** podría ser la siguiente:

Figura 35. Llamando a un método que requiere de un parámetro.



Nota técnica

Todos los ejemplos de este apunte se han estado haciendo a través de la traducción al español de Alice, que tiene el problema de la mala codificación de nuestros símbolos especiales: tildes, ñ, o la doble puntuación por encima de una vocal, como en el caso de pingüino. Por lo mismo, se han omitido de manera intencional, porque en **tiempo de ejecución**, no se leen bien.

Otros tipos de parámetros

En el ejemplo de la banda Bichos se ilustraron dos tipos de parámetros: objetos y texto. Pero un parámetro puede también ser un **número**, un **valor booleano** — verdadero o falso — un **color** — rojo, azul, verde, etc. —, o de otros tipos.

Cada uno de estos tipos de valores contribuye a enriquecer las posibilidades de programación, sobre todo los de tipo numéricos, que son uno de los más comunes. Veamos un ejemplo al respecto.

Ejemplo

Un mago realiza una ilusión de levitación, en la cual los objetos parecen surgir mágicamente en el aire. El mago señala con una varita mágica a su asistente y ella se levanta suavemente en el aire y luego flota de regreso a su posición original en la mesa. Entonces, el mago realiza el mismo truco con el conejo. El conejo, siendo un objeto más ligero, flotará en el aire un poco más alto que el asistente del mago. La escena inicial se ilustra en la imagen que acompaña a este párrafo.

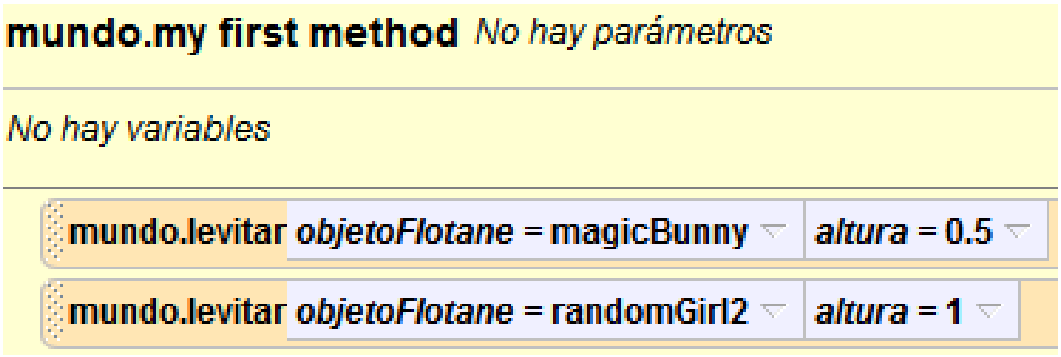
Debido a que al asistente del mago y el conejo, se le debe hacer levitar de la misma manera, la animación puede ser implementada como un método único si utilizamos un parámetro para comunicar qué objeto es el que debe flotar. La altura del objeto para flotar podría ser también pasada como argumento al método, por lo que los dos parámetros son necesarios, uno un objeto — que le vamos a llamar **objetoFlotante** —, y un segundo que se denominará **altura**. Un posible pseudocódigo o guión textual se muestra a continuación.

```
Parámetros: objetoFlotante, altura
Hacer en orden:
     1. El brazo derecho del mago apunta hacia objetoFlotante
     2. objetoFlotante se mueve hacia arriba altura metros
     3. objetoFlotante se mueve hacia abajo altura metros
```

Se pide que usted cree los parámetros **objetoFlotante** y **altura**. Un posible código final es el que se muestra en la Figura 36:

Figura 36. Método que tiene dos parámetros de distinto tipo.

Las llamadas a la levitación están escritas en **my first method**, tal como se muestra a continuación:



Nota técnica.

Cuando esta animación se ejecute, el resultado es bastante sorprendente. El conejo flota hacia arriba y hacia abajo justo como se esperaba. Pero, el asistente del mago se mueve horizontalmente en lugar de flotar hacia arriba en el aire. Por supuesto, el asistente del mago se reclina sobre la mesa, por lo que el método **mover** hacia **arriba** no es exactamente lo que teníamos en mente.

Para resolver este problema de orientación, las instrucciones *mover* y *flotar* son revisados para utilizar la opción **asSeenBy**, que nos permite seleccionar una dirección basada en la perspectiva de un objeto diferente. En este ejemplo, hemos seleccionado el suelo como referencia, como se muestra en la Figura 36.

Ejercicios

Recordatorio. Asegúrese de añadir comentarios a los métodos para documentar lo que el método hace y qué acciones se llevan a cabo por las secciones de código dentro del método.

1 Rana escapando de un depredador.

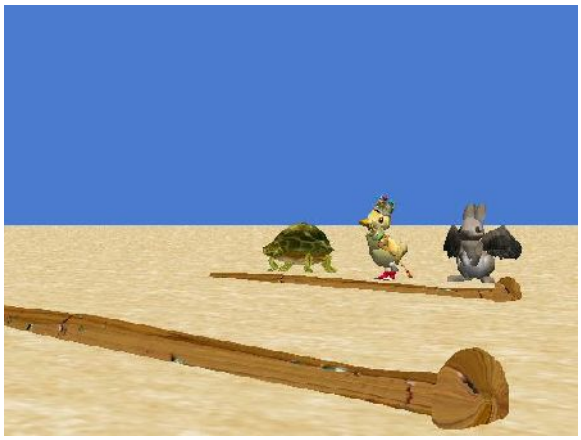
En el estanque de lirios, las ranas disfrutan saliendo del agua de vez en cuando para calentarse en el sol. Por supuesto, las ranas se ponen un poco nerviosas cuando un depredador se avista. En este buen día, una hambrienta serpiente vaga en la escena. Se le pide crear un escenario similar al de abajo y mostrar una animación de las ranas saltando en el estanque cuando se acerque una serpiente.

Escriba un método en el cual la serpiente apunte a una rana, y que luego se arrastre hacia ella. Después, la rana tiene que arrancar hacia en el estanque y saltar dentro, para sentirse a salvo. Su método se debe usar un parámetro para especificar cuál será la rana que escape.

2 Carrera.

Este es un curioso tipo de carrera, porque se sabrá quién va a ganar la carrera antes que comience. Se le recomienda de manera encarecida resolver este problema, ya que éste será la antesala de mundos más complicados que vendrán a medida que avancemos con los contenidos.

Se le pide crear un mundo con tres personajes alineados como si estuvieran en la raya de partida de una carrera. Añada otros objetos en el mundo que puedan servir como líneas de salida y de llegada, como se muestra más abajo:



Cree un método que haga que simultáneamente cada personaje se mueva hacia adelante para pasar la línea de meta. Utilice tres parámetros — uno para cada uno de los personajes — que especifique la cantidad de tiempo que cada uno de ellos vaya a tardar en completar la carrera.

3 Dragones.

Cuenta la leyenda que los dragones son parientes lejanos de los pollos. Por lo tanto, no nos sorprende que el pasatiempo favorito de los dragones sea un juego de «*pollo*». La escena muestra un mundo con cuatro dragones, cuidadosamente colocados en un patrón similar a un diamante, por lo que cualquier dragón está a la misma distancia de los otros.

Cree una simulación en la que dos dragones se enfrentan entre sí y se levantan para volar a una altura ligeramente mayor que la del nivel del suelo. La idea es que se intercambien de posición. Su simulación debe utilizar un método llamado **vueloDelDragon**, que tiene

cuatro parámetros: los dos dragones seleccionados, y las alturas de vuelo para cada uno de ellos. La imagen que acompaña a este párrafo proporciona un esbozo de cómo podría ser la escena inicial.

4 Movimiento de ruedas.

Este ejercicio es para que usted pueda trabajar con los vehículos de motor que tienen ruedas. El reto consiste en que el movimiento de las ruedas sea diferente que el movimiento general de los vehículos de motor. Las ruedas deben girar mientras el coche se mueva hacia adelante. Para obtener una parte de la apreciación de este tipo de animación, cree un mundo simple con un automóvil y un camión. Además, considere dos métodos:

- **rotacionDeRueda**, que debe recibir como parámetro una de las ruedas y girar una revolución; y
- **movimientoDeVehiculo**, que debe mover hacia adelante uno de los vehículos de motor, mientras las ruedas estén girando. El parámetro que reciba debe ser el automóvil o el camión.

Resumen

Hasta el momento, en esta unidad hemos estudiado cómo escribir nuestros propios métodos y cómo utilizar los parámetros para lograr la comunicación con ellos. En particular, nos hemos centrado en mundos que tienen objetos que han de interactuar entre sí, a través de métodos parametrizables. Una de las ventajas de usar métodos es que el programador puede pensar en una colección de acciones como si sólo fueran una sola, en algo que en Ciencias de la Computación se conoce con el nombre de *abstracción.* También, los métodos hacen que sea más fácil depurar nuestro código. Los comentarios fueron utilizados para documentar los métodos, y así otorgarle un mayor poder de legibilidad. También se indicó que su uso está considerado dentro de las buenas prácticas de programación.

Hemos utilizado los parámetros para organizar la transferencia de los valores de un método a otro. En un método, un parámetro actúa como una caja que va a recibir un valor en particular, que recibe el nombre de *argumento*.

Los ejemplos presentados hasta el momento incluyen objetos, sonido, cadenas y parámetros numéricos. Los parámetros permiten que los métodos sean escritos de forma genérica. El método puede ser llamado con distintos argumentos para llevar a cabo la misma tarea con valores diferentes.

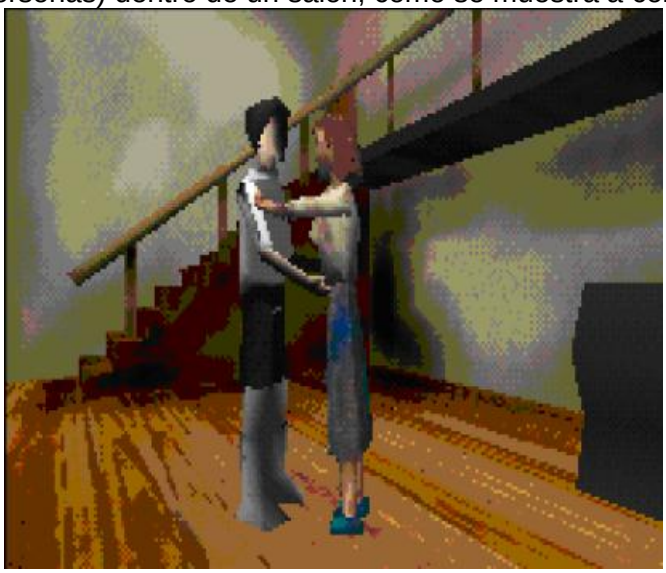
Conceptos importantes vistos en la unidad

- Un método es una colección bien diseñada de instrucciones que se llevarán a cabo cuando se necesite.
- Los métodos definen los comportamientos de un objeto.
- Los métodos que implican acciones para más de un objeto tiene una perspectiva más global, y se les denominan *métodos de mundo*. En Alice, empiezan con el nombre *World.«algo»*
- Para ejecutar un método, el método debe ser llamado o invocado.
- Los parámetros se utilizan para la comunicación entre métodos.
- Un parámetro puede ser declarado para representar el valor de un tipo en particular:
  - Objeto;
  - Booleano;
  - Número;
  - Sonido;
  - Color;
  - Cadena de caracteres, etc.
- En una llamada a un método, el valor de un parámetro de un método es un *argumento*.

Proyectos

1 Danza.

El objetivo de esta animación es que la pareja realice un paso de baile tradicional en un cuadrado, como el que se usa en el vals y en otras danzas. Cree una escena con un hombre y una mujer (personas) dentro de un salón, como se muestra a continuación.

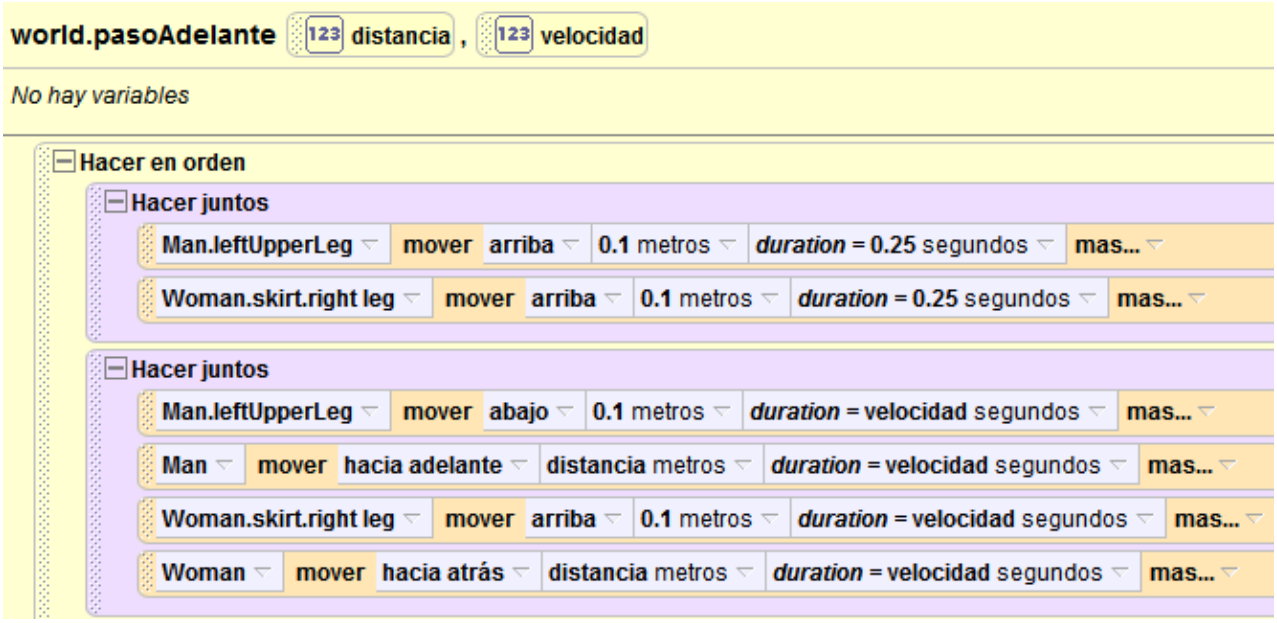


En el primer paso de una escena, el hombre da un paso hacia adelante, lo que hace con su pierna izquierda, al mismo tiempo que la mujer da un paso hacia atrás, que realiza con su pierna derecha. Esto no es tan sencillo como parece. Una manera de hacer que un personaje mueva las piernas para que parezca hacer un «*paso*» es elevar una pierna una pequeña cantidad y, a continuación, mover la pierna hacia adelante como cuando se mueve de arriba hacia abajo. Luego, la otra pierna realiza una acción similar. Así, pues, para que dos personajes parezcan bailar uno con el otro, requiere coordinar la elevación de la pierna, mover y la acción de la caída para ambos personajes. Un posible guión para el primer paso se muestra a continuación.

Método: pasoAdelante
Parámetros: *distancia, velocidad*

```
Hacer en orden:
      1. Hacer juntos:
            a) La pierna izquierda del hombre debe subir a
            velocidad.
            b) La pierna derecha de la mujer debe subir a
            velocidad.
      2. Hacer juntos:
            a) Hombre avanza hacia adelante distancia en
            velocidad.
            b) La pierna izquierda del hombre se mueven hacia
            abajo en velocidad.
            c) Mujer se mueve por detrás a una cierta distancia
            en velocidad.
            d) La pierna derecha de la mujer debe bajar a cierta
            velocidad.
```

Escriba un método denominado **pasoHaciaAdelante**, que se define con dos parámetros: *distancia*, que especifica la distancia de avanzar, y *velocidad* que especifica la duración del movimiento de avance, como se muestra a continuación:



Después del método *forwardStep*, realices tres pasos:

1. **pasoDerecho**, donde el hombre y la mujer tomar un paso hacia la derecha de él, y la izquierda de ella.
2. **retroceso**, donde el hombre se da un paso hacia atrás, lo hace con la pierna izquierda y que de forma simultánea la mujer haga un paso hacia delante con pierna derecha.
3. **pasoIzquierdo**, cuando la pareja realiza un paso hacia el otro lado: a la izquierda del hombre y a la derecha de ella.

Tendrá que experimentar con la cantidad de movimientos que tendrán que hacer las piernas para que el baile resulte ser lo más armónico posible. A continuación, deberá crear un método denominado **giro**, para que el hombre gire alrededor de la mujer.

Si los cuatro pasos se han ejecutado correctamente en secuencia, la pareja se mueve formando un cuadrado, como en la pista de baile. Cree un método para llamar a todos los métodos con el fin de animar a la pareja a realizar una figura seguida por un giro para un baile. A continuación, cree un segundo método para animar la realización de una figura diferente de danza, pidiendo los pasos de baile en un orden diferente.

2 Balón mano.

Cree un mundo con una mano derecha sosteniendo un balón. Haga que los dedos estén cerca para agarrar la pelota. Después, lance la pelota en el aire, mientras la mano abre los dedos Por último, la mano captura la pelota antes que la mano vuelva a cerrar sus dedos. Como sugerencia, revise el apartado sobre propiedades de un vehículo, que le ayudará a tomar la pelota, mientras la mano se mueve.

3 Ra maneja un bote.

Cree un mundo con un bote, una persona sentada dentro de él, una isla, y un muelle situado a 25 metros de la isla. En el mundo que se muestra a continuación, el egipcio Ra está sentado en un bote. Diseñe un método para que el bote que maneja Ra esté a 25 metros esté cerca del embarcadero. Una de las formas sugeridas para hacer esto sería la de crear los métodos:

remarIzquierda, remarDerecha, controlarTorsoYCabeza — para controlar la espalda y la cabeza — y **empezarARemar y detenerElRemaje**, con el fin de poner el cuerpo de Ra en posición de remo.

4 Robot que ordena una habitación.

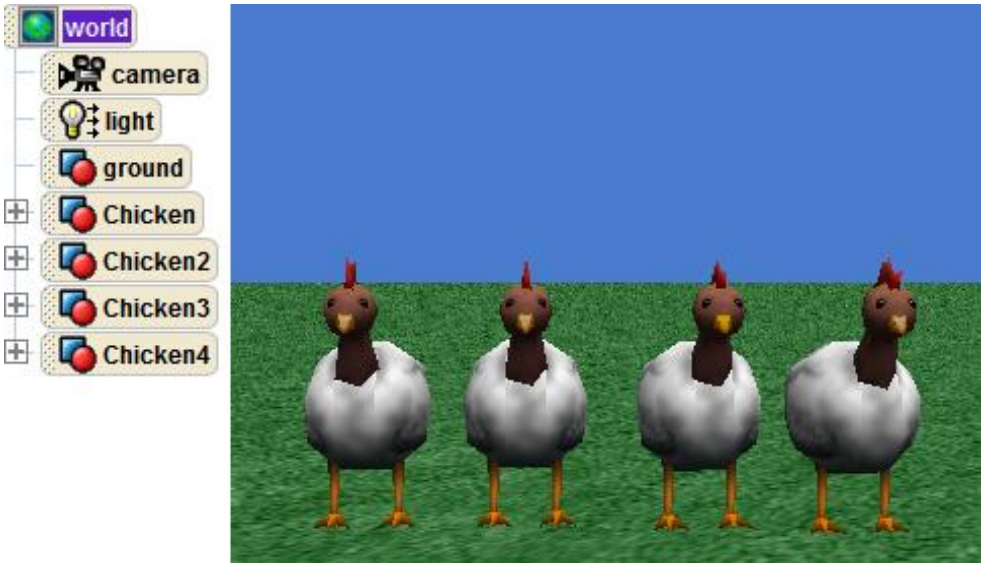
Alice recibió un robot gorila como regalo. Ella está tratando de encontrar la manera de programar el robot para ayudarla a ordenar las cosas de su habitación, que es una tarea que tiene que hacer cada sábado por la mañana.

En la escena inicial de esta animación, el robot está parado en el medio de la habitación de Alice cerca de varios objetos esparcidos por el suelo, como por ejemplo una barra de pesas, una piñata y un oso de peluche. El robot debe estar ubicado entre todos los objetos de manera que pueda ser fácilmente manipulado durante la animación, sin que se aleje de los demás objetos. Escriba un programa para enseñarle el robot que recoja de un objeto a la y lo reordene. Escribir dos métodos que se llamen *tomar* y *soltar*, cada uno de los cuales tienen un parámetro que debe identificar el nombre del objeto que va a recoger o soltar. El método de recolección debe hacer que el robot recoja un objeto en la mano. El método para soltar debe hacer que el robot ponga el objeto en algún lugar distinto de donde comenzó. Se recomienda que el robot gire un cuarto de revolución a la izquierda o la derecha antes de poner el objeto abajo.

2.3 Consejos y Técnicas

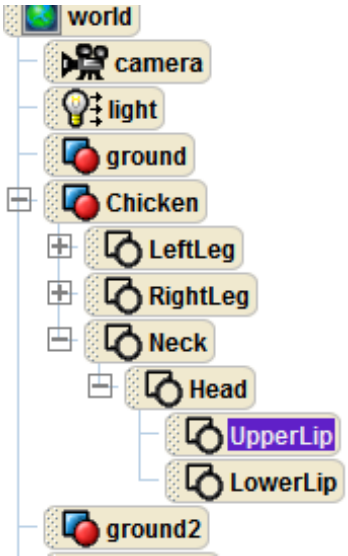
Renombrar un objeto

En algunas animaciones varios objetos del mismo tipo pueden estar dentro de la escena al mismo tiempo. Considere la escena de la figura adjunta, en la cual tenemos cuatro pollos, que Alice automáticamente nombra como Chicken, Chicken2, Chicken3, Chicken4. Cada una de estas denominaciones sirve para identificar un objeto de otros en el mundo. Por supuesto, que esto se puede cambiar, para que los nombres se ajusten de mejor manera a lo que queramos conseguir.



Para cambiar el nombre:

1. Haga clic con el botón derecho del ratón en el objeto que alterar, que está dentro del árbol de objetos.
2. Seleccione la opción **Cambiar el nombre** en el menú emergente.
3. Introduzca el nuevo nombre en el cuadro de diálogo que aparece.
4. Asegúrese de pulsar la tecla **Enter** después de escribir el nuevo nombre.



Propiedades del color

Con sólo mirar los pollos en la escena, es imposible determinar cuál es el pollo2. Una solución sería cambiar la apariencia de cada pollo de alguna manera, de tal modo que cada uno pueda ser fácilmente identificado.

Un posible cambio es la propiedad color del labio superior de cada pollo. Para esto, seleccione el labio superior del pollo que quiera modificar[19], como se ve en la imagen de la derecha.

[19] Recuerde que el labio forma parte de la cabeza, y que ésta a su vez se encuentra en la cabeza.

A continuación, altere el color en la ficha de **propiedades** correspondiente, tal como lo hizo en el mundo de los muñecos de nieve. De esta manera, al cabo de tres pasadas más, los pollos serán fácilmente identificables.

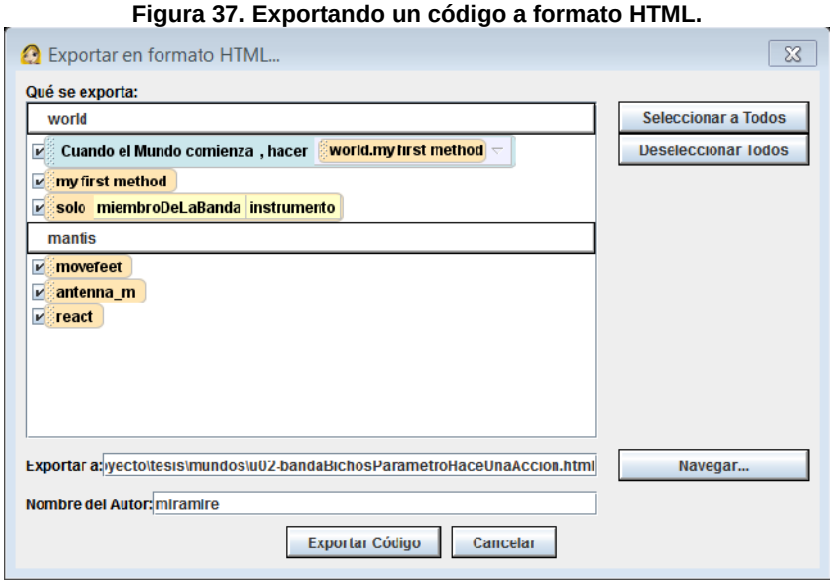
Esta técnica de identificación será muy útil en futuros mundos que vayamos construyendo. Si queremos saber que algo ha sucedido en el curso de alguna animación, podemos cambiar algunas propiedades del objeto, como el **color** o el **tamaño**.

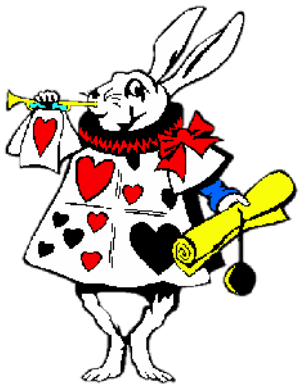
Exportar el código fuente del programa

A menudo es útil exportar el código fuente del programa completo, o de alguno de los métodos a formato HTML, con el fin de poder compartirlos con otras personas a través de la WWW. Para esto:

1. Haga clic en el menú **Archivo**.
2. Seleccione **Exportar el código para imprimirlo**.
3. Elija el método, la ruta donde quiera guardar el HTML, seleccione lo que quiera guardar, y para terminar, presione el botón **Exportar Código**.
4. El **Nombre del autor** es **obligatorio.** La Figura 37 ilustra esto:

Figura 37. Exportando un código a formato HTML.





Segunda Parte

Métodos de nivel y Herencia

Las galerías de modelos 3D nos provee la opción de emplear un diverso y bien diseñado conjunto de **clases de objetos** para rellenar y crear un escenario en un mundo virtual. Como ustedes saben, cada uno de los objetos agregados a un mundo viene con un conjunto predefinido de acciones que puede realizar: mover, girar, rodar, cambiar el tamaño, por nombrar algunos. Después de escribir varios programas, es natural pensar en ampliar las acciones que un objeto pueda realizar.

De hecho, podemos añadir funcionalidades a cualquier personaje con un método diseñado específicamente para este tipo de objeto. En la programación orientada a objetos, un método que se usa para agregar funcionalidad a un determinado tipo de objeto se denomina **método de clase**.

En Alice, utilizamos el término objeto de nivel, porque pensamos en un determinado tipo de objeto como un personaje modelado de forma tridimensional. Esta segunda parte de la unidad se centrará en métodos de nivel de objetos y que denominaremos también como **métodos de nivel de personajes**. Estos métodos son un poco especiales porque podemos guardar el personaje junto con su nuevo método, como si fuera un nuevo tipo de objeto. Los nuevos tipos de objetos aún no saben cómo realizar todas las acciones del modelo original. En lenguajes orientados a objetos, diríamos que los nuevos tipos de objetos **heredan** todas las propiedades y acciones de los objetos originales, que pertenecen a una **clase base**. Una vez que un nuevo objeto se ha creado, se puede utilizar más tarde en algún programa diferente, lo que permite reutilizar el código en otras oportunidades.

2.4 Creación de nuevos personajes

En la parte anterior, a nivel de mundo los métodos fueron escritos para animar a más de un objeto en una escena, donde ellos interactúan entre sí a través de alguna manera. A partir de ahora, nos centraremos en los en los métodos que operan a nivel de objeto. Una diferencia importante entre los métodos de nivel de mundo y los métodos de nivel de objeto, es que este último está diseñado específicamente para un cierto tipo, de tal manera que la nueva creación se pueda guardar para dar lugar a un nuevo personaje.[20]

[20] Adoptaremos la convención de decir que objeto es sinónimo de personaje.

Un método de nivel de personaje

Considere la patinadora que se muestra en la escena de invierno, que se muestra en este párrafo. Queremos que ella realice acciones típicas de patinaje artístico, para lo cual sería conveniente que la patinadora ya supiera como patinar hacia adelante y hacia atrás, junto con la realización de movimientos de giro y de eje. Pero, como los otros personajes de la galería, ella sólo sabe cómo realizar simples movimientos, que pueden ser los primitivos mover o girar.

Por lo tanto, vamos a escribir algunos de los métodos para enseñar a la patinadora cómo realizar movimientos más complejos. Partiremos con un método para que ella realice un movimiento hacia adelante.

Los movimientos de patinaje son acciones complejas que requieren varias instrucciones que implican movimiento de diversas partes del cuerpo. Un posible guión textual para un movimiento hacia adelante se muestra a continuación.

Tabla 7. Algoritmo de patinaje.

```
Hacer juntos:
      Hacer en orden:
            1. Deslizar hacia la izquierda:
                  1. Elevar la pierna derecha y haga girar la parte
                  superior derecha del cuerpo.
                  2. Baje la pierna derecha, y haga que el cuerpo
                  quede erguido.
            2. Deslizar hacia la derecha:
                  1. Levantar la pierna izquierda y hacer girar la
                  parte superior izquierda del cuerpo.
                  2. Bajar la pierna izquierda, y retornar el cuerpo
                  erguido.
            3. Desplazar el cuerpo completo hacia adelante 2
            metros.
```

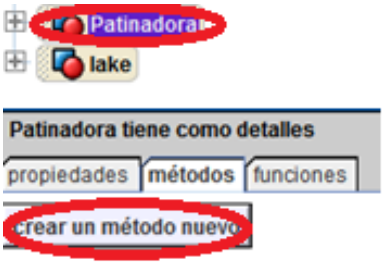
El guión de la acción de patinar en hielo tiene dos segmentos: deslizar la pierna izquierda, y deslizar la pierna derecha. Acá hemos aplicado la técnica de «*dividir para conquistar*», ya que hemos separado la gran tarea en dos subproblemas que abordaremos de manera separada. Para deslizar a la izquierda, la pierna derecha se levanta y el cuerpo se da vueltas o enrolla hacia la derecha. Después, la pierna derecha se baja y el cuerpo se pone en posición vertical, o erguido. De forma análoga se definió el caso del deslizamiento hacia la derecha. Ambos tipos de deslizamiento se realizan al mismo tiempo para que el cuerpo se mueva hacia adelante.

Este método está diseñado específicamente para la patinadora de hielo y no implica ningún otro tipo de objetos. De esta manera, el método que acabamos de revisar está formulado de forma exclusiva para un personaje en particular, de tal manera que lo conversado hasta el momento recae en la categoría de método de

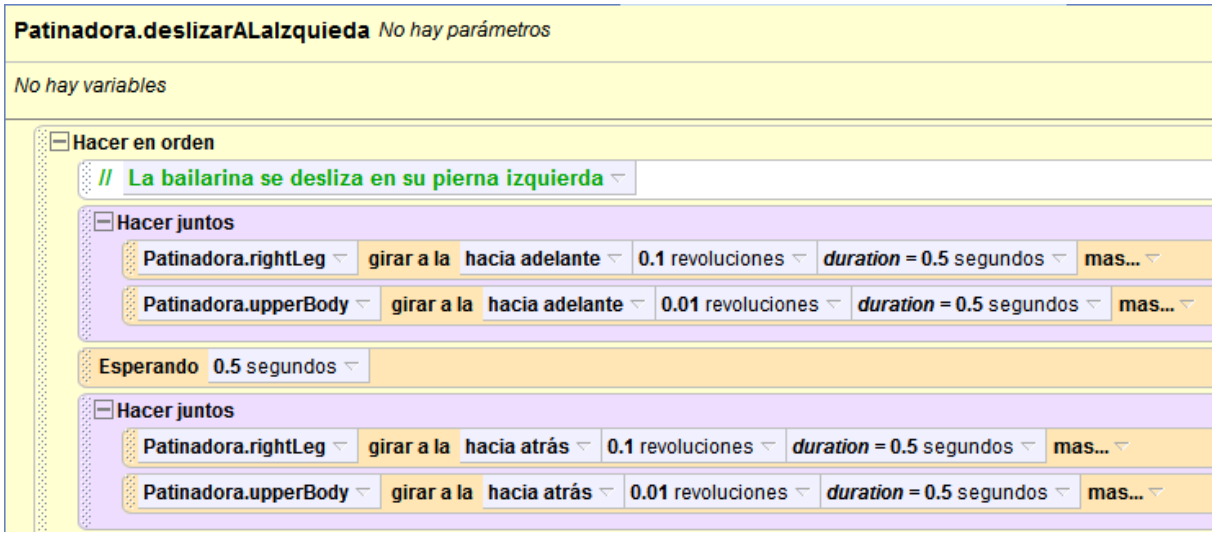
nivel de personaje.

Entonces, para crear un método de nivel de objeto, nos dirigimos al árbol — de objetos, valga la redundancia —, y seleccionamos la opción **create new method**, como lo ilustra la Figura 38. Después, le damos el nombre que queramos, y nos ponemos a escribir las instrucciones en el editor de código.

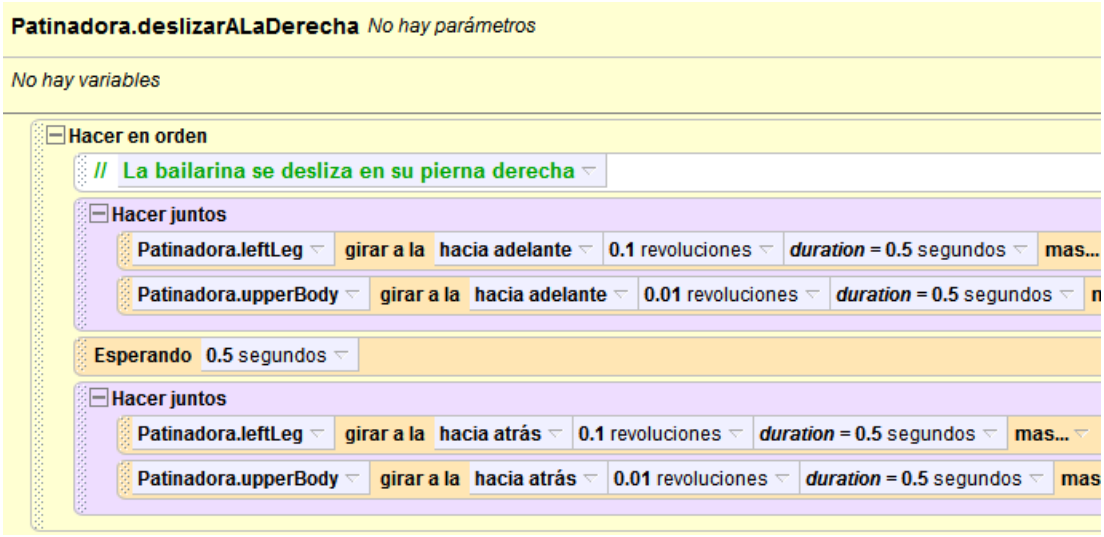
Figura 38. Primer paso para crear un método de nivel de objeto.



Al igual como lo hacíamos con los métodos de mundo, en los de objeto — o de personaje — procedemos de la misma manera. En primer lugar vamos a crear uno que se llame **patinar**, que lo vamos a descomponer en dos etapas: (1) deslizamiento hacia la izquierda; y (2) deslizamiento hacia la derecha. Al momento de crearlo, note que ahora Alice no habla de **World.patinar**, sino de **Patinadora.patinar**, lo que indica que lo que estamos construyendo es un método de nivel de objeto. Es importante que usted no se deje intimidar por lo complejo que a lo mejor pudiera parecerle el código del método **deslizarALaIzquierda**, donde el cuerpo de la bailarina adopta la actitud de ir hacia adelante, gracias a la palanca que hace con su pierna izquierda. Las acciones se muestran a continuación:

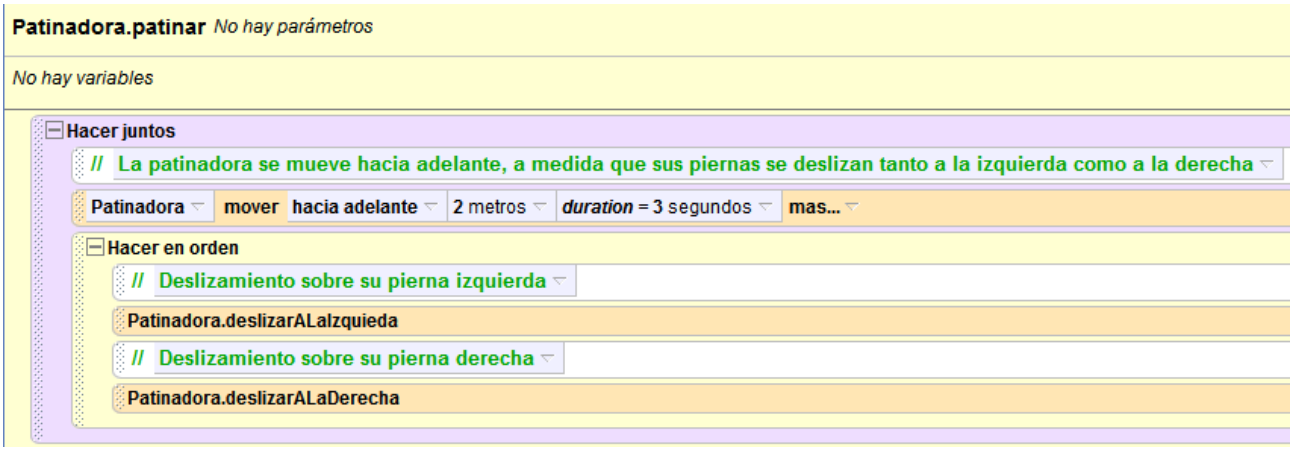


Entre una y otra se ha dejado una corta espera para permitir que la patinadora avance, que se logra con la instrucción **Esperando**. A continuación, la pierna derecha se encoge, el cuerpo gira y el cuello se devuelve a su posición original, con el fin de preparar el movimiento de la otra pierna. El código para implementar el método **deslizarALaDerecha** es bastante similar:



Ya que hemos trabajado por separado los métodos para ir a la izquierda y a la derecha, los combinamos en el método **patinar**, tal como se muestra en la Figura 39:

Figura 39. Ejemplo de método de objeto que invoca a otros métodos de objeto.



Las acciones de **mover hacia adelante** con los **deslizamientos** se hacen en paralelo para permitir que la bailarina efectivamente avance al mismo tiempo que hace las contorsiones en el hielo. Ahora vamos a escribir un segundo método para hacer que la patinadora de hielo efectúe un giro. El pseudocódigo para un giro podría ser el que sigue:

Tabla 8. Algoritmo para implementar el giro de la bailarina.

```
Hacer en orden:
      1. Hacer al mismo tiempo:
            a) Posicionar la pierna izquierda hacia arriba,
```

```
        para que la bailarina se prepare para el giro.
             b) Levantar brazos hacia los lados.
        2. Hacer al mismo tiempo:
             a) Doblar la pierna (a la altura de la rodilla)
        hacia dentro y hacia afuera durante el giro.
             b) Bajar los brazos de manera gradual mientras se
        produzca el giro.
             c) La bailarina gira sobre su eje seis veces.
        3. Baje la pierna izquierda hacia atrás, para volver a
    la posición inicial
```

La implementación del método de giro queda como ejercicio para el lector.

Guardar un nuevo personaje

La patinadora de hielo ahora tiene dos nuevos métodos: **patinar** y **girar**. La escritura y las pruebas de los métodos tomaron algo de tiempo y esfuerzo para lograrlo. Sería una lástima no poder usar esto en otro programa de animación que podamos crear más tarde. Esta es la razón por la que desea guardar un objeto y sus flamantes nuevos métodos como si fuera un nuevo objeto, o un nuevo personaje. Por lo tanto, guardar la patinadora de nieve con los métodos patinar y girar como un nuevo tipo de personaje es exactamente lo que queremos hacer, de tal manera que podamos reutilizarla en otros mundos y no tener que escribir estos métodos nuevamente en otro programa de animación.

Guardar un objeto nuevo como un nuevo personaje es un proceso de dos pasos. El primero consiste en cambiar el nombre del objeto, porque la idea es guardar en un archivo de Alice un modelo 3D que sea diferente al de la bailarina original. Para estos efectos, podemos llamarle **PatinadoraInteligente**.

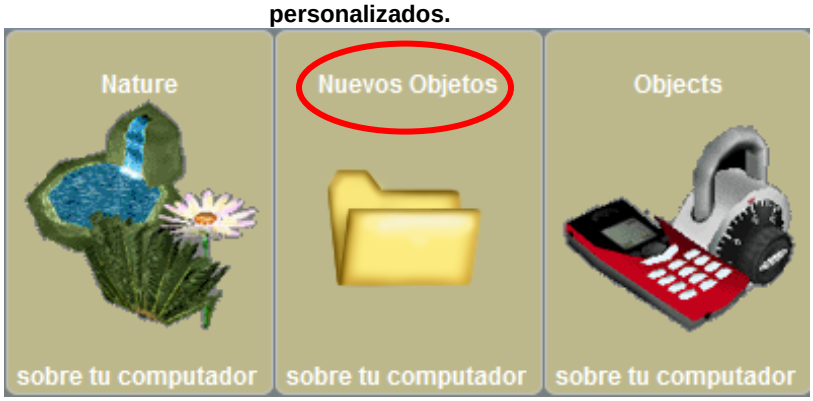
El segundo paso consiste en guardar el objeto **PatinadoraInteligente**. Para eso, haga clic con el botón derecho en el árbol de objetos en el nuevo personaje, y marque la opción **guardar el objeto**. Después, seleccione el directorio donde lo quiera dejar. En la práctica, se recomienda dejarlo en esta ruta:

```
<directorio de Alice>/Required/gallery/English/NuevosObjetos
```

En este caso, **NuevosObjetos** es el nombre de una carpeta que podemos crear para almacenar nuestras creaciones. Aunque el nombre sea arbitrario, se recomienda que no contenga caracteres especiales como tildes o eñes.

Una vez que el objeto se haya guardado como un nuevo tipo de modelo en 3D, puede ser utilizado en cualquier mundo mediante la **importación** del archivo. En este ejemplo, el objeto **PatinadoraInteligente** será *casi igual* que la patinadora convencional, con la diferencia que ahora ella sabe patinar y girar ☺. Si usted va a la galería, encontrará la carpeta que se ha construido para estos fines:

Figura 40. Incorporación de un directorio creado por el usuario que contiene objetos personalizados.



2.5 Herencia

En los lenguajes de programación como C++, Java o C#, la creación de una nueva clase de objetos se denomina **herencia**. La idea básica que usted debe manejar es que esta característica de la programación orientada a objetos permite agregar funcionalidad a los objetos que se encuentran en la galería de personajes, para definir comportamientos adicionales del nuevo objeto.

Por ejemplo, recuerde el objeto **PatinadoraInteligente** de la sección anterior: ella hereda todas las características de una *Patinadora* convencional, pero cuenta con el agregado del nuevo par de métodos que se definieron para ella, que no tiene su **clase madre**: **patinar** y **girar**.

Beneficios de la herencia

La herencia es considerada uno de los puntos fuertes de lenguajes orientados a objetos, debido a que admite reutilización de código. Lo que ocurre es que muchos programadores se dedicaban a reescribir el código que habían diseñado antes. Lo bueno de la herencia es que le permite al programador escribir código una vez y volver a utilizarlo más tarde en un programa diferente, sin tener que empezar todo desde cero.

Una ventaja importante de la creación de nuevos personajes en Alice es que nos permite compartir código con otros proyectos que pudiera estar desarrollando en equipo, porque cada persona puede escribir métodos para cada uno de los personajes del mundo virtual, y guardar lo que haya hecho. A nivel de grupo, se puede trabajar con un solo mundo cuyos personajes hayan sido los aportes individuales de cada uno de sus miembros.

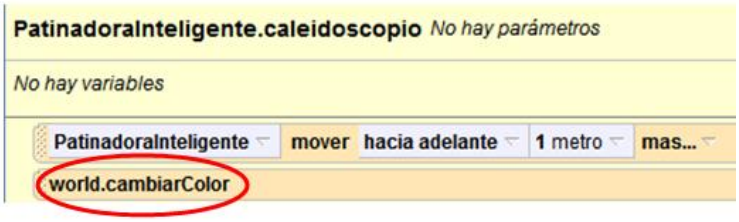
Algunas pautas para escribir métodos de nivel de objetos

Los métodos de nivel de personaje — o de objeto — son una poderosa característica de Alice. A continuación se muestran algunas directrices importantes que usted deberá tener en cuenta cuando trabaje con herencia en Alice:

1. Se incentiva la creación de métodos que funcionen a nivel de personaje. Algunos personajes de Alice ya tienen varios métodos definidos. Por ejemplo, el león que se muestra en la imagen tiene funcionalidades adicionales como caminar hacia adelante, rugir y cargar.

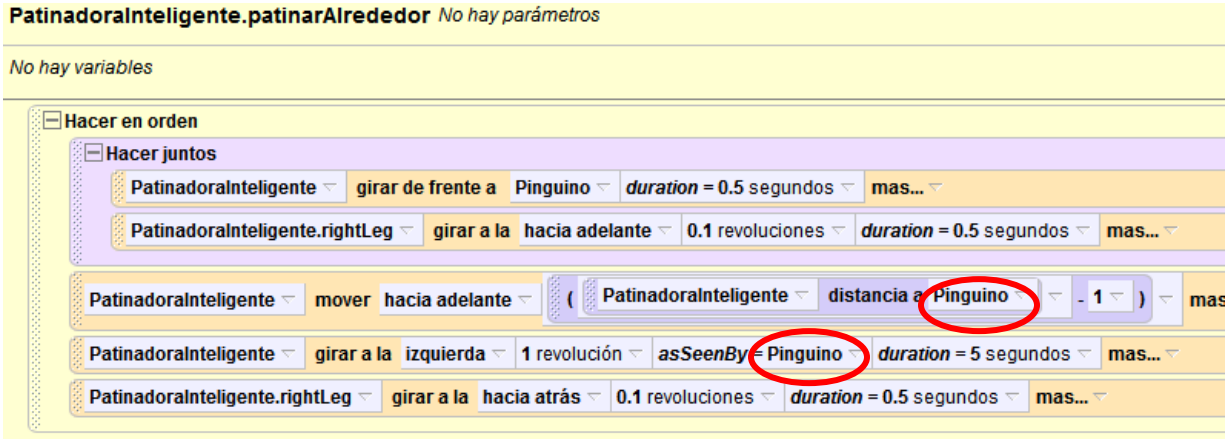
2. Si el nuevo objeto tiene varias pistas de sonido, éstas se deben importar hacia él, y no el en mundo. Un sonido que se ha importado de un personaje se guarda con el objeto al momento de ser creado. Siguiendo con el ejemplo de la imagen anterior, se ha agregado el sonido del rugido.

3. No llame a los métodos de nivel de mundo dentro de la definición de métodos nivel de objeto. En la imagen que se exhibe al final de este punto aparece el método **PatinadoraInteligente.caleidoscopio**, y que dentro de él está llamando a uno de mundo denominado **world.cambioColor**. Como se explicó anteriormente, una motivación importante para crear un método de personaje, es que se pueda usar en otros programas, y por lo tanto, en otros mundos. En este caso, si el objeto **PatinadoraInteligente** se guardara con el método **caleidoscopio**, es probable que Alice experimente errores en tiempo de ejecución, si acaso en el nuevo proyecto el método **world.cambioColor** no estuviera definido.



4. No utilice instrucciones para otros objetos dentro de un método de nivel de personaje. Los métodos de nivel de los personajes están claramente definidos para un objeto específico, de tal manera que la idea es guardar este personaje para volver a utilizarlo en otros proyectos. En este sentido, no podemos depender de otros objetos que formen parte del mundo en que se esté construyendo al nuevo objeto. Por ejemplo,

supongamos que un pingüino se añade a nuestra escena de invierno, y definimos un método denominado **patinarAlrededor**, como el que se muestra en la figura adjunta: si guardamos el objeto **PatinadoraInteligente** con el nuevo método, y lo utilizamos en otro mundo donde no haya pingüinos, Alice también va a experimentar errores, ya que faltan elementos que complementen la información con la cual se haya codificado el nuevo método.



Métodos de personaje con algún objeto como parámetro

El último de los puntos que se trató en la guía anterior, es cuando estábamos creando un método para un nuevo personaje, y que éste requiere de algún objeto para que funcione.

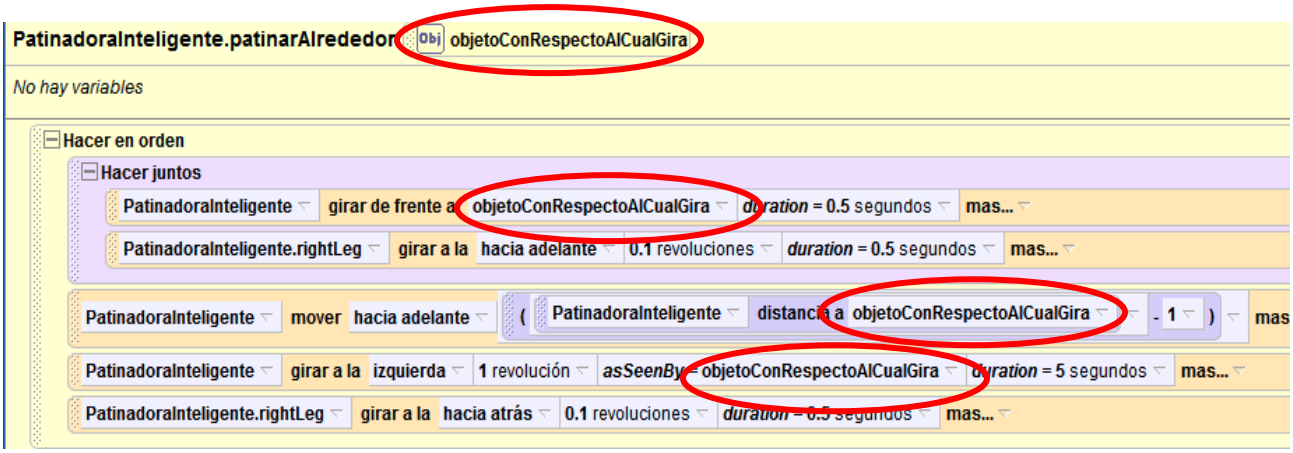
Se recomienda no usar un nombre genérico para referirse a él, porque puede ser que en el nuevo proyecto no estuviera definido, como en el caso del pingüino.

Lo que podemos hacer en este caso es usar algún objeto del mundo como parámetro. Vamos a usar el mismo ejemplo anterior, en que le pedíamos al objeto **PatinadoraInteligente** que fuera capaz de patinar alrededor de otro objeto, que en este caso corresponde a un pingüino. El método **patinarAlrededor** se puede modificar para usar un parámetro de objeto, que le vamos a llamar **objetoConRespectoAlCualGira**, tal como se muestra en la Figura 41. El hecho que esté definido de esta manera ayuda a que no tengamos que preocuparnos por algún objeto en particular.

Por otra parte, Alice no permite que el método **patinarAlrededor** sea invocado sin necesidad de pasar de un objeto como argumento del parámetro **objetoConRespectoAlCualGira**.

Esto nos va a dar seguridad de que al menos un objeto deberá estar ahí para que la niña patine a su alrededor.

Figura 41. Usando un parámetro en un objeto.



Ejercicios

1 Combinación de candado.

Diseñe un mundo que contenga una llave conformada por una combinación de seguridad, como la que se muestra en la imagen adjunta. Cree cuatro métodos que estén a nivel de objeto:

1. **unoALaIzquierda**, que permita girar un número a la izquierda.
2. **unoALaDerecha**, que permita girar un número a la derecha.
3. **revolucionALaIzquierda**, que haga que la perilla gire una revolución hacia la izquierda.
4. **revolucionALaDerecha**, que haga que la perilla gire una revolución hacia la derecha.

A continuación, cree un método denominado *abierto* que sirva para abrir la cerradura, y otro de cierre. Guarde este nuevo objeto con el nombre de **CandadoDeSeguridadModificado**.

Luego, abra un nuevo mundo y agregue el objeto que acaba de diseñar. Escriba un programa que permita abrir el candado, a partir de esta combinación:

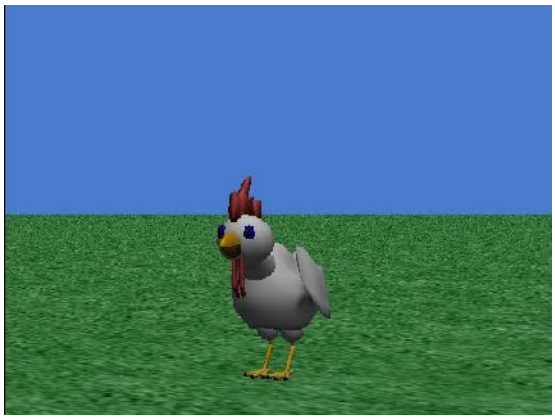
25 izquierda, 16 derecha, 3 izquierda

Cuando haya terminado las 16 derechas, el dispositivo de seguridad deberá hacer una revolución completa hacia la izquierda, y que vuelva a quedar ahí cuando termine. Utilice sonidos que permitan simular los movimientos hacia la izquierda y hacia la derecha. Emplee también una pausa de 0,5 segundo entre movimiento y movimiento, para darle un toque más realista al problema.

2 Caminata de pollo.

Agregue un pollo al mundo, como se aprecia en la imagen adjunta. Diseñe un método llamado **caminar** en el cual el pollo haga un paso a la izquierda, y después otro a la derecha. Después, cree un método para hacer que el pollo realice una coreografía de alguna canción que usted le pase como parámetro.. Guardar el pollo como un nuevo personaje llamado **PolloBailador**.

Después, haga un mundo nuevo que haga uso de este personaje y de los métodos que acaba de crear.



3 Práctica de Samurai.

Cree un mundo con un Samurái y escriba un método para que el samurái se mueva. Por ejemplo, puede inventar métodos para golpear hacia la derecha y hacia la izquierda; para patear a izquierda y derecha; y para hacer giros en estas direcciones. Guarde el samurái como un nuevo objeto llamado *SamuariEntrenado*.

A continuación, inicie un nuevo mundo y añada dos objetos de tipo *SamuariEntrenado*, y haga una animación donde estos dos personajes queden uno frente al otro y practiquen sus movimientos. Siéntase libre de elegir la secuencia de acciones que considere más apropiada.

Resumen

Los métodos de nivel de personaje se presentan como aquellas funcionalidades que han sido diseñadas para un tipo de objeto en particular, de tal manera que es posible guardar las nuevas creaciones, e invocarlas en nuevos proyectos, siguiendo los consejos que se dieron con respecto a las buenas prácticas para el manejo de herencia. Cuando hay herencia, un objeto replica las características de otro, y puede agregarle nuevas funcionalidades, como vimos en el ejemplo de la patinadora y en los ejercicios propuestos. Considere que la herencia es una forma de hacer moldes de objetos, a partir de otros primitivos, de tal manera que usted como programador pueda hacer un reutilización del código las veces que estime necesario.

Conceptos importantes de esta parte

Un personaje de nivel método está definido para un determinado tipo de objeto.

- Nuevos modelos de personajes se pueden crear mediante la definición de métodos de nivel de objeto, y luego se puede guardar la creación con un nuevo nombre.

- La herencia es un concepto en el cual los nuevos tipos de objetos se definen o derivan a partir de otro objeto, que recibe el nombre de **clase base**. En Alice, los modelos en 3D que podemos encontrar en las galerías corresponden a las clases base, y nuestros nuevos personajes son similares a las clases derivadas.
- Los métodos de nivel pueden ser escritos para aceptar parámetros de objeto, lo que permite a un método de nivel de personaje interactuar con otros objetos.

Proyectos

1 Patinadora inteligente.

Construya una patinadora inteligente que sea aún mejor de la que se presentó en la sección anterior. Cree los métodos

- **patinarAdelante**
- **patinarAlrededor**
- **patinarHaciaAtras:**
  - Las acciones deben ser similares a las del método **patinarAdelante**,, salvo que ahora tiene que hacer un deslizamiento hacia atrás.
- **saltar.** En este caso, la patinadora debe moverse hacia atrás, doblar, levantar una pierna, y a continuación, moverse hacia arriba (en el aire) y girar alrededor de sí misma dos veces antes de aterrizar en el hielo y bajar la pierna volver a su posición inicial.

Empiece con un mundo de tipo invernal. Añada su personaje, y haga que la nueva patinadora haga gala de sus nuevas cualidades artísticas, a través de llamadas a los métodos que se han descrito. Agregue un pingüino y un pato, para que la niña patine alrededor de estos objetos.

2 Creación propia.

Elija un animal o una persona de una de las galerías, con la condición que el personaje seleccionado deba tener por lo menos dos piernas, los brazos, y/o las alas que le permitan moverse, girar y rodar. Escriba tres métodos del objeto que permitan ampliar las definiciones primitivas de estas características. Por ejemplo, si elige una persona, que el mover hacia la derecha incorpore en su definición gestos con las manos, movimientos de cabeza, etc. La idea es que usted practique lo nuevo que hemos estado revisando hasta el momento. Después, abra un mundo en el cual y agregue sus nuevos personajes, y hágalos interactuar de la forma que a usted le parezca mejor. Siéntase libre de experimentar.

2.6 Consejos y Técnicas

Propiedades

Los objetos tienen propiedades que guardan información sobre sí mismo. En la parte inferior izquierda de la ventana de la Figura 42, las propiedades del objeto pez se enumeran, entre las cuales podemos encontrar valores como color, opacidad, vehículo, textura de la piel, y otros, que le dicen a Alice cómo mostrar y animar el objeto.

Figura 42. Accediendo a las propiedades de un objeto.

Ajustar propiedades en tiempo de ejecución

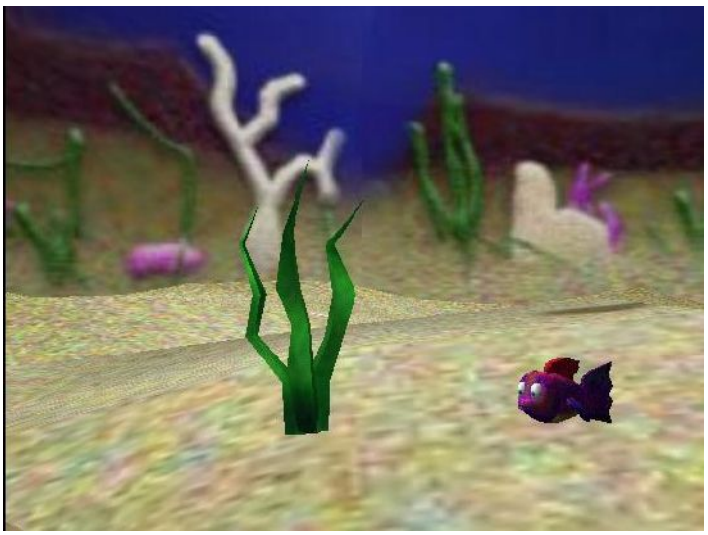
A lo largo de este apunte, se utilizaron ejemplos para configurar las propiedades de un objeto en el momento que una escena inicial se crea. Por ejemplo, se establece la propiedad color del suelo a azul para hacer que parezca agua. Esta sección apunta a cómo cambiar ciertas propiedades mientras una animación está en marcha. En Ciencias de la Computación, decir «*mientras una animación está en marcha*» equivale a decir «*en tiempo de ejecución*».

Si usted compra un nuevo teléfono celular, usted tendrá que sacarlo de la caja y luego probar todas las características interesantes del aparato: una especie de «*meterle mano*» sin que nos tomemos la molestia de leer el manual de instrucciones. De la misma manera, usted puede aprender acerca de las propiedades de los objetos mediante esta técnica, ya que Alice está pensado para que usted pueda aprender a medida que interactúa con los elementos que conforman la interfaz del programa.

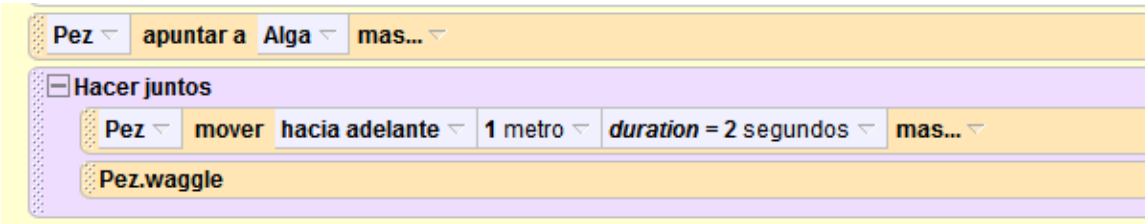
Para establecer alguno de estos valores en tiempo de ejecución, arrastre desde el cuadro de dialogo la propiedad que está en el editor, y seleccione el nuevo valor que quiera que tenga. Alice genera automáticamente la instrucción para que esa propiedad se altere en tiempo de ejecución. Veamos algunos ejemplos prácticos de esto.

Ajuste de la opacidad

Considere al pez que se muestra en esta escena, que está en nadando en dirección al alga. Cree instrucciones para que el pez esté frente a las algas y, a continuación, nade hacia ellas, como se muestra a continuación.



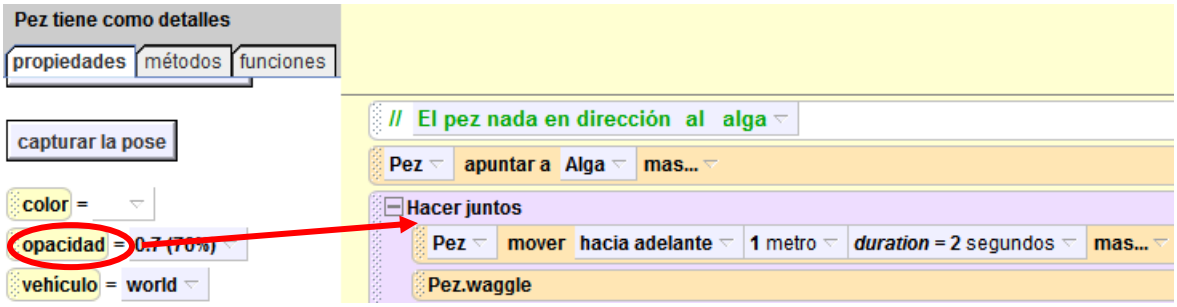
El método menear la cola se define para que los peces muevan su cola de izquierda a derecha.



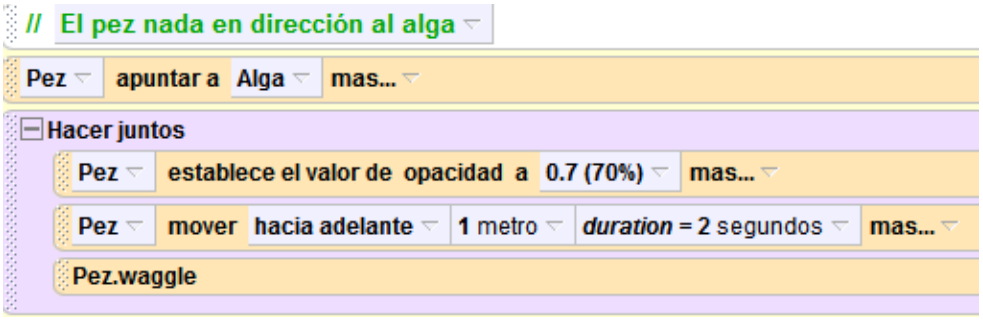
Como el pez se desplaza hacia las algas, ellas también se alejan de la cámara. Para simular las condiciones submarinas, debemos considerar que como el pez nada lejos de la cámara, éste debiera desaparecer porque el agua diluye nuestra visión de

los objetos distantes. Podemos hacer que pez se haga menos visible cambiando la propiedad opacidad. Como la opacidad se reduce, un objeto se vuelve menos diferenciado. Para cambiar la opacidad, haga clic en opacidad en el cuadro de dialogo del panel de propiedades y arrástrelo en el editor. En el menú emergente, seleccione el porcentaje de opacidad, tal como se muestra en la Figura 43:

Figura 43. Ajustando la opacidad en el editor de código.



El código que resulta es similar al siguiente:



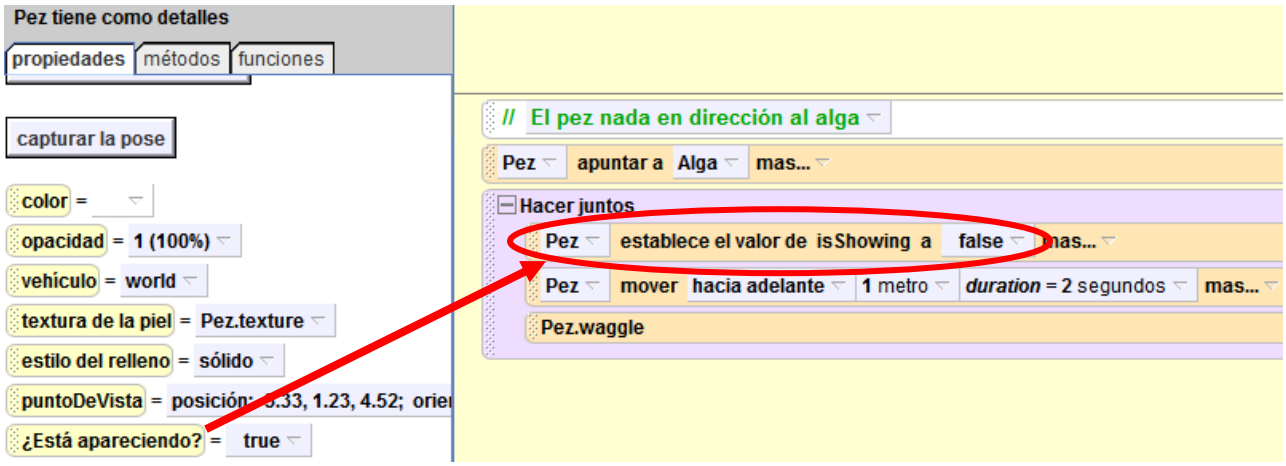
Cuando el mundo se ejecuta, el objeto pez se desvanecerá, como se muestra en la Figura 44. Bajar la opacidad hace que el objeto se vuelva menos opaco y, por lo tanto, menos visible. Cuanto el valor se ha establecido en un 0%, significa que es completamente invisible a nuestros ojos, pero no quiere decir que el objeto se haya eliminado, y sigue formando parte del mundo.

Figura 44. Efecto del ajuste de la opacidad en tiempo de ejecución.

Ajuste isShowing

En lugar de hacer desaparecer gradualmente algo en la distancia, podemos querer que algo deje de verse de manera abrupta. Alice proporciona un segundo mecanismo para hacer que un objeto sea invisible, por medio de una propiedad denominada **isShowing**. En el momento en que un objeto se agrega por primera vez a un nuevo mundo, Alice hace automáticamente el objeto visible en la escena, e *isShowing* es considerado verdadero. Cuando queremos que un objeto sólo se evapore en tiempo de ejecución, la propiedad isShowing se debe establecer en falso. Para hacer esto, arrastre el cuadro de dialogo a la propiedad isShowing en el mundo y seleccione falso en el menú emergente, como se muestra en la Figura 45:

Figura 45. Hacer que un objeto desaparezca de la vista.



Cuando un objeto establece la propiedad isShowing en falso, tampoco quiere decir que sea eliminado del mundo. En su lugar, el objeto simplemente no se muestra en la pantalla. Más adelante, podemos revertir esta situación haciendo que la propiedad isShowing sea igual a verdadera. Note además que en la imagen anterior, el valor de la propiedad **¿Está apareciendo? (o is showing)** aparece como **true** dentro de la ficha propiedades. Eso significa que el objeto tiende a mostrarse, pero esto se puede modificar directamente en el editor. Al final se ejecuta lo que esté dentro del código.

Funciones o preguntas

Alice proporciona un conjunto de preguntas integradas, en el sentido que puede consultar sobre los valores de las propiedades de los objetos y de las relaciones entre ellos. En otros lenguajes de programación, las preguntas a menudo se llaman **funciones**, y que ellas tienen que **devolver algún valor**. Entonces:

¿Qué tipo de valor se puede esperar recibir cuando se haga una pregunta en Alice?

Los valores pueden ser de varios tipos diferentes. Cinco de los más comunes son los que se muestran en la Tabla 9:

Tabla 9 . Tipos de preguntas (funciones) más comunes.

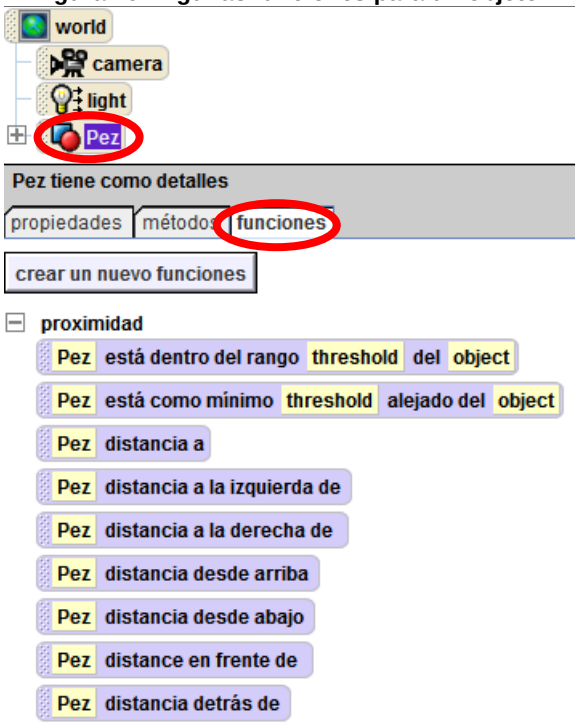
Tipo	Ejemplo
Número	5; 3.14
Valor Lógico	Verdadero o Falso
String	“Hola mundo”
Objeto	Un robot, un conejo, una silla
Posición	Rotacional o traslacional.

¿Qué preguntas se pueden formular?

Al igual que con los métodos, algunas de las preguntas son de nivel de mundo y otras de tipo de objeto. Las preguntas de nivel de personaje atañen directamente a propiedades específicas del objeto en sí mismo, como su altura, anchura y profundidad. Las preguntas de nivel de mundo son más útiles y usadas, ya que tienen relación con aspectos como el ratón, la duración de ciertas operaciones, y algunas operaciones matemáticas.

A modo de ejemplo de preguntas nivel de personaje, vamos a continuar con el ejemplo del pez utilizado anteriormente. Para ver una lista de posibles preguntas para el objeto pez, seleccione pez en el árbol de objetos y luego la ficha de preguntas en el panel de detalles, como se muestra en la Figura 46.

Figura 46. Algunas funciones para un objeto.

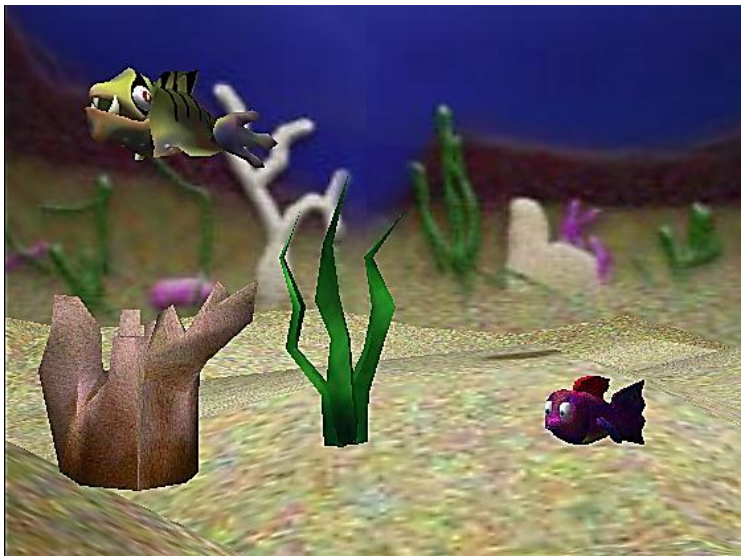


Usted podrá notar que las preguntas de personajes se dividen en categorías:

- **Proximidad**: distancia con respecto a otros objetos.
- **Tamaño**: dimensiones tales como altura, el ancho y profundidad, y cómo se pueden comparar con las dimensiones de otros objetos en el mundo.
- **Relación espacial**: orientación en comparación con otro objeto en el mundo. Por ejemplo: estar a la izquierda o estar a la derecha.
- **Punto de vista:** posición en el mundo
- **Otros**: como el nombre de una subparte del objeto.

Las preguntas que se muestran en la Figura 46 son preguntas de proximidad, como por ejemplo: ¿el pez está dentro del umbral de otro objeto?, o ¿el pez estará al menos en un cierto umbral de distancia con respecto a otro objeto? Si pensamos que el umbral corresponde a una distancia en metros, ambos tipos de preguntas devuelven un valor verdadero o falso. Otras preguntas, como ¿cuál es la distancia del pez que está a la derecha (o izquierda)?, retornará un número que expresa la distancia en metros con respecto a los otros objetos.

Preguntar sobre distancia.



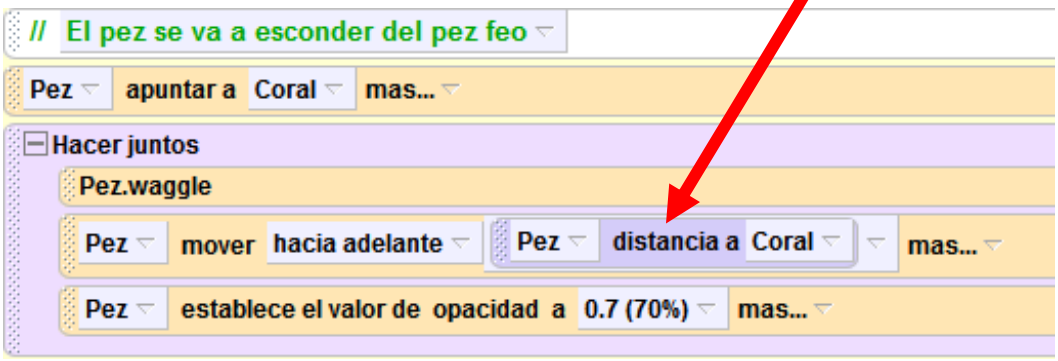
En la imagen adjunta se ha añadido otro personaje, que se llama **pezFeo**. Lo que queremos es que el pez nade hasta el arrecife de coral y se esconda del otro. El código del programa necesita ser modificado para hacer nadar el pez hasta el coral. Pero: ¿A qué distancia se encuentra de su objetivo?

Una manera de saberlo es utilizar prueba y error, por medio del intento reiterado con varias distancias hasta encontrar una que funcione, de tal manera que efectivamente veamos al pez oculto en el coral, y no que esté a punto de llegar, o que se haya pasado. Otra técnica para encontrar la distancia es preguntarle directamente a Alice: ¿cuál es la distancia del pez al coral? Alice responderá a la pregunta mediante la devolución de la distancia solicitada. Entonces, puede ser que el pez avance exactamente la distancia que haya retornado la pregunta.

En primer lugar, hay que modificar el código para indicarle al pez que se desplace hasta el coral según lo indique la distancia que se obtuvo tras la pregunta. Para esto, arrastre la distancia al cuadro de dialogo en el editor, y sustituya el valor 1 metro que tenía en el código original, por el objeto de destino, que en este caso corresponde al coral, como se muestra en la Figura 47.

Figura 47. Trabajando con la función de distancia entre dos objetos.

El código que resulta es el siguiente:

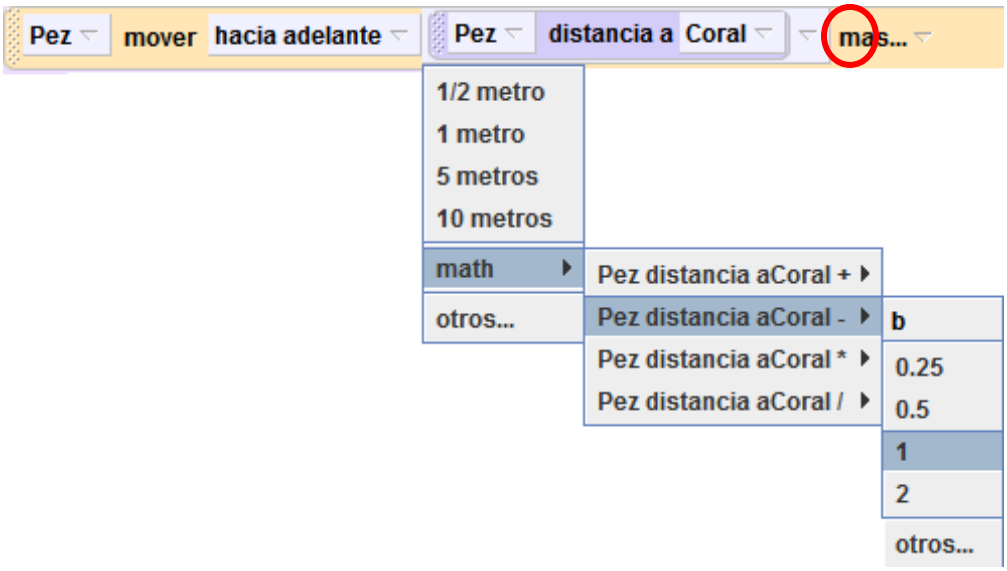


Tenga presente que una pregunta no incluye el uso de signos de interrogación. Pero no se preocupe, porque Alice lo entenderá como tal.

Expresiones

En algunas ocasiones, puede ser que alguna expresión numérica que aparece como resultado de alguna función no nos brinde el resultado esperado. Por ejemplo, en el código anterior pedíamos que el pez se moviera tantos metros como lo indicara la distancia que existiera hasta el coral. Es posible que usted no necesite llegar hasta el **centro** de ese objeto, sino que a lo mejor quisiera disminuirla ligeramente. Entonces, para ajustar la distancia entre el pez y el coral, podemos trabajar con algunas **expresiones aritméticas** que vienen incorporadas con el motor de Alice: suma (+), resta (-), multiplicación (*) y división (/). Para hacer algún cambio, haga clic a la derecha del cuadro de diálogo distancia, y a continuación seleccione **math->(Pez distancia a Coral)-**, y luego presione el número 1:

Figura 48. Ejemplo de expresión matemática para restar una cantidad dada.

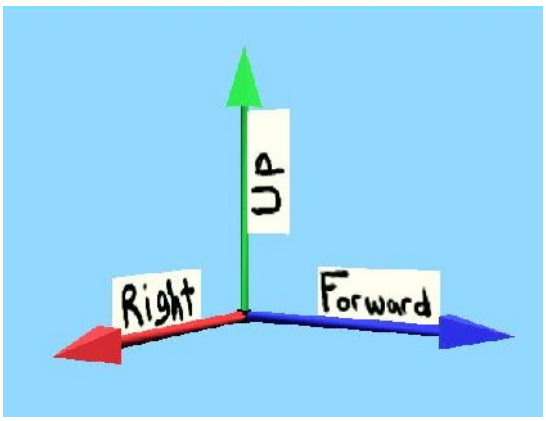


La instrucción resta un metro desde la distancia, como se muestra a continuación. Ahora, el objeto pez se detendrá antes de chocar con el coral.

Instrucción «*moverse hacia*»

La instrucción **moverse hacia** traslada un objeto a una ubicación específica en el mundo tridimensional en que estamos trabajando. Así, una ubicación se especifica mediante una referencia a la posición del objeto a lo largo de un eje 3D, como se muestra en la Figura 49.

Figura 49. Visualización de los tres ejes coordinados.

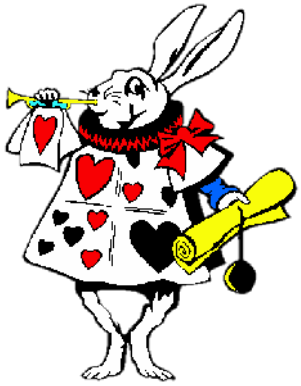


Cuando un objeto se agregó por primera vez al mundo, se coloca automáticamente en la **posición (0,0,0)**, que está en el **centro del mundo**. La **primera** coordenada especifica la posición izquierda/derecha, el **segundo** una **posición arriba/abajo**, y el **tercero una posición hacia adelante o hacia atrás relativo al centro del mundo.**

Aunque podemos decirle a Alice que para mover un objeto a una ubicación se pueden introducir las coordenadas, gran parte del tiempo, utilizamos la instrucción **moverse hacia** en base a la posición del objeto hacia el cual queramos movernos, ya que resulta más sencillo de imaginar, y además porque Alice «*conoce*» la ubicación espacial de cada uno de los objetos que formen parte del mundo.

A modo de ejemplo, en la imagen adjunta aparece un entrenador practicando basquetbol en el gimnasio. El hombre va a lanzar la pelota hacia el aro. Queremos animar el movimiento de la pelota en el borde del aro del tablero. Bastará con elegir el objeto de pelota de basquetbol, y arrastrar la instrucción **moverse hacia**, con destino a la red:

Queda como ejercicio darle mayor realismo a la animación, para que la pelota describa esta trayectoria: llegar hasta el suelo, y de ahí salir disparada hacia el sector del aro, de tal manera que termine acabando en la red, y cayendo nuevamente al piso.



Tercera Parte

Eventos

El verdadero mundo que nos rodea es interactivo. Por ejemplo: conducimos automóviles que giran a la derecha o la izquierda cuando movemos el volante, cambiamos el canal en el televisor mediante el envío de una señal de un mando a distancia, podemos presionar un botón en un controlador de juego para hacer que un personaje salte, camine o dispare. Esta introducción es dar paso a la creación de programas interactivos con Alice, donde los objetos que participan de las escenas responden a los clics del mouse y las pulsaciones de ciertas teclas. Nos hemos concentrado en escritura de programas de naturaliza no interactiva, ya que los personajes que intervienen ejecutan un cierto flujo de instrucciones de manera lineal. En esta parte de la segunda unidad, veremos cómo se pueden hacer programas interactivos.

Gran parte de la programación de computadores funciona como el programador lo tiene previsto, ya que el control del flujo de todas las acciones corre por cuenta suya. Sin embargo, muchos programas basan su funcionamiento a partir de la interacción con el usuario, ya que éste es quien determina el orden de las acciones. Por ejemplo, hacer un clic con el ratón y pulsar una tecla puede significar una señal para indicarle a Alice que haga alguna acción determinada. El clic del ratón o la tecla es un **evento**. Un *evento* es algo que sucede como respuesta a alguna acción que el usuario haya llevado a cabo.

Naturalmente, todo esto lleva un poco de planificación y organización, ya que es necesario decirle a Alice cómo realizar un tipo particular de evento, y luego indicarle lo que tenga que hacer cuando ocurra. Esto significa que tenemos que escribir métodos que describan las acciones que deben emprender en los objetos cuando algún evento se ha gatillado.

Después veremos cómo pasar parámetros a métodos controlados por eventos. En algunos lenguajes de programación, esto puede resultar ser algo complicado, pero en Alice la respuesta al manejo de eventos se ha construido de una manera que permita omitir las dificultes que esto conlleva.

2.7 Eventos

Controladores de eventos.

Control de flujo

Escribir un programa interactivo tiene una gran diferencia con respecto a uno que no lo sea, y ella radica en la forma en que la secuencia de acciones está controlada: en uno no interactivo, la secuencia de acciones está predeterminada por el programador, ya que él diseña por completo un completo guión gráfico, y a continuación escribe el código de programación para la animación de las acciones. Una vez que el programa está construido y probado, el programa ejecuta la misma secuencia de las acciones que fueron diseñadas de antemano. Pero, en un programa interactivo la secuencia de acciones se determina en tiempo de ejecución.

Eventos

Cada vez que el usuario hace clic con el ratón, o presiona alguna tecla, se desencadena un evento como respuesta del programa a ese estímulo. O bien, algunos objetos que estén en escena pueden alterar su posición inicial, y generar con eso otro evento. El punto al que queremos llegar es el siguiente: cada vez que el programa se ejecuta, es posible que diferentes interacciones con el usuario termine generando versiones diferentes de las animaciones. Por ejemplo, en un juego de video que simula una carrera de autos, el orden de las escenas se determinan en función de la habilidad que tenga el jugador para maniobrar el vehículo, y de los obstáculos que pongan los demás autos y los tipos de pistas que vayan apareciendo conforme avancen las acciones.

Manejador de eventos

La pregunta que se debe responder es:

¿Cómo afecta esto a lo que usted hace como programador?

Usted debe pensar en todos los posibles eventos y hacer planes para lo que debería suceder como respuestas a esos hechos. Los métodos que se denominan **manejadores o controladores de eventos** se escriben a continuación para llevar a cabo las respuestas. Finalmente, el evento debe estar ligado al controlador. Un **enlace** es una forma de conectar el evento con el controlador de eventos. Una cosa importante a tener en cuenta es que cada vez que se produce un evento, la ubicación de los objetos en la escena no ser la misma, ya que las acciones del usuario pueden cambiar la escena y la ubicación de los objetos en la escena entre las llamadas al controlador de eventos.

El teclado como ejemplo de control

Empezamos con un ejemplo aéreo que está a cargo de un simulador de vuelo. La escena inicial muestra el biplano en el aire y algunos objetos en el suelo. En tanto, un sistema guiado le permitir al usuario ser el piloto, para poder moverlo hacia adelante, hacia la izquierda y la derecha. Por supuesto, el biplano es una estrella en un espectáculo

aéreo, por lo que tendrá que programar la nave para que pueda hacer alguna acrobacia, como el clásico movimiento sobre su propio eje, que deberemos programar.

Entradas

La idea de un simulador de vuelo es permitir que el usuario interactúe con el avión, de tal manera que la persona envíe una señal a Alice para animar algún movimiento en particular. En este caso, tanto el teclado como el ratón pasan a ser los dispositivos emblemáticos que permiten la generación de estas entradas.

Para un simulador de vuelo, el usuario puede presionar un conjunto de teclas. Por ejemplo, las de flecha se pueden utilizar para determinar cada posible dirección de movimiento. La entrada también se puede obtener a través de clic con el ratón, el movimiento de una bola de desplazamiento, o el uso de un joystick. En este apunte, nos basaremos en el uso teclado y del ratón como los dispositivos que se usarán para animar las interacciones.

En nuestro simulador de vuelo, pulsaremos la flecha y la barra espaciadora para recibir retroalimentación por parte del usuario, de tal manera que si él presionara la tecla de flecha hacia arriba, el avión se moverá hacia delante. Si el usuario pulsa las teclas de flecha izquierda o derecha, el avión gire a la izquierda o la derecha. Para el giro acrobático sobre su propio eje, se usará la barra espaciadora. La selección de estas teclas es arbitraria.

Diseño, guiones gráficos

Ahora, estamos listos para diseñar el programa de simulador de vuelo. Cada vez que el usuario presiona una fecha de dirección o la barra espaciadora, se genera un evento. El programa de animación consiste en métodos para responder a estos eventos. Estos métodos se denominan **manejadores de eventos**. Para simplificar el análisis, vamos a trabajar con dos eventos posibles:

1. Pulsar la barra espaciadora para dar la vuelta del avión.
2. Tecla de flecha hacia arriba para mover el avión hacia adelante.

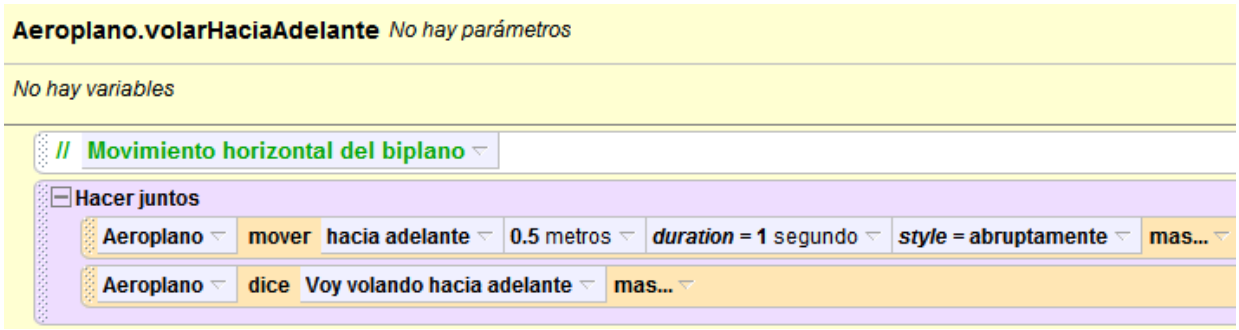
Como tenemos dos eventos, vamos a requerir de dos controladores de eventos, como se muestra en la Tabla 10. En este caso, el uso de sonido es absolutamente opcional, pero sirve para darle más realismo a las acciones.

Tabla 10. Un par de controladores de eventos.

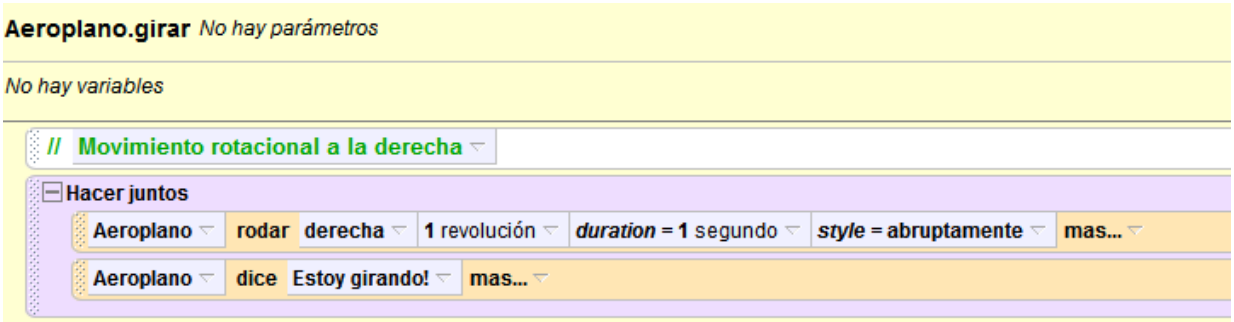
Evento: Presionar la barra espaciadora. **Respuesta:** **Hacer juntos:** 1. El avión gira una vuelta completa. 2. Reproducir el sonido del motor del avión.	**Evento:** Presionar la tecla de flecha hacia arriba. **Respuesta:** **Hacer juntos:** 1. Mover el avión hacia adelante. 2. Reproducir el sonido del avión en movimiento.

Métodos controladores de eventos

El único objeto afectado por los principales eventos de pulsación de teclas es el avión. Por lo tanto, los métodos pueden ser de nivel de objeto para el avión, y estos son: **volarHaciaAdelante**, y **girar.** El primero de ellos se encargará de responder ante el evento de pulsar la tecla de flecha hacia arriba, con el fin de simular el movimiento hacia adelante del avión, como se ilustra a continuación. En lugar de algún efecto de sonido, se agregó un texto que indicara lo que el avión está haciendo:



El método **girar** controla el evento que al mantener presionada la barra espaciadora, el avión gira sobre su propio eje, como se muestra en esta figura:

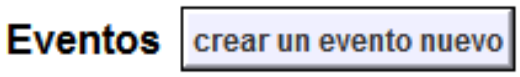


Los métodos para ir hacia la **derecha** o hacia la **izquierda** quedan como ejercicio para el lector.

Enlace los eventos con los controladores de eventos

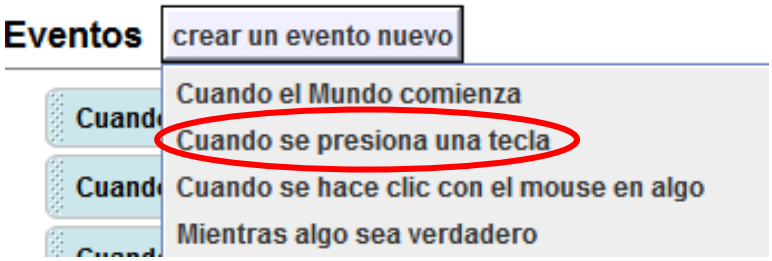
Cada método del controlador de eventos debe estar vinculado con el evento que se va a utilizar, para que finalmente el método se active como respuesta. En el editor es donde se crean los vínculos de los eventos, como se muestra en la Figura 50:

Figura 50 . Lugar donde se crean los eventos.

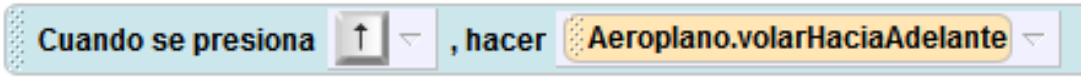


En primer lugar, hay que crear un evento. Presionamos en el botón **crear un nuevo evento**, y luego seleccionar lo que corresponda con el tipo de acción que se vaya a realizar. La Figura 51 ilustra esto:

Figura 51. Creación de un evento asociado con la pulsación de una tecla.



A continuación, cuando se ha marcado la opción **Cuando se presiona una tecla**, elegimos la de flecha hacia arriba para **enlazarla** con el método de volar hacia adelante:



El proceso se repite para vincular la barra espaciadora para el método girar sobre su propio eje:

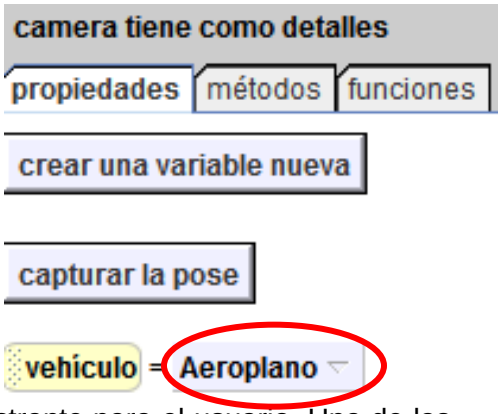
Implementación incremental

Ahora que los enlaces se completaron, el mundo que acabamos de crear debe ser probado. Para esto, hay que guardar lo que hayamos hecho, y luego apretar el botón de reproducción. Tenga en cuenta que esta animación aún no está terminada, ya que faltan los métodos que se deban ejecutar cuando se pulsen las teclas de flecha izquierda y flecha derecha, y de otras acrobacias que se podrían escribir. Sin embargo, es importante probar los métodos asociados con los controladores de eventos a medida que se vayan creando. La regla que podemos seguir es la siguiente:

Escribir un método y probar, escribir otro método y volver a probar… hasta que termine la construcción del programa.

Esta es una buena estrategia de trabajo que se conoce con el nombre de **implementación incremental**. La ventaja es que facilita de sobremanera la tarea de **depuración del programa**, ya que permite mejorar los problemas que pudieran haber apenas se haya construido algún nuevo bloque de programación.

Nota técnica

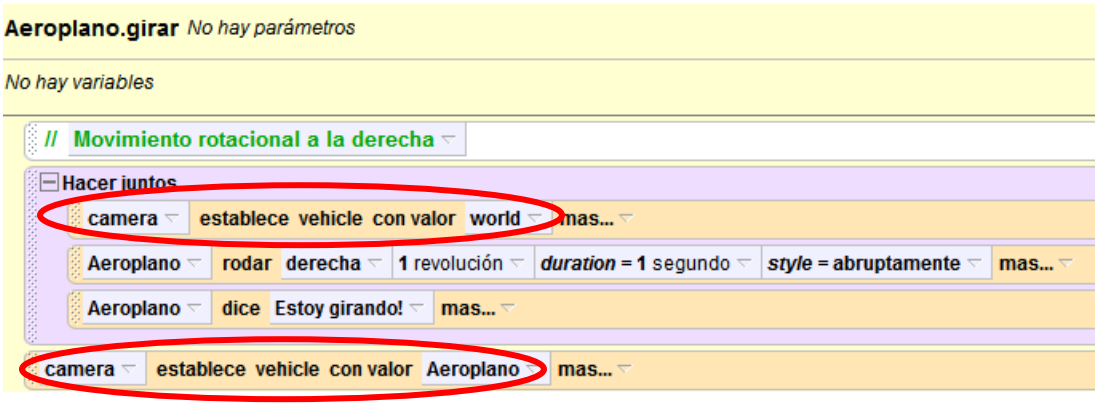


Un mundo interactivo como el del simulador de vuelo requiere que el usuario sepa con anticipación cuáles son las teclas que deba presionar para que el programa funcione correctamente. Se podría escribir un método de nivel de mundo en el cual se muestre un texto en 3D que brinde una breve explicación del sistema, de tal manera que al cabo de unos cuantos segundos, el texto desparezca, y comience la simulación propiamente tal.

El avión en el simulador de vuelo es un excelente ejemplo de un objeto que se pueda mover fuera de la vista de la cámara. Por supuesto, la desaparición del avión de la visión del mundo sería frustrante para el usuario. Una de las técnicas para mantener siempre la visión en el avión, es que éste pase a ser el **vehículo para la cámara**.

A veces, cuando un problema se soluciona, otro aparece. Este es el caso de usar el avión como el vehículo de la cámara, porque cuando se activa el método de girar sobre su propio eje, la cámara también lo va a hacer, y va provocar que todo el mundo gire. Para evitar este problema, quizás el método debería ser revisado para establecer temporalmente que el vehículo de la cámara sea el mundo, mientras el giro se desarrolla. Luego, el vehículo puede ser nuevamente el aeroplano justo antes que el método termine:

Figura 52. Ejemplo de cambio de vehículo en tiempo de ejecución.



Ejemplo del control del ratón:

En este segundo ejemplo vamos a trabajar con la ardilla Slappy, que acaba de recibir su propio vehículo motorizado, que funciona a control remoto. Esta animación mostrará el primer paseo de Slappy. Queremos que el usuario pueda controlar el avance y retroceso de la motocicleta nieve. Para esto, vamos a trabajar con un interruptor remoto, como se muestra a la izquierda. Lo que haremos es lo siguiente: cuando el usuario haga clic en el botón verde del interruptor, la moto va a seguir adelante, y Slappy se montará en la moto mientras grite algo como «*wahoo*». Cuando el usuario haga clic en el botón rojo del interruptor, la moto y Slappy se moverán hacia atrás y Slappy mirará a la cámara y dirá: «*esta tontería está dando vueltas en reversa*».

Guiones gráficos que participan en esta historia

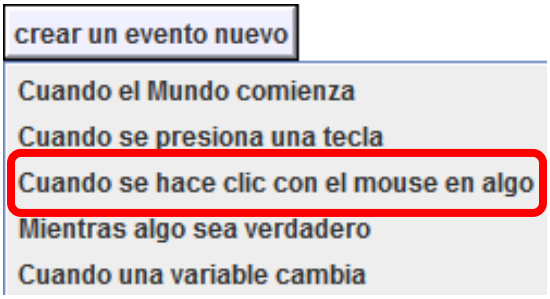
Identificamos que dos eventos pueden ocurrir: un clic en el botón rojo y un clic en el botón verde del interruptor de control. Dos controladores de eventos deben ser diseñados: uno para una animación hacia adelante, y otro para ir en reversa, tal como se muestra en estos algoritmos:

Evento: Hacer clic en el botón rojo **Repuesta:** **Hacer al mismo tiempo:** 1. Slappy y la motocicleta se mueven hacia adelante. 2. Slappy dice «*wahoo*»	**Evento:** Hacer clic en el botón verde **Respuesta:** **Hacer al mismo tiempo:** 1. Slappy y la moto se mueven hacia atrás. 2. Slappy dice «*esta tontería está dando vueltas en reversa*»

Métodos controladores de eventos

Por supuesto, dos objetos están involucrados en estas acciones. Por lo tanto, los métodos deben ser métodos de nivel de mundo. En ambos casos, Slappy debe parecer estar sentado en la motocicleta. Con esto en mente, es evidente que los movimientos de Slappy y de la moto nieve deben tener lugar al mismo tiempo. Como a menudo ocurre en el caso de programación, hay dos formas de hacer que la moto de nieve y Slappy avancen juntos. Una de ellas consiste en sincronizar los movimientos por medio de un bloque **hacer juntos**. Una segunda manera es que la moto sea el **vehículo** para Slappy. Los métodos controladores de eventos que se ilustran a continuación dan por hecho que se ha trabajado con la propiedad vehículo para sincronizar a Slappy con la moto nieve. La ventaja de utilizar el vehículo es que sólo una instrucción es necesaria para mover ambos objetos. El bloque *hacer juntos* es útil para sincronizar movimientos con sonido.

Enlace a los controladores de los eventos



Cada método de animación debe estar ligado a los del ratón, de tal manera que cada clic gatille la ejecución de un método como respuesta. El método **desplazarseHaciaAdelante** debe estar vinculado a un clic del ratón en el botón verde del interruptor, y el método **desplazarseHaciaAtras** debe activarse cuando se apreté el botón del ratón en el círculo rojo. Para hacer un enlace, deberá seleccionar la opción **Cuando se hace clic con el mouse en algo**. La Figura 53 muestra las vinculaciones que se han hecho entre los clics en ambos botones, junto a los respectivos métodos que se tienen que activar:

Figura 53. Asociando métodos con clics de ratón.

Nota Técnica

Hemos observado que Slappy y la moto debe se mueven juntos como si fueran sólo un objeto. Para garantizar que la moto y Slappy avancen juntos, no sólo hay que hacerlo con la propiedad de que la moto sea el vehículo para Slappy, sino que puede ser necesaria la inclusión de la instrucción **apuntar a**.

Ejercicios

1 Simulador de vuelo.

(a) Cree un mundo que dé cuenta de las acrobacias del aeroplano que se describió en esta parte. Implemente los métodos **volarHaciaAdelante** y **girarSobreSuEje**, y haga los links con los controladores de eventos correspondientes. Además, haga que ambos métodos se ejecuten de manera abrupta, para que la animación se desarrolle de manera más fluida. Si su computador tiene sonido, pruebe agregando alguna pista de audio para darle un toque más real.

(b) Agregue los métodos **volarHacialzquierda** y **volarHaciaDerecha**, para que se activen con los eventos de apretar la tecla de flecha izquierda y flecha derecha, respectivamente.

(c) Construya un método llamado **giroEnReversa**, que se active cuando el usuario presione la tecla ENTER. El funcionamiento es lo contrario de **girarSobreSuEje**.

2 Simulador de vuelo (versión alternativa).

Abra un mundo que contenga un aeroplano. La diferencia con el problema anterior, es usted pueda experimentar con otra clase de eventos, en el sentido que ahora tendrá que trabajar con la opción de **permitir que las flechas muevan a un objeto**, que en este caso pasará a ser el avión.

3 Manejando la ardilla.

(a) Implemente por completo el problema de la ardilla y la motocicleta que fue descrito con anterioridad en esta parte.

(b) Modifique los métodos de moverse hacia adelante y de moverse hacia atrás. En el caso del primero, Slappy tendrá que mirar hacia adelante, elevar sus patas delanteras en el aire, y que luego las baje hacia la motocicleta cuando el método esté por finalizar. Para el segundo, haga que la ardilla mire permanentemente a la cámara a medida que la motocicleta retroceda.

4 Tutor de tipeo.

Aprender a escribir con rapidez sin mirar el teclado es una habilidad que requiere de mucha práctica. En este ejercicio vamos a crear un tutor de mecanografía que alienta al usuario a escribir un conjunto específico de letras. Use algunas del texto 3D para crear alguna palabra en el mundo. Por ejemplo, usted podría crear la palabra ALICE con las letras A, L, I, C, y E, y crear un método para cada letra que la haga girar dos veces. Cuando el usuario escribe una letra en el teclado que coincida con la letra en la pantalla, la letra en la pantalla debe desaparecer. También incluya un método adicional llamado **girarPalabra** que haga mover la palabra sobre su propio eje 2 revoluciones a la izquierda, y 2 a la derecha cuando el usuario presione la barra espaciadora.

5 Movimiento rotacional.

Un tema que es bastante popular en física es el movimiento rotacional. Cree un mundo en el cual hayan tres clases de objetos: brújula, reloj de sobremesa y neumático. Se le pide que diseñe movimientos realistas para cada uno de ellos. Por ejemplo: si uno de los objetos es la brújula, las agujas debieran girar en direcciones opuestas, para luego terminar apuntando hacia el norte. La idea es que al hacer clic en alguno de ellos, se active su respectivo movimiento rotacional. Investigue sobre ello para que su trabajo sea más realista.

6 La magia del científico loco.

¿Quién dice que un científico loco no tiene habilidades mágicas? Cree un mundo que contenga a un científico que esté detrás de una mesa, la que contiene encima una licuadora, taza y una jarra. El punto es que cuando el ratón haga clic en cualquiera de los elementos que están sobre la mesa, el científico deberá levantar cualquier brazo, como si estuviera en actitud de lanzar algún hechizo, y hacer girar 2 revoluciones el objeto seleccionado. Tanto la jarra como la copa tienen ciertos tipos de líquidos, que pueden desaparecer cuando hayan sido seleccionados.

7 **Feel like a Ninja.**

Un ninja está intentado hacer una película de karate, pero necesita de un poco de práctica. Cree un mundo con un objeto ninja en un templo de karate. Escribir los siguientes métodos de movimiento que van a permitir que el ninja se ensaye todo lo que necesite para la película:

a) **patearConPiernaDerecha**, **patearConPiernaIzquierda**. Estos permiten que el ninja levante las piernas izquierda y derecha para patear, incluyendo los movimientos que deba hacer con el pie.

b) **golpearConBrazoDerecho**, **golpearConBrazoIzquierdo**. Tal como su nombre lo indica, la idea es que estos movimientos permitan simular golpes de brazo, con los movimientos intermedios que haya que considerar.

Construya los métodos y enlaces que sean necesarios para que el ninja haga sus movimientos. Pruebe generando acciones a partir de pulsaciones de teclas y de clics de ratón.

2.8 Parámetros hacia controladores de eventos

A lo largo de esta unidad, hemos hablado de la importancia que tienen los parámetros como herramienta que permiten personalizar los métodos para trabajar con diferentes objetos, valores numéricos, sonidos o cadenas de caracteres. Los parámetros son útiles para la construcción de métodos de nivel de mundo o de personajes. En esta sección veremos cómo conjugar los parámetros con los controladores de eventos para mejorar la experiencia de interactividad de los programas que construyamos. Como ha sido la tónica hasta el momento, vamos a emplear ejemplos que van a permitir ilustrar esta nueva combinación de características.

Ejemplo de parámetro numérico

Jack está planeando probarse para el equipo de hockey sobre hielo de su universidad. Como un buen deportista, Jack sabe que «*la práctica hace al maestro*», y por lo mismo ha decidido entrenar en la superficie del lago congelado, donde ha puesto una red de hockey, en la cual quiere practicar su puntería con el palo. Una escena inicial es esta:

Diseño en una historieta

Esta animación será la primera etapa para la construcción del juego de hockey interactivo. Para esto, se debe pensar en los eventos que se vayan a generar, y cuáles serán los métodos que sean necesarios de programar. Vamos a permitirle al usuario que seleccione el nivel de potencia que tendrá cada uno de los tiros de entrenamiento. El factor de potencia determinará qué tan rápido Jack lance el disco, y qué tan lejos vaya a llegar. El factor de potencia será seleccionado por un clic del ratón sobre uno de los botones que están en la parte inferior derecha de la escena. El botón amarillo indicará baja potencia, verde indicará potencia intermedia y rojo indicará alta potencia. Un pseudocódigo, expresado como guión textual se muestra a continuación:

Evento: Hacer clic en baja potencia **Respuesta:** Llamar al método **lanzar** con potencia baja.	**Evento:** Hacer clic en potencia intermedia **Respuesta:** Llamar al método **lanzar** con potencia media.	**Evento:** Haga clic en alta potencia **Respuesta:** Llamar al método **lanzar** con potencia alta.

Tres eventos y un controlador de eventos

Hasta el momento, hemos definido tres posibles eventos, por lo que en principio podemos necesitar de tres controladores, de tal manera que sea uno para cada uno. Si lo pensamos con calma, la escritura de estos tres métodos parece innecesaria, se consideramos que lo único que cambia es el tipo de potencia de tiro. Por lo tanto, una mejor alternativa de solución es escribir uno de los métodos, y enviar el nivel de potencia como un parámetro.

Vamos a construir un método de nivel de mundo para controlar el movimiento de disco iniciado por Jack, cuyo nombre será **lanzamiento**. En el editor de este nuevo método, crearemos un nuevo tipo de parámetro, que va a representar a la potencia, que será de naturaleza numérica, y al que le daremos el 1 como valor inicial. El nombre que le daremos será igual a **potenciaDeTiro**.

Nota técnica

El método que vamos a construir se va a llamar **lanzamiento**, que obedece al siguiente guión textual:

Tabla 11. Algoritmo para simular diferentes potencias de tiro.

Parámetros: potenciaDeTiro
Hacer en orden:
1. **Hacer al mismo tiempo:**
   a) Tanto Jack como el palo de hockey giran hacia la derecha. La duración de esto lo determina *potenciaDeTiro*
2. **Hacer al mismo tiempo:**
   a) Tanto Jack como el palo de hockey giran hacia la izquierda. La duración de esto lo determina *potenciaDeTiro*.
3. El disco de hockey se mueve en dirección al arco, según *potenciaDeTiro*.
4. El disco de hockey tiene que volver a su posición inicial.

Para que esta animación se pueda llevar a cabo de manera correcta, es necesario introducir el concepto de **maniquí** o de **dummy object** en Alice, que son útiles para capturar alguna posición dentro del mundo. En este caso, hemos hecho uso de uno de ellos para controlar la posición inicial del disco de hockey, de tal manera que cuando la animación haya terminado, el objeto vuelva a donde estaba en un principio. Para hacer esto:

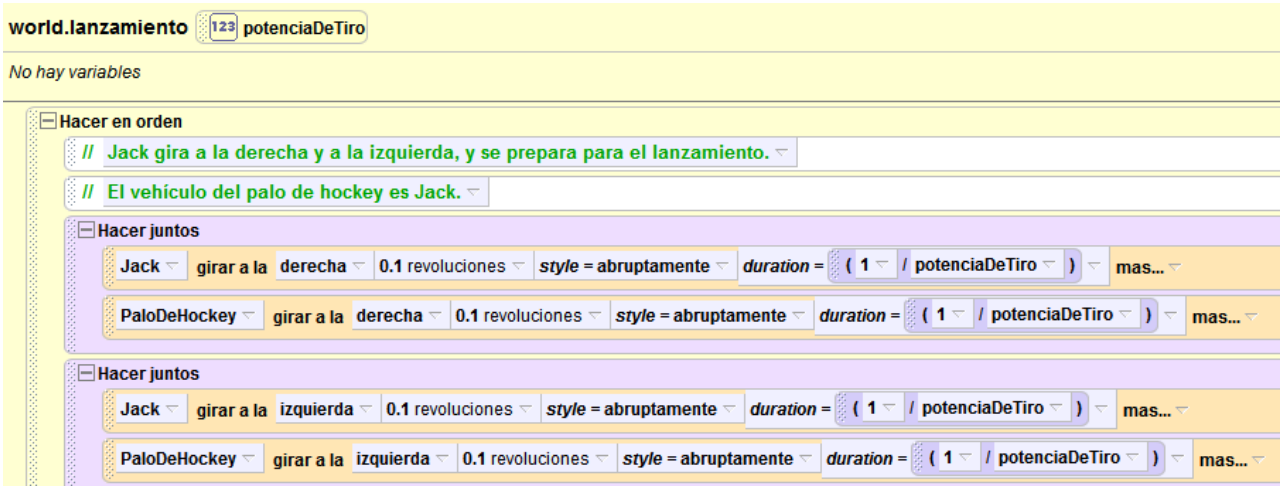
1. Vaya a la configuración de la escena inicial, y haga clic en el disco de hockey.
2. Presione el botón **más controles >>**
3. Después, presione donde dice **dejar caer (drop) el maniquí en el objeto seleccionado**.
4. Haga clic en **Hecho**.
5. Después, en el árbol de objetos encontrará un ícono de carpetas que dice **Maniquíes**.

Cuando esté por terminar el código, se invoca el comando **moverse hacia**, y selecciona la opción **dummy1** que está disponible el menú contextual.

Un parámetro se usa tanto para la duración como para la distancia

El método de lanzamiento se construye usando el parámetro de potencia de duración, que nos ayudará a determinar la velocidad de lanzamiento. El método se muestra en la Figura 54. Observe que el parámetro de duración se establece como como **1/potenciaDeTiro**. De esta manera, si el factor de potencia es un número grande, la duración será más corta, y la animación ocurrirá más rápido.

Figura 54. Método que gatilla un evento basado en un parámetro numérico.



Enlaces a los controladores de eventos

Si vamos al editor de eventos, podemos construir algo similar a lo que nos ofrece la Figura 55, en la cual se han definido de manera arbitraria los valores para el parámetro **potenciaDeTiro**, que dan cuenta que existe una proporcionalidad directa entre el número que representa y la fuerza que se le quiera imprimir al lanzamiento:

Figura 55. Asociación de eventos con métodos que tienen parámetros numéricos.

Pruebas

Cuando se utilizan los parámetros en la programación orientada a eventos, es de gran importancia ejecutar la animación varias veces para asegurarse que todo funcione como se espera. Una conocido técnica para la realización de pruebas de parámetros numéricos es intentar con valores pequeños, grandes, negativos, o cero, con el fin de detectar posibles anomalías que no siempre son fáciles de prever cuando uno construye algún programa.

Ejemplo de un parámetro de objeto

Vamos a trabajar en este escenario: en el antiguo mundo de la mitología griega, Zeus era un Dios omnipotente. Si Zeus se enfurecía, un relámpago sería lanzado desde los cielos, y atacaba a cualquiera que se interpusieron en el camino. Esta animación es una simulación de una antigua tragedia griega.

La escena inicial se construye con Zeus, que tiene acceso visual a un templo, y está montado en una nube con vistas a una escena del templo a su cargo sobre una nube; usaremos un objeto rayo, y algunos filósofos griegos: Eurípides, Platón, Sócrates, y Homero. La escena inicial del templo se ilustra en la imagen de la derecha. El rayo está oculto dentro de la nube que está inmediatamente detrás de Zeus. También, un objeto de humo se ha posicionado debajo de la tierra, inicialmente fuera de la vista. En términos de animación, el humo se ha situado bajo la tierra 5 metros, en la etapa de diseño del mundo inicial.

Guión textual para la tragedia griega

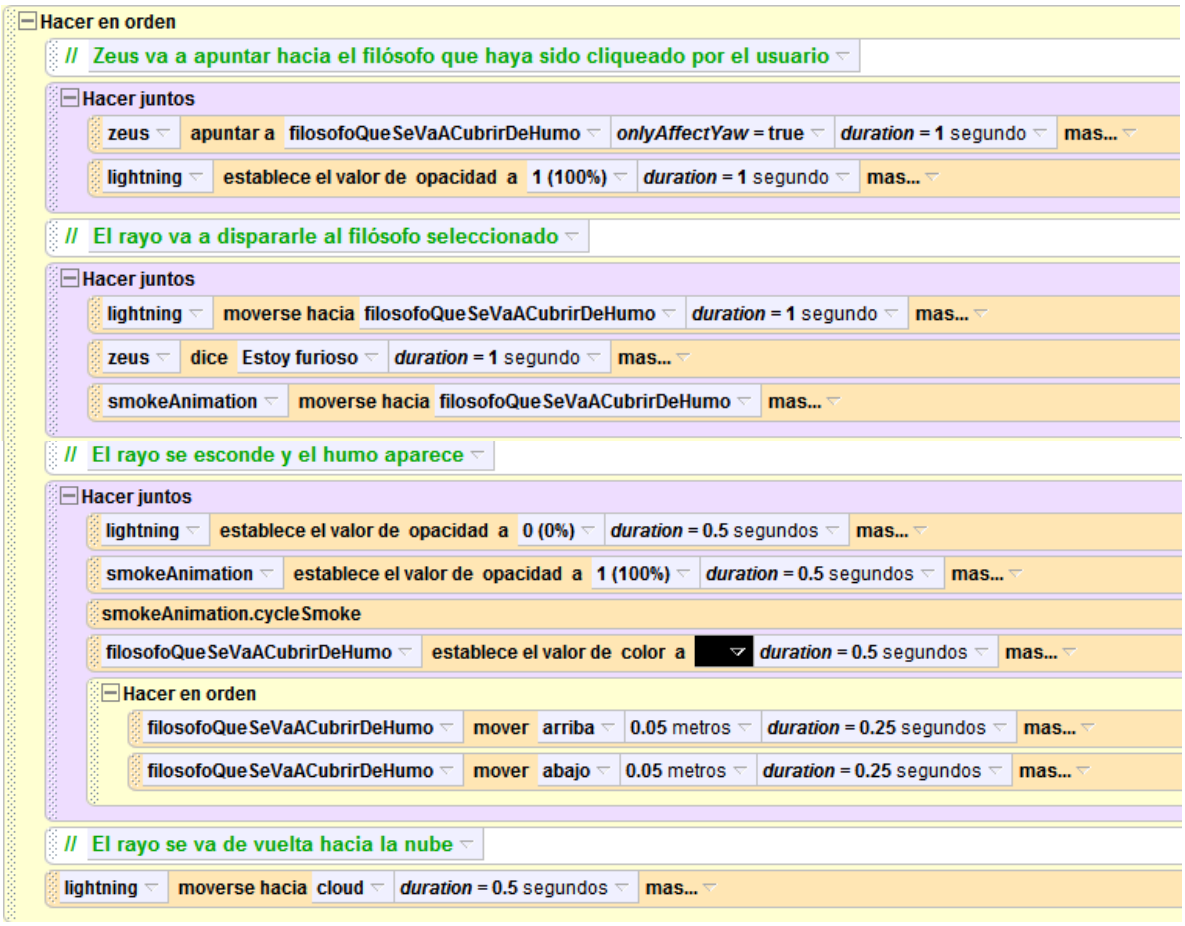
Para hacer que esta animación sea interactiva, permita que el usuario sea capaz de elegir el objeto filósofo que vaya a ser la próxima víctima de la ira de Zeus. De esta manera, cuando el usuario haga clic en uno de los objetos, Zeus se dirigirá al personaje, el relámpago le caerá por encima, y aparecerá el humo que terminará por cubrir al filósofo. El pseudocódigo de lo que se quiere hacer es esto:

```
     Evento: Se ha hecho clic con el ratón sobre un objeto de
tipo filósofo
     Respuesta del controlador de eventos:
          Hacer en orden:
               1. Zeus apunta hacia el objeto que se haya
          cliqueado.
               2. El relámpago golpea el objeto.
               3. El humo aparece desde la tierra y cubre al
          objeto seleccionado, para representar su trágico final.
```

Escribir el controlador de eventos

El primer paso consiste en definir y crear el método de mundo que llamaremos **tragediaGriega**, cuyo código puede consultar en la Figura 56. Se ha definido un parámetro que se nombró como **filosofoQueSeVaACubrirDeHumo**, que naturalmente deberá ser de tipo Objeto. Es importante remarcar que los nombres que ocupemos son absolutamente arbitrarios, y se han puesto para que se expliquen por sí mismos.

Figura 56. Algoritmo para representar la tragedia griega.

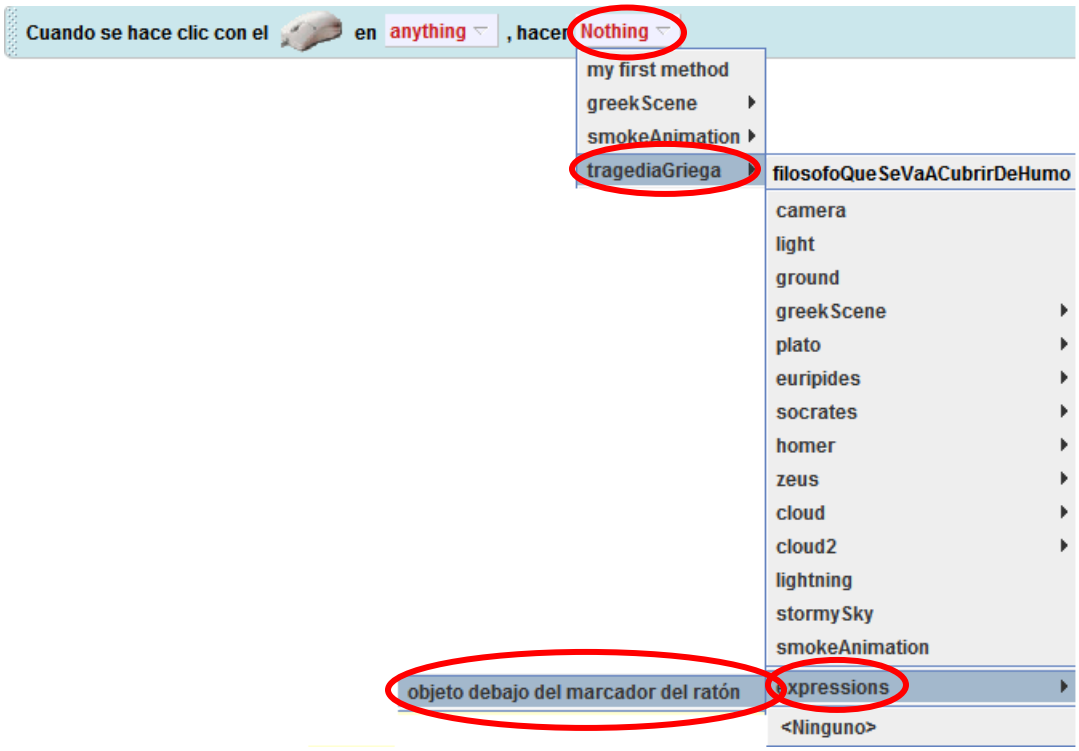


En los programas interactivos, es posible que el usuario pueda repetir una acción. En este mundo, se puede hacer clic con el botón del ratón en más de un filósofo, y por lo tanto el rayo deberá estar preparado para otra posible selección. Una vez que se ha seleccionado un filósofo, el rayo desaparece cuando el nivel de opacidad ha sido llevado al 0%, luego de lo cual volverá a retornar a la nube. De esta manera, en caso que el usuario hiciera clic en otro objeto, el rayo se volverá a hacer visible y podrá atacar de nuevo.

Enlace el evento a un controlador de eventos

Vaya al editor de eventos, y seleccione la opción **cuando se hace clic con el mouse en algo**. Después de esto, arrastre el método **tragediaGriega**, junto con el parámetro **filosofoQueSeVaACubrirDeHumo**, como se muestra a continuación:

Figura 57. Asociar un clic del ratón con la ejecución de un método que tiene un objeto como parámetro.



Las pruebas del programa

El mundo ya está completo. Por supuesto, el programa debe someterse a las pruebas de marcha y habiendo Zeus disparado los rayos. Cuando nosotros hayamos probado este programa, en cada uno de los filósofos se haya hecho clic, para asegurarse de que el relámpago correctamente da al blanco. Pero cuando se hace clic sobre las nubes, el rayo golpeó las nubes, convirtiéndolas en color negro. Y cuando se hace clic en la escena en sí, la escena se volvió negro! Este no es el comportamiento que queríamos o esperábamos. Otro problema con la animación es que el usuario puede hacer clic en un objeto que ya ha sido tocado con un rayo. Una solución a estos problemas será presentado más adelante.

Ejercicios

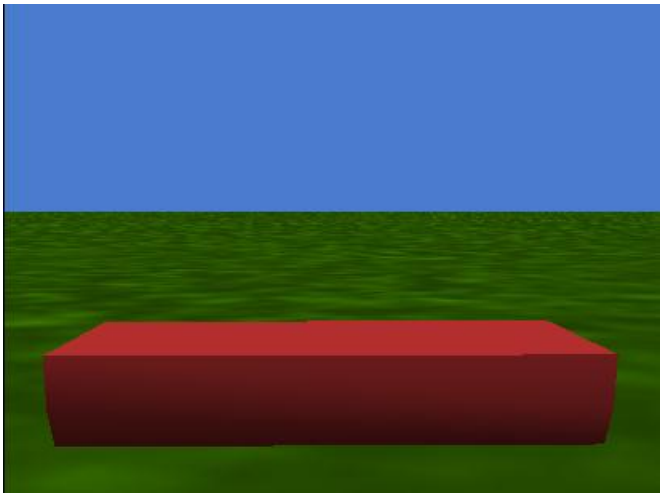
1 Carrusel.

Cree una escena de un parque de diversiones con un carrusel. En esta animación, el carrusel debe tener al menos cuatro animales: caballos, jirafas, y otros a su elección. Agregue un interruptor de dos vías a la escena inicial. Cree dos controladores de eventos: uno para que el carrusel girar hacia la derecha, y otro para que lo haga hacia la izquierda. Cuando el botón verde se presiona el carrusel debe girar hacia la derecha, y hacerlo hacia la izquierda cuando se presione el de color rojo. Construya el guión textual que tendrá que aplicar al carrusel. Agregue además un clip de sonido que simule el audio del carrusel en movimiento.

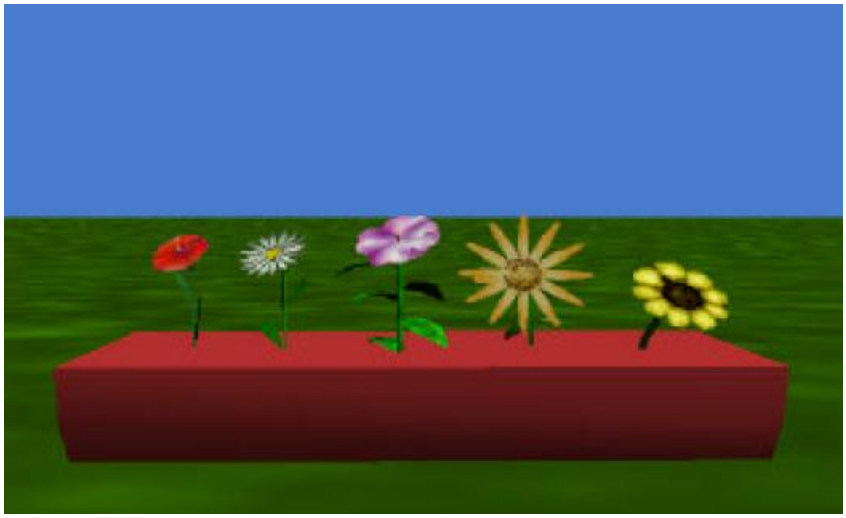
2 Festival de la nieve.

Su equipo ha creado un muñeco de nieve como la principal atracción de un festival de nieve. El objetivo de este ejercicio es que añada cuatro bolas de colores, de tal manera que al hacer clic en alguna de ellas, el mono de nieve adquiera la tonalidad seleccionada. El desafío es que debe un solo método que permita implementar el cambio de color del muñeco.

3 Florero.



Es primavera, y usted espera con ansias que las flores crezcan. Usted decide darles un poco de ayuda. Cree una escena inicial de una caja que contenga cinco flores a su elección. En la imagen de la derecha, la caja se ve vacía. Escriba un método para hacer crecer una flor en la caja, lo que se tiene que hacer en función de la tecla que el usuario presione. Por ejemplo, si el usuario pulsa «*g*», el girasol crecerá, pero si presiona la tecla «*m*», la margarita lo hará. Para el cultivo de las flores, cree un evento del tipo «*cuando se presione la tecla <tecla>*», que haga florecer aquella que haya sido predefinida por usted, dependiendo de la pulsación de teclado, que pasará a ser el parámetro de trabajo. Cuando todas las flores hayan crecido, la imagen final debiera ser la siguiente:



4 Deslizamiento de pingüinos.

Una de las actividades favoritas de los pingüinos en el zoológico local es deslizarse hacia abajo en una pendiente de hielo hacia el estanque. Cree un mundo con una escena de un lago congelado, con tres pingüinos en la pendiente, como se muestra a continuación.

Permita que el usuario haga clic en un pingüino, y que éste se deslice hacia abajo a través de la pendiente en dirección al estanque. Cada pingüino se desliza sobre su espalda mientras se desliza. Cuando llega a la punta de la pendiente, debe girar varias veces en el aire, para luego caer dentro del estanque. A continuación, deberá hundirse cinco metros. Escriba un sólo método controlador de eventos. Cuando se haga clic sobre un pingüino, haga que este objeto empiece a girar sobre sí mismo 3 veces cuando esté por lanzarse al agua. Añada un sonido de golpe de agua cuando el objeto impacte en ella.

Resumen

Esta parte ha estado dedicada a la creación de mundos interactivos, que permiten la creación de programas mucho más interesantes, como juegos o simulaciones. En Alice, la creación de eventos responde a acontecimientos bastante simples que se gatillan principalmente por pulsación de alguna tecla o por algún clic con el ratón.

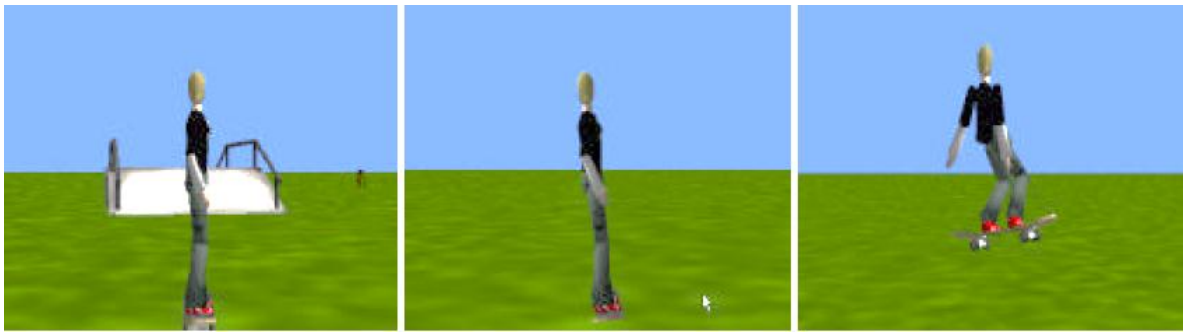
Conceptos importantes en este capítulo

- Un *evento* es algo que sucede en tiempo de ejecución, y que por lo mismo impide que el programa funcione de manera lineal, ya que es el usuario el que dirige el curso de las acciones.
- Un evento es creado por el usuario cuando pulsa alguna tecla o hace clic con el ratón.
- Un evento está vinculado a un método del controlador de eventos.
- Cada vez que se produce un evento, el controlador de eventos maneja el que le corresponda, lo que termina generando una respuesta a cambio.
- Es posible pasar un parámetro a un método asociado con el controlador de eventos.
- Los parámetros nos permiten escribir un método que puede manejar varios eventos relacionados.

Proyectos

1 Skate.

El objetivo de este mundo es simular a una joven patinadora y sus movimientos asociados. Cree un mundo que contenga una patinadora sobre un monopatín. Agregue algunos objetos donde pueda saltar por encima. Una escena de ejemplo se muestra a continuación.



La patinadora debe tener los métodos **saltar** y **girar**. El programa debe permitir que el usuario pulse las flechas de dirección hacia arriba o hacia abajo para mover la cámara y a la patinadora hacia adelante o hacia atrás, y las flechas de dirección para hacer que el patinador se apoye a la izquierda/derecha mientras la cámara se desplaza hacia la izquierda/derecha. Al final de cada movimiento, la patinadora debería volver a su posición original. La tecla «*s*» se puede usar para hacer que la patinadora salte y la tecla «*g*» para hacer su giro.

Consejos

1. Si usted está teniendo problemas para que la muchacha se mueva hacia donde usted desea, trate de usar las instrucciones «*as seen by*» con respecto a la cámara o con respecto al monopatín.
2. Una manera de hacer que la cámara siga la acción es colocarla detrás de la patinadora, y hacer que la joven sea el vehículo. Un evento del tipo «*mientras el mundo se está ejecutando*» se puede utilizar para hacer que la cámara

constantemente avance hacia adelante. Esto permitirá que la muchacha esté siempre moviéndose, y que la cámara la acompañe mientras salte o haga giros.

2 Paracaidismo.

Alice ha tomado un nuevo pasatiempo: paracaidismo. Ella está en un helicóptero, y viste un paracaídas, y debe saltar sobre la rampla, que está delante de ella. En el mundo que se muestra a continuación, se ha agregado un medio cilindro invertido conectado al helicóptero que hará las veces de plataforma de salto. También, se ha utilizado un arnés para darle mayor seguridad a Alice. La idea de esta animación es el de proporcionar un sistema que le permita al usuario guiar a Alice a cuando ella salte desde la plataforma del helicóptero con destino a la rampla. Cuando el usuario crea que Alice haya golpeado la parte superior de la rampla, deberá presionar la tecla ENTER para que Alice abandone su paracaídas. Estos son los métodos que tendrá que implementar:

1. **saltar**, que permitirá que Alice salte de la plataforma del helicóptero. Se debe activar con la barra espaciadora.
2. **deslizarHaciaArriba, deslizarHaciaAbajo, deslizarHacialzquierda, deslizarHaciaDerecha**, que permita mover a Alice en la dirección señalada. Se activan con las flechas de movimiento.
3. **encogimientoDePiernas**: permite que Alice recoja las piernas al saltar o cuando esté en el aire. Se activa con la tecla «*e*».
4. **sacarParacaidas**: permite que Alice se deshaga de su paracaídas, que en la práctica significa hacerlo desaparecer. Se activa con la tecla ENTER.

Recuerde que primero debe Alice saltar de la plataforma antes del deslizamiento y no debe bajar su paracaídas hasta que llega a la rampa.

3 Golf.

Alice ha estado tomando clases de golf y quiere demostrar sus nuevas habilidades ante sus amigos. Un truco de fantasía ha aprendido es un «*rebote*». En esta clase de tiros, la pelota de golf se golpea hacia un objeto intermedio, para luego rebotar y dirigirse a algún agujero en particular.

Cree un mundo que contenga a Alice, una esfera que va a a actuar como una pelota de golf, y cuatro objetos de círculo que serán los hoyo que hay en la tierra, junto a otros cuatro pequeños objetos — a su elección — que actúen como intermediarios para el rebote de tiro. Organice estos últimos elementos, de tal manera que cada uno esté a 10 metros de Alice, con diferentes ángulos de visualización. Las siguientes imágenes muestran una posible escena inicial.

Para animar el disparo rebote, cree un método que se llame **rebote** que tome dos parámetros **objetoIntermedio** y ***agujeroDeDestino***. Cuando se llame al método, la animación:

1. Debe simular que Alice mueve sus manos.
2. La bola debe viajar diez metros.
3. La bola tiene que rebotar en **objetoIntermedio**.
4. La bola tiene que cambiar su dirección hacia ***agujeroDeDestino***.
5. Una vez que la bola entra en el agujero, Alice tiene que levantar sus manos en señal de triunfo.

Construya los enlaces en el editor de eventos para que el usuario pueda hacer clic en el objeto intermedio, y en el agujero de destino en el cual quiera que caiga la pelota de golf.

4 Movimientos de una tortuga.

En este proyecto vamos a construir un controlador de movimientos para ayudar a una tortuga a que realice ejercicios para su próxima carrera contra una liebre. Cree un mundo que sólo contenga una tortuga, y a continuación construya los métodos de movimiento que se enuncian a continuación:

Nombre del método	Descripción	Método de activación con teclado
moverLaCabeza	Permite que la cabeza de la tortuga se mueva un poco.	Flecha hacia adelante.

Nombre del método	Descripción	Método de activación con teclado
menearLaCola	Permite que la cola de la tortuga se menee.	Flecha hacia abajo.
unPasoHaciaAdelante	Permite que la tortuga avance un paso hacia adelante, mientras ella da un paso.	ENTER.
caminarHaciaAdelante	Combina los 3 métodos anteriores, para que parezca un paso más realista. Todos los movimientos deben ejecutar en la misma cantidad de tiempo, o ocurrir de manera simultánea.	C.
darseVuelta	La tortuga hace un giro en 180º, y siempre debe caminar mientras lo haga.	V.
girarALaIzquierda, girarALaDerecha	La tortuga debe girar a la izquierda o a la derecha. En cualquier caso, debe caminar mientras gira.	Flecha izquierda, flecha derecha, respectivamente.
ocultarCabeza, mostrarCabeza	La cabeza de la tortuga se debe ocultar o mostrar.	O para ocultar, M para mostrar.
hablar	La tortuga mira a la cámara y dice «*hola*».	Barra espaciadora.

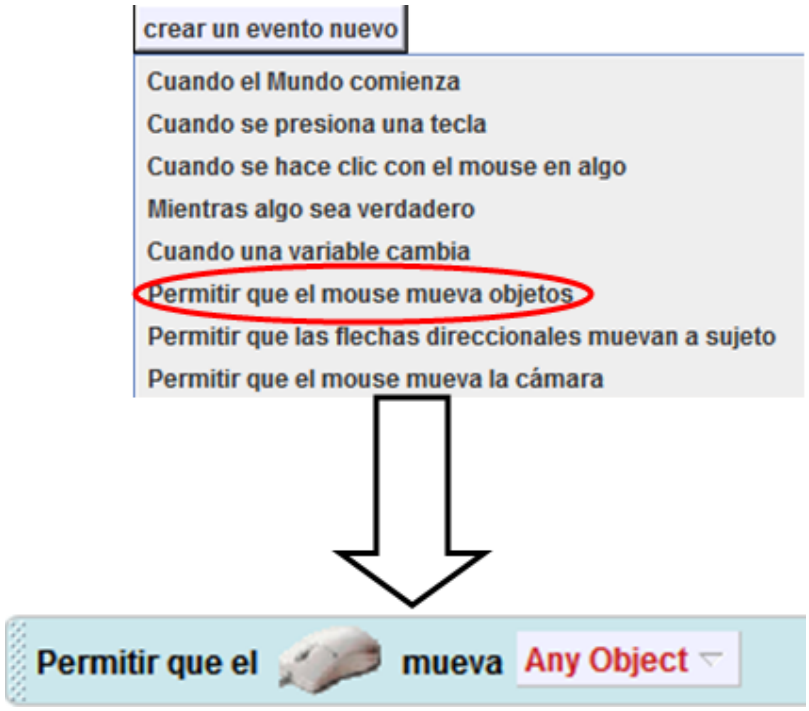
2.9 Consejos y Técnicas

Permitir que el ratón mueva los objetos

Los mundos interactivos que se han presentado hasta ahora, han utilizado eventos en los cuales el usuario hace clic con el ratón sobre algún objeto o utiliza pulsaciones de teclas para controlar el movimiento de alguno de ellos. Es posible hacer que incluso el usuario pueda arrastrar objetos del mundo con el ratón mientras un programa está en ejecución. Para ilustrar el mecanismo, consideremos un mundo como el que se muestra en la figura adjunta, en el cual habrá que reordenar el mobiliario de una habitación. Para simplificar las cosas, se han dejado afuera las paredes.

Para que el usuario pueda mover los muebles, cree un evento del tipo **permitir que el mouse mueva objetos**, como se muestra en la Figura 58:

Figura 58. Permitiendo que el ratón controle a los objetos del mundo durante la ejecución de un programa.



Inserción de imágenes

Es posible visualizar en Alice imágenes planas en 2D, que pueden tener algún formato como GIF, JPG u otros. Para agregar imagen en 2D, seleccione la opción **hacer una cartelera** del menú **Archivo**. En el cuadro de diálogo de selección, navegue hasta la imagen que quiera agregar, y finalmente presione el botón **importar**. Un posible resultado se muestra en la Figura 59:

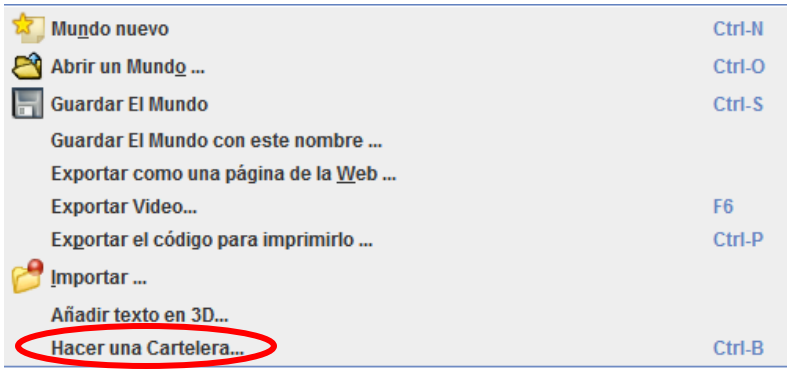


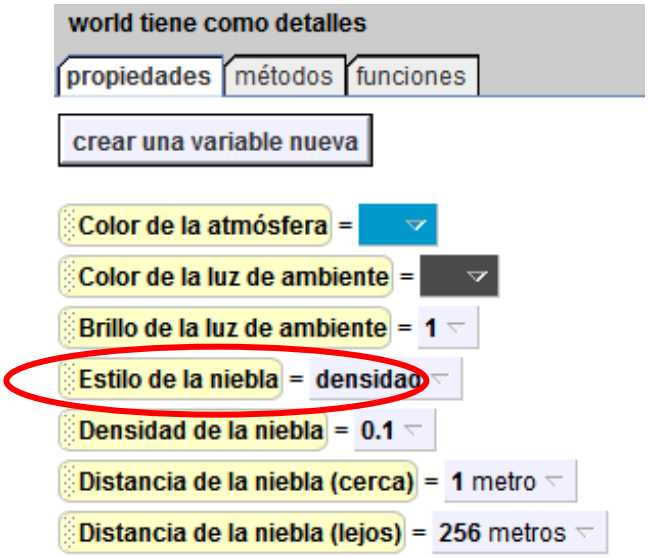
Figura 59. Resultado de la importación de una imagen.

Este tipo de imágenes puede ser útil si usted quiere incorporar elementos en **dos dimensiones** a sus animaciones. Note además que el nombre del archivo de imagen aparece en el árbol de objetos, y por lo tanto puede acceder a sus propiedades, y pedirle que ejecute algunos métodos convencionales como moverse en alguna dirección, rodar, girar, etc.

Efectos especiales: niebla

Consideremos una escena como la que se ilustra en estas líneas. En ella, un caballero armado está buscando a un dragón entremedio del bosque. Lo que queremos hacer es que el animal parezca que estuviera escondido detrás de un árbol. En este caso, sería más realista emplear un poco de niebla, para que el trabajo del caballero sea más difícil.

Para esto, haga clic en el objeto **mundo**, y luego seleccione la propiedad **Estilo de la niebla**, y ajuste el nivel de densidad que quiere que tenga. Entenderemos que la densidad se refiere al espesor de la niebla. Entre más grande sea el valor, más gruesa se pondrá.



El resultado de la aplicación de esta propiedad es este:

Orientaciones Pedagógicas

Para conseguir los aprendizajes esperados, es necesario que los estudiantes tengan claridad con respecto a los siguientes puntos:

- En Alice, las clases vienen predefinidas como modelos tridimensionales para ciertos tipos particulares de objetos, que se pueden emplear en diferentes tipos de proyectos de animación.
- El proceso de crear y mostrar un objeto en la pantalla recibe el nombre de instanciación de una clase.
  - En este sentido, Alice trabaja tanto con métodos de mundo como de objetos.
- Los parámetros permiten comunicar los valores que un método requiera para funcionar.
- Una instrucción corresponde a una sentencia que permite que un objeto ejecute una determinada acción.
- Las funciones permiten hacer preguntas sobre alguna condición, o computar un determinado valor.
- Una expresión consiste de alguna operación matemática donde intervengan valores numéricos, o de otra naturaleza.
- Nosotros interactuamos con objetos en nuestro propio mundo.
- Existen diferencias entre hacer programas interactivos con respecto a otros no interactivos.
- Un evento gatilla una determinada acción, y que las acciones ocurren en respuesta al evento.
- Los eventos se pueden considerar como planes que se pueden ejecutar en caso que algo que pudiera ocurrir dentro de la ejecución del programa.
- Es recomendable probar los métodos a medida que se van creando.
- Los métodos funcionan a nivel de mundo.

Para que estos entendimientos se puedan llevar a cabo de manera exitosa, se recomienda que los estudiantes desarrollen en las sesiones de práctica no sólo los

problemas propuestos al final de la unidad, sino también que reflexionen en torno a estas preguntas, que se pueden distribuir en el tiempo que el profesor estime conveniente:

- ¿Cómo se determinan las clases que vayan a ser necesarias para el proyecto que el estudiante vaya a abordar?
- ¿Qué diferencia hay entre una clase y un objeto?
- ¿Cómo se le pasan valores a un método?
- ¿Cómo el paradigma de la orientación a objetos permite la creación de clases que dependan de otras pre-existentes? En este caso, se alude al concepto de herencia.
- ¿Cuándo se puede emplear una función para ayudar en el proceso de toma de decisiones durante las etapas de diseño e implementación del programa?
- ¿Cómo se crea una aplicación interactiva en Alice?
- ¿Por qué el teclado y el ratón son los dispositivos que se usan para *decirle* a Alice que haga una determinada acción?
- ¿Por qué la técnica de desarrollo incremental es un concepto clave dentro del contexto de la programación interactiva?
- ¿Qué es una variable de tipo mutable, y cómo puede ayudar a hacer un seguimiento en los cambios que se hagan en una animación?
- ¿Cuáles podrían ser algunos puntos a favor, en cuanto a la creación de nuevas clases mediante herencia?

Unidad 3

Estructuras de Programación

En esta unidad aprenderemos a controlar el flujo de ejecución de las acciones que se definen en el algoritmo o guión de un programa, por medio de la incorporación de sentencias condicionales y de bucles finitos e infinitos.

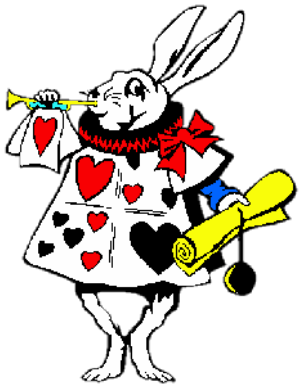
Sentido educativo de la unidad

Aprendizajes esperados.

- Reconocer y aplicar instrucciones de tipo «*if/else*» dentro de un programa.
- Identificar cuándo es necesario trabajar con bloques de programación que permitan la repetición de un conjunto de instrucciones.
- Identificar cuándo es necesario hacer preguntas a un objeto, para establecer condiciones de términos para instrucciones «*if/else*» o en bucles acotados y no acotados.

Contenidos.

- Elementos básicos de lógica booleana.
- Sentencias condicionales.
- Bucles finitos.
- Bucles infinitos.
- Introducción al uso de funciones.



Primera Parte

Estructuras de Control

En esta primera parte de la tercera unidad partiremos revisando el concepto de *ejecución condicional* de algún segmento de código de un programa. Una *condición* corresponde a la respuesta que podamos hacer sobre una situación puntual del mundo, de tal manera que nuestro programa comprueba el **estado actual** en el mundo, y elabora una *decisión* que le servirá para ejecutar o no alguna porción de código. La ejecución condicional es un concepto clave en la programación de computadores que nos permite llevar a cabo una parte de un código fuente, siempre y cuando alguna situación dada sea verdadera o no.

En el mundo real, tomamos decisiones todo el tiempo; en el mundo virtual, emplearemos decisiones para determinar el curso de las acciones que tomen nuestras animaciones. A lo largo de esta parte veremos cómo poder hacer esto dependiendo del **resultado que arroje alguna pregunta para ciertos objetos que formen parte del mundo**. En términos de programación, **las preguntas corresponden a las funciones que habíamos mencionado en la unidad 2,** y que ahora vamos a abordar de manera más profunda.

3.1 Decisiones y preguntas lógicas

Esta sección comienza con una explicación del proceso de toma de decisiones en un programa. Las decisiones son útiles cuando necesitamos que algún método o algunas instrucciones se ejecuten bajo determinadas condiciones.

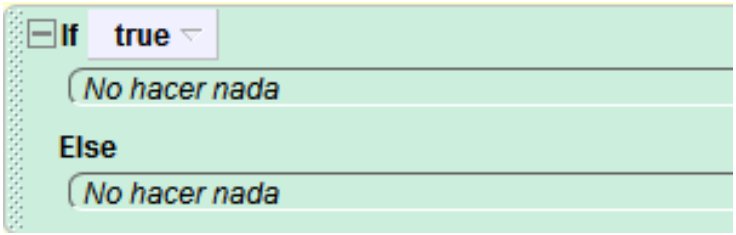
Decisiones

A veces la vida misma se basa en tomar una decisión tras otra. Por ejemplo, sólo vamos a cortar el pasto si es que la hierba no está húmeda. O también podemos decir que vamos a lavar un plato siempre y cuando detectemos que tiene cierto nivel de suciedad.

La programación de computadores está llena de decisiones, que se modelan por medio de instrucciones **if/else** — o **si/sino** —, que deben ejecutar las instrucciones que estén en su interior, dependiendo de la **pregunta lógica** que se haya enunciado. **Siempre se trabaja con este tipo de preguntas**, que se caracterizan por devolver

dos tipos de resultados: **verdadero** o **falso**, y que es común a todos los lenguajes de programación**.** En Alice, una instrucción **if/else** tiene el siguiente aspecto:

Figura 60. Aspecto inicial que tiene la instrucción if/else.



El color verde de este bloque es una pista visual que los diseñadores de Alice han implementado, como una forma de guiar al programador mientras trabaja. La instrucción if/else tiene dos partes:

1. Si la respuesta a la pregunta es **verdadero**, se debe **ejecutar lo que venga a continuación del if.** Todo lo que esté contenido debajo del **else** se omite.
2. Si la respuesta a la pregunta da como resultado falso, entonces se invierten los papeles: **el if no se ejecuta, y el else sí.**

Ejemplo de uso de «*if-else*» («*si-sino*»)

En la unidad anterior habíamos revisado el ejemplo de la tragedia griega, en la cual Zeus lanzaba un rayo que terminaba carbonizando a alguno de los filósofos con los cuales se haya enfurecido. En esa ocasión estábamos trabajando con parámetros que podíamos señalar con el ratón para practicar el concepto de eventos.

Sin embargo, ocurre también que si el usuario hace clic en otros objetos que no fuera alguno de los griegos, también terminaba de color negro. Acá se nos presenta el problema de controlar que el elemento seleccionado corresponda efectivamente a un filósofo y no en cualquier otro personaje. Por lo tanto, va a ser necesario que introduzcamos ciertos cambios en el método **tragediaGriega** que reflejen esta nueva situación.

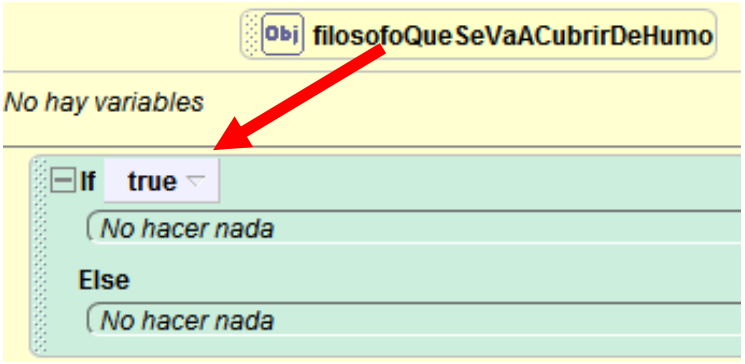
En este método, cualquier objeto que era cliqueado por el ratón era pasado como el argumento asociado al parámetro **filosofoQueSeVaACubrirDeHumo**. Lo que haremos a continuación es ocupar un bloque de instrucciones **if/else** que nos permita discriminar el tipo de objeto que se vaya a seleccionar. Para estos efectos, vamos a arrastrar al interior del método **tragediaGriega** el bloque **if/else**, dejando que la opción **true sea la primera que veamos**, porque resulta más natural pensar en positivo que en negativo. Luego, el bloque **hacer en orden** debe quedar dentro del grupo if/else, de tal manera que antes que el rayo se dispare, Alice consulte sobre cuál objeto vaya a recaer la acción. En la siguiente página veremos cómo implementar la pregunta

¿filosofoQueSeVaACubrirDeHumo == Homero?

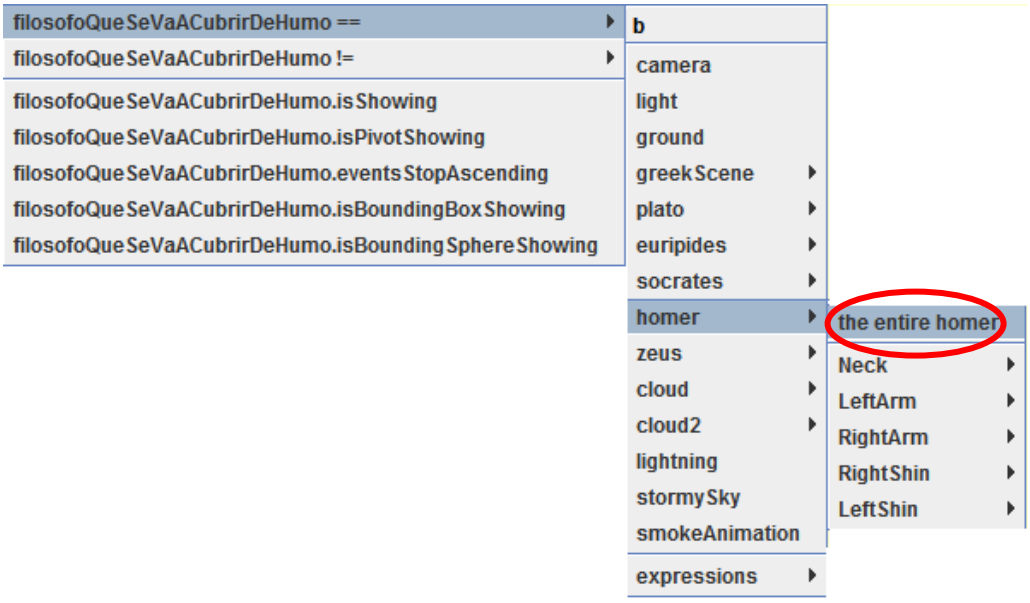
En Alice, el símbolo **==** se debe interpretar como **es igual que**. En este caso, estamos haciendo una **pregunta** que nos debe responder con verdadero o con falso si acaso el valor del parámetro **filosofoQueSeVaACubrirDeHumo** es o no es Homero.

Para crear la pregunta, lo que haremos será arrastrar desde la esquina superior del método **tragediaGriega** el parámetro **filosofoQueSeVaACubrirDeHumo** hacia el área que dice **true** dentro del **if**, de tal manera que el método **tragediaGriega** vaya cambiando gradualmente para incorporar nuevas características. Vamos a seguir estos pasos para implementar la pregunta **¿filosofoQueSeVaACubrirDeHumo == Homero?**:

1. Una vez que hemos colocado el bloque **if/else** en el código, debemos arrastrar el parámetro hacia el espacio que dice **true**:



2. Después se abre un **cuadro con opciones**, en el cual debemos elegir aquella que tiene **= =** en su definición:



¿Qué estamos haciendo con esto? Le estamos diciendo a Alice que **evalúe si acaso es verdadero el hecho que el objeto con que el usuario haga clic sea igual con el filósofo Homero**. En caso afirmativo, que haga todo lo que se indique dentro del bloque **if**, o de lo contrario, que ejecute el **else.**

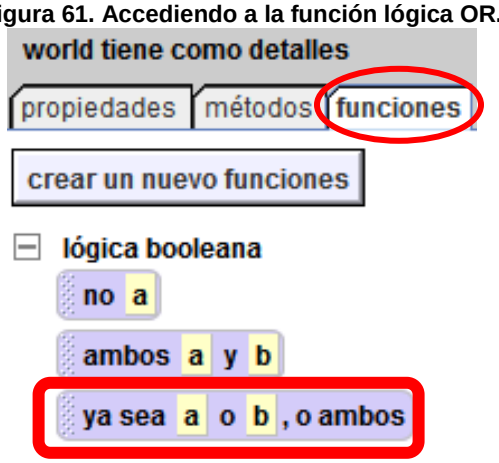
Operadores lógicos

Como usted habrá notado, la pregunta **filosofoQueSeVaACubrirDeHumo == Homero** sólo comprueba la condición que el objetivo del rayo sea igual a Homero. Pero, ¿qué pasa con los otros filósofos? Pensemos que Platón, Sócrates o Eurípedes también tienen que ser víctimas de la furia de Zeus Este es un ejemplo en donde más de una condición se tiene que cumplir. Necesitamos reacondicionar la pregunta para permitir la inclusión de los otros personajes.

Una manera de hacer que esto ocurra es por medio del uso del operador **de disyunción or**, que está disponible dentro de **las preguntas lógicas que están presentes en el mundo**, como se muestra en la Figura 6-1-4. Significa exactamente lo que suena: **una cosa, O la otra, O bien ambas**. Vale decir:

SI el filósofo es igual a Platón O igual a Eurípides O igual a Sócrates O igual a Homero, ENTONCES debe ser atacado por el rayo de Zeus.

Figura 61. Accediendo a la función lógica OR.

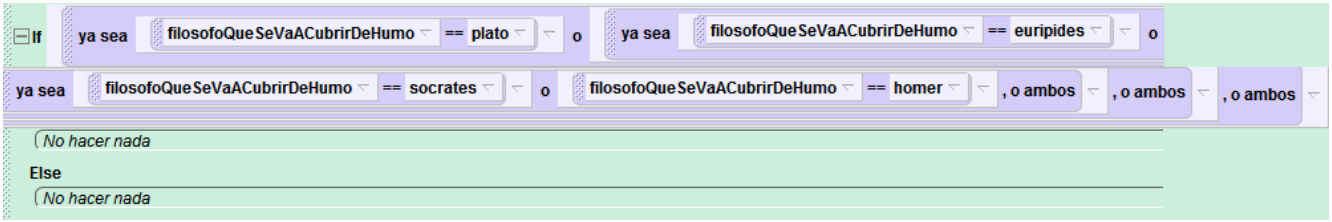


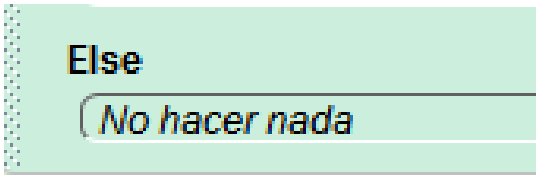
Para utilizar el operador **or**:

1. Se debe arrastrar la pestaña **ya sea a o b, o ambos** hacia la condición **true** que está dentro del if. Cuando el cuadro de diálogo pregunte por un valor, asígnele **true**, ya que por lo general resulta más intuitivo trabajar con condiciones verdaderas que con aquellas falsas.
2. Al igual que en el ejemplo anterior cuando sólo teníamos a Homero dentro de la condición, vamos a arrastrar el parámetro **filosofoQueSeVaACubrirDeHumo** hacia el primero de los **true** que aparecen dentro del **or**. Después, elegimos la opción que éste sea igual a Eurípides (sólo por hacer un orden alfabético, nada más).
3. Lo que uno pensaría de manera natural es repetir la misma operación anterior, con el valor Homero. De hecho, si tuviéramos sólo dos filósofos esto sería correcto. Si usted procede de esta manera, notará que le va a faltar espacio para agregar a los que faltan. Por lo tanto, en lugar del segundo true, vuelva a arrastrar el operador **ya sea a o b, o ambos,** y en el true de más a la izquierda haga lo que tenía pensado con Homero.
4. Notará que será necesario volver a colocar **ya sea a o b, o ambos** para que todos los filósofos tengan un lugar asegurado en la sentencia.

La primera línea tendría un aspecto similar al mostrado en la imagen adjunta. Puede que sea diferente, dependiendo si usted eligió poner a los filósofos en orden alfabético. Es importante destacar que en Alice usted verá las tres primeras líneas en una sola: acá se ha hecho un recorte de la imagen para que quede más legible.

Figura 62. Una expresión lógica que utiliza dos veces el operador OR.



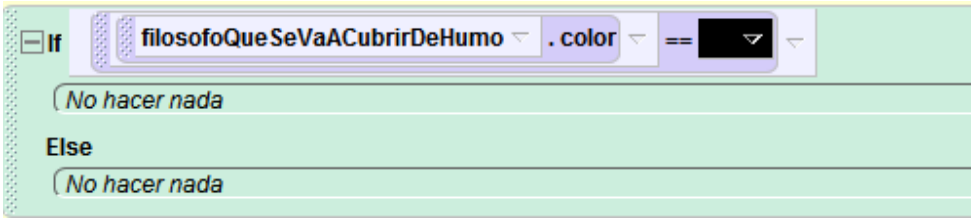


Lo interesante es que al momento de ejecutar el programa, si hace clic en alguno de los filósofos, el rayo caerá sobre el que haya elegido, mientras que si presiona el botón del ratón en otra zona, no habrá acción. Esto último se debe a que la condición del **if** no fue verdadera, y por lo tanto, pasamos a ejecutar lo que esté dentro del **else**, que por el momento hemos dejado vacío. El operador or es sólo uno de los tres **operadores lógicos** disponibles en Alice.

Otro que está disponible es el de **negación lógica**, que recibe el nombre de **no**, y lo que hace es comportarse al revés de lo que dice. Por ejemplo, si decimos

no color == negro

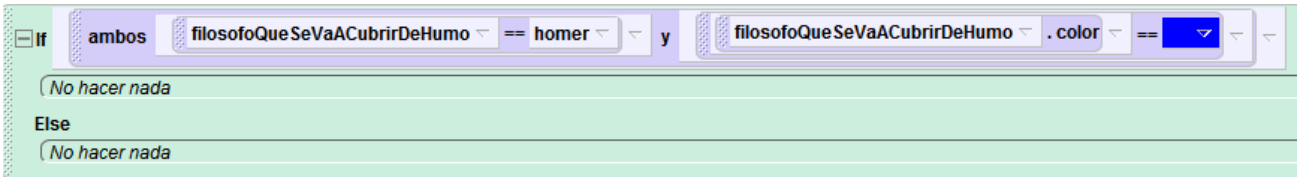
Estamos preguntando si acaso el color de pelo de algún objeto no es igual a negro. O dicho de otra forma, estamos consultando si el color de pelo es distinto de negro. De esta manera, es verdadero cuando lo que «*está adentro*» es falso, y el falso cuando lo que esté adentro sea verdadero. En este caso, si el color de pelo fuera igual a café, entonces la sentencia completa es verdadera, ya que café es distinto de negro. En el ejemplo anterior, podríamos preguntar si acaso **no se cumple que el color de un filósofo sea negro**. Para escribir lo que se ve a continuación, es necesario tomar un filósofo cualquiera, y arrastrar la propiedad **color** hacia la condición **true** que está dentro del **if**. Una vez que haya hecho eso, basta con arrastrar el nombre del parámetro, y «*dejarlo caer*» encima del filósofo que haya escogido.



El tercer operador es la **conjunción lógica**, que se denota por la letra **y**. La particularidad que tiene es que todas las expresiones booleanas que esté evaluando tengan que ser verdaderas, para la conjunción lo sea. En caso que alguna de ellas fuera falsa, la conjunción también será falsa.

Por lo tanto, es de suma importancia el cuidado que hay que tener cuando tengamos varios operadores lógicos dentro de alguna sentencia **if** o de algún **else**. La siguiente expresión se evalúa como verdadera — o **true** — si el parámetro es igual a Homero, **y** además el color corporal del parámetro fuera azul **o** negro.

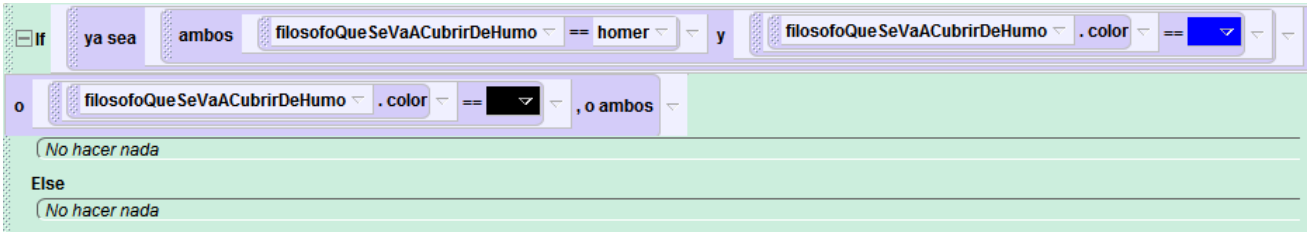
Figura 63. Ejemplo de uso de una conjunción lógica dentro de una expresión condicional.



La expresión que se muestra a continuación es verdadera si se cumple una de dos condiciones — o ambas:

1. Si el parámetro es Homero **y** el color corporal del parámetro es azul.
2. Si acaso el color corporal fuera negro.

Figura 64. Ejemplo de expresión lógica que combina conjunción y disyunción.



Estos ejemplos son un ejemplo de lo complicado que puede llegar a ser las expresiones lógicas, a medida que las vamos incluyendo unas dentro de otras. En general, se recomiendo incluir la menor cantidad posible de operadores lógicos. En caso que fuera necesario incluir varios, se sugiere **anidar** varios if, como se mostrará en la próxima sección.

Figura 65. Implementación del método de la tragedia griega.

```
world.tragediaGriega ( [Obj] filosofoQueSeVaACubrirDeHumo)
No hay variables
    If ( ya sea ( filosofoQueSeVaACubrirDeHumo == plato ) o ( ya sea ( filosofoQueSeVaACubrirDeHumo == euripides ) o ( ya sea ( filosofoQueSeVaACubrirDeHumo == socrates ) o ( filosofoQueSeVaACubrirDeHumo == homer ) , o ambos ) , o ambos ) , o ambos )
        Hacer en orden
            // Zeus va a apuntar hacia el filósofo que haya sido cliqueado por el usuario
            Hacer juntos
                zeus apuntar a filosofoQueSeVaACubrirDeHumo onlyAffectYaw = true duration = 1 segundo mas...
                lightning establece el valor de opacidad a 1 (100%) duration = 1 segundo
```

```
mas...
    // El rayo va a dispararle al filósofo seleccionado
    Hacer juntos
        lightning moverse hacia filosofoQueSeVaACubrirDeHumo duration = 1 segundo mas...
        zeus dice Estoy furioso duration = 1 segundo mas...
        smokeAnimation moverse hacia filosofoQueSeVaACubrirDeHumo mas...
    // El rayo se esconde y el humo aparece
    Hacer juntos
        lightning establece el valor de opacidad a 0 (0%) duration = 0.5 segundos mas...
        smokeAnimation establece el valor de opacidad a 1 (100%) duration = 0.5 segundos mas...
        smokeAnimation.cycleSmoke
        filosofoQueSeVaACubrirDeHumo establece el valor de color a (0, 0, 0) duration = 0.5 segundos mas...
        Hacer en orden
            filosofoQueSeVaACubrirDeHumo mover arriba 0.05 metros duration = 0.25 segundos mas...
            filosofoQueSeVaACubrirDeHumo mover abajo 0.05 metros duration = 0.25 segundos mas...
    // El rayo se va de vuelta hacia la nube
    lightning moverse hacia cloud duration = 0.5 segundos mas...
Else
    zeus dice ( Idiota..... es a Homero, no a unido con ( filosofoQueSeVaACubrirDeHumo como secuencia de caracteres (string) ) ) duration = 2 segundos mas...
```

Anidamiento de sentencias condicionales

Uno de los problemas que siguen quedando en la escena de Zeus, es que el filósofo seleccionado sigue quedando con vida, a pesar de haber sido impactado por rayo furioso del dios griego. ¿Cómo podemos evitar que esto suceda? Una solución es usar otra instrucción **if**, que le permita a Zeus disparar un relámpago sobre un filósofo que no haya sido previamente impactado. Debido a que cada uno de ellos queda convertido en color negro, podemos usar esta propiedad para determinar si el objeto ha sido alcanzado por un rayo. El proceso de creación para probar el color del parámetro se realiza en tres pasos. En primer lugar, la sentencia **if** se agrega al código, dentro de la cual escogemos la opción que el color del parámetro **filosofoQueSeVaACubrirDeHumo** sea igual a negro. Para esto, arrastramos la propiedad color de cualquiera de los filósofos hacia el **true**, y luego reemplazamos el nombre del objeto por el del parámetro, como se hizo en los ejemplos anteriores.

De esta manera, si el parámetro tiene color negro, entonces Zeus dirá que no puede eliminar a algo que ya fue atacado por el rayo. En caso contrario — vale decir, que el color del objeto no sea negro — ejecutar todo el código que teníamos de antes. Cuando uno habla de lo que se hace en caso contrario, se hace referencia a la sentencia **else**. Por lo tanto, en términos prácticos, lo que tenemos que hacer es arrastrar todo el bloque anterior **hacia el contenedor que está dentro del else.**

Operadores relacionales

En el mundo de Zeus, hemos ocupado operadores lógicos, pero sin embargo, existen otro conjunto que también desempeña un rol importante dentro de la programación de estructuras condicionales, que son los **operadores relacionales**, los que — como su nombre lo indica — permiten relacionar a dos elementos. Por ejemplo, uno puede comparar edades, determinar si dos magnitudes son iguales o diferentes entre sí, etc. En el caso particular del mundo de Zeus, hemos aplicado el operador relacional

==

De hecho, las sentencias condicionales pueden hacer una mezcla de operadores lógicos con relacionales para obtener preguntas o consultas cada vez más complejas, y que permiten implementar todos los detalles lógicos que el programador haya considerado.

A menudo, tenemos que comparar números, y ejecutar un determinado código si acaso se cumple o no una cierta relación entre ellos. Por ejemplo, si la edad de una persona es mayor o igual a 7, entonces estará en condiciones de entrar a una sala de cine acompañado de sus padres. Alice ofrece seis tipos de operadores relacionales que están agrupados en la categoría **matemática** de las funciones que ofrece el mundo, y que se detallan en la Tabla 12:

Tabla 12. Operadores relacionales que ofrece Alice.

Operador	Descripción
a == b	El número **a** es **igual** a **b.**
a != b	El número **a** es **distinto** de **b**.
a > b	El número **a** es **mayor** que **b.**
a >= b	El número **a** es **mayor o igual** que **b.**
a < b	El número **a** es **menor** que **b.**
a <= b	El número **a** es **menor o igual** que **b.**

Para crear una expresión lógica para el caso de la comparación de la edad del niño, hagamos lo siguiente:

1. Dentro de un bloque **if/else**, arrastre la viñeta **a>=b** que está dentro del grupo de funciones del mundo.
2. El valor de **a** reemplácelo por un parámetro **edadDePersona**.
3. El valor de **b** cámbielo por el número 7.

Por lo tanto, el resultado final tendría un aspecto como el siguiente:

```
If  edadDePersona >= 7
    // Si lo anterior se cumple, entonces la persona puede ingresar al cine acompañada de sus padres
Else
    // Si la edad NO fuera mayor o igual que 7, entonces NO puede entrar al cine
```

Ejercicios

1 Modificaciones al mundo de Zeus.

(a) Modifique el mundo de Zeus para hacer que cada filósofo diga algo diferente cuando se haga clic sobre él, según lo especificado en esta tabla:

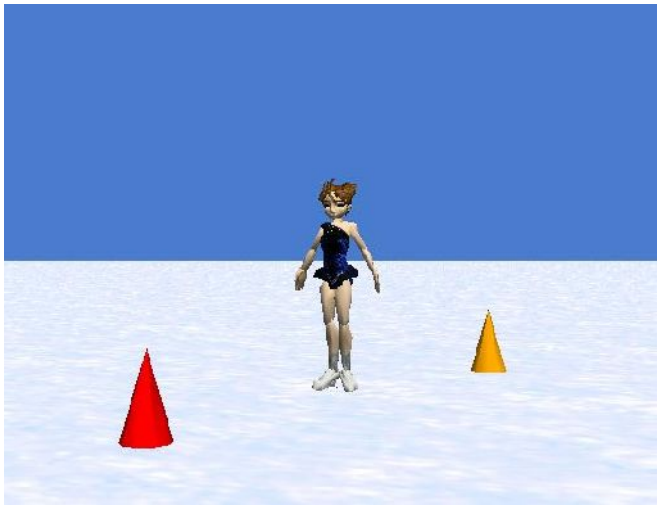
Si se hace clic en …	Entonces, debe decir
Eurípides	Puedo escribir los versos más tristes esta noche.
Platón	Pienso, luego existo.
Homero	Preparo las mejores micheladas de Grecia.
Sócrates	Los días de nuestra vida son como las arenas de un reloj.

Emplee una alguna sentencia **if/else** para determinar qué filósofo fue cliqueado, e indique el mensaje que deba decir.

(b) Modifique el mundo de Zeus, de tal manera que si se hace clic sobre Homero, él sólo avance hacia abajo, y retorne a la escena al cabo de 2 segundos, sin que se vea lastimado por el rayo de Zeus.

2 Patinaje alrededor de conos.

Cree un mundo que contenga a la **PatinadoraInteligente** que habíamos creado en la unidad anterior, junto a los conos que aparecen a continuación:



En este caso, ella se encuentra practicando algunos giros en el hielo. Para esto, la patinadora va a apuntar hacia un cono, para luego avanzar en dirección a él, ocupando los movimientos que son propios de una patinadora inteligente. La idea es que cuando se encuentre cerca de uno, haga un medio giro alrededor de él, de tal manera su cabeza quede apuntando al otro cono. A continuación, tiene que seguir patinando hacia el otro objeto, de tal manera que cuando esté por aproximarse a su nuevo destino, haga un giro completo alrededor de él. La manera en cómo la patinadora «*sabe*» que está cerca de llegar a uno de los conos, puede ocupar la función de proximidad

`PatinadoraInteligente` está dentro del rango `threshold` del `object`

O bien, otra alternativa puede ir por el lado de usar la función de **distancia**, y usar algún operador relacional como **a<b** para preguntar si acaso la distancia que exista entre la patinadora y el cono será menor que dos metros. Escriba un programa que permita a la protagonista hacer un recorrido completo por los dos conos.

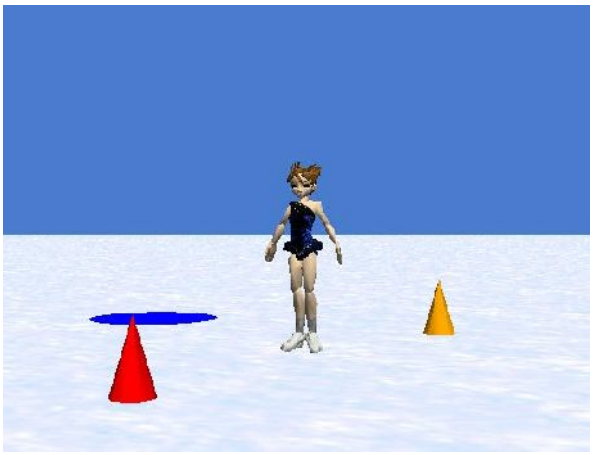
4 El ocho.

Modifique lo que hizo en el problema anterior para permitir que la patinadora dibuje un 8 alrededor de los conos.

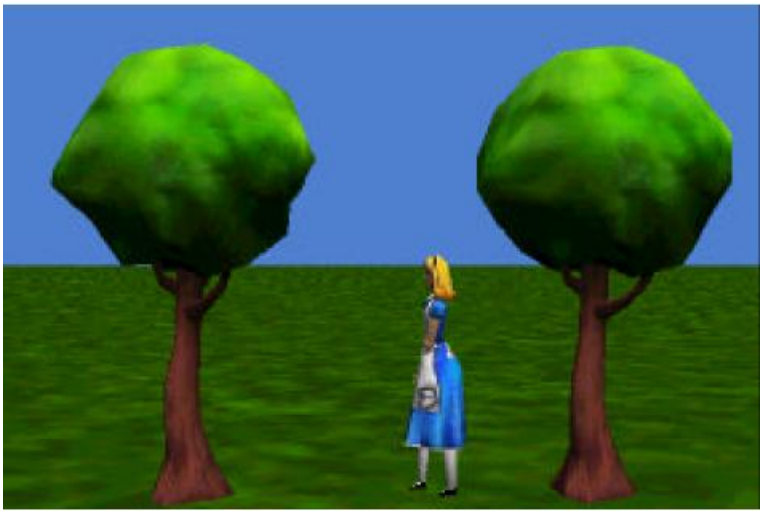
5 Peligro en el hielo.

Para este ejercicio, puede ocupar alguno de los mundos 3 ó 4. La diferencia es que ahora usted tendrá que agregar un agujero en el hielo, que se va a implementar por medio de un círculo de color azul, como lo que se muestra a continuación.

Haga que el mundo sea interactivo, en el sentido que le permita al usuario utilizar el ratón para mover el agujero alrededor de la superficie de hielo. Entonces, a medida que la patinadora se desplace a lo largo de su trayectoria, usted puede mover el lago azul e interponerlo en la ruta de ella. Modifique el método movimiento de la patinadora, de tal manera que permita comprobar si ella está andando por encima del lago. De ser así, entonces ella deberá caer, y exclamar «*glu, glu, glu*».



6 Movimiento a través de los árboles.



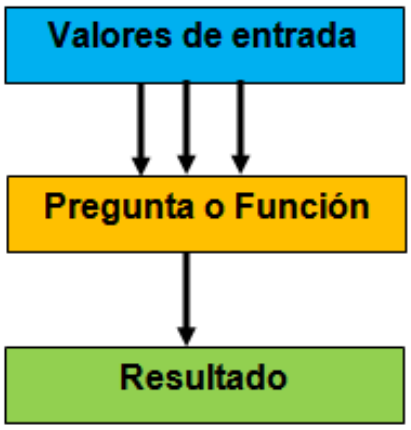
Cree un mundo en el cual aparezcan dos árboles, y un objeto que esté entremedio de ellos, como el caso de Alice que se ilustra en la imagen de la izquierda. La idea es hacer que Alice camine entremedio de los árboles. La idea es que sea un mundo interactivo, en el cual ella avance un paso cada vez que el usuario presione la tecla ENTER. Alice debe caminar hasta que llega a un árbol, y luego dar la vuelta para dirigirse al otro. Evite las colisiones.

3.2 Funciones

Introducción a las preguntas

Alice utiliza el término **pregunta** para referirse a lo que en otros lenguajes de programación se conoce bajo el nombre de **función**. En términos simples, una función es algo que recibe ciertos valores de entrada — aunque es posible que a veces eso no sea así —, y que a cambio genera alguna salida. La Figura 66 ilustra este concepto.

Figura 66. Esquema general de una función.



Para que no resulte tan abstracto, piense en una función como si fuera una máquina Wurlitzer, en la cual el usuario inserta una moneda, selecciona la canción, que luego se reproducirá en el aparato. En este caso, la entrada viene dada por la moneda y el nombre de la canción, mientras que la salida corresponde a la pista de audio que se reproduce a través de los altoparlantes.

Abstracción

Uno de los beneficios más importantes que conlleva trabajar con una función, es que nos permite pensar en el proceso general que acarrea llegar al resultado, en lugar de los micro detalles que permiten conseguirlo. Por ejemplo, cuando vamos a pagar al supermercado y vemos la caja registradora, tendemos a pensar en el resultado del precio total de la compra, que para nosotros es lo más importante, en lugar de todas las operaciones intermedias que haya tenido que hacer para llegar al cómputo final: buscar en la base de datos el precio artículo cuyo código de barras coincide con el valor que entrega la pistola que los lee, guardarlo en un acumulador, buscar los precios de los otros objetos que formen parte de la compra, agregarlos a la suma total, para luego mostrar el saldo final de la boleta.

De la misma manera, podemos pedirle a una función que lleve a cabo pequeñas acciones **que se traducen en preguntas**. Algo que debemos tener en claro al momento de trabajar con funciones es responder a esta pregunta:

¿Qué tipo de información necesitamos obtener de una función?

La respuesta a esa pregunta la vamos a develar a continuación. Mientras tanto, pensemos que una función — al igual que un método — es una característica de la programación que permite agrupar pequeños pasos en uno mayor. La diferencia entre una función y un método, es que la primera retorna algún tipo de valor, y el otro no. En Alice, una función puede retornar valores como:

- Número.
- Verdadero o Falso.
- Objeto.
- String o cadena de símbolos.

- Color.
- Mapa de texturas.
- Sonido.
- Pose.
- Posición de un objeto.
- Orientación.
- Punto de vista.
- Marco de referencia.
- Luz, dirección, texto 3D, etc.

Funciones definidas por el usuario

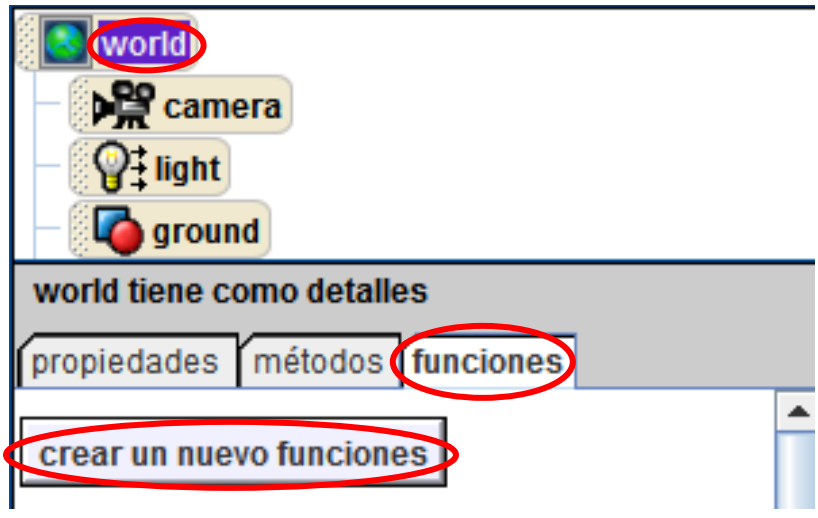
Ya hemos usado algunas de las funciones de Alice. Por ejemplo, en el mundo de Zeus, hemos preguntado si acaso el valor del parámetro corresponde con el nombre de alguno de los filósofos que participan en la tragedia griega. Lo mismo hemos aplicado para averiguar previamente si el color del personaje era negro, con el objetivo de evitar que un objeto fuera atacado más de una vez por el rayo.

Sin embargo, en algunas ocasiones puede ser necesario emplear alguna pregunta que no exista en Alice. Esta herramienta nos provee la funcionalidad de poder crear nuestras propias funciones personalizadas. Veamos cómo hacerlo.

Creación de una función

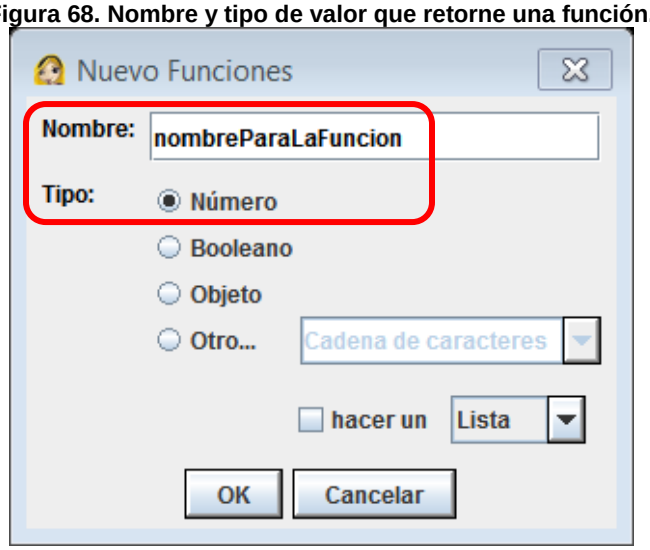
Al igual que los métodos, las funciones también pueden estar definidas a nivel de mundo, o a nivel de personaje (o de objeto). Para crear una función definida por el usuario, seleccione el mundo o el objeto con el que quiera trabajar. En la ficha de preguntas, haga clic en el botón **crear nueva función**, como se ve en la Figura 67:

Figura 67. Creación de una nueva función de mundo en Alice.



A continuación, aparecerá un cuadro de diálogo que contiene un formulario, en el cual tiene que indicar un nombre para la función, y el tipo de valor que vaya a retornar.

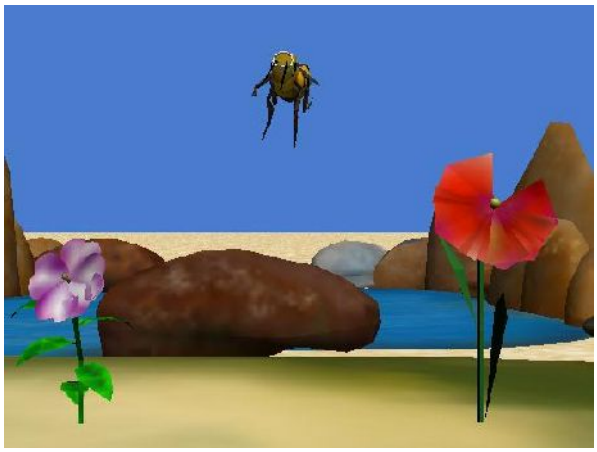
Figura 68. Nombre y tipo de valor que retorne una función.



Ejemplo de función booleana

En esta parte de la unidad, vamos a hacer un ejemplo de función booleana, con el objeto que usted se familiarice con este importante concepto. Recordemos que cuando algo es de naturaleza booleana, es porque está asociado con **valores de verdad**: **verdadero** o **falso**, que se utilizan con frecuencia en los bloques condicionales **if/else**, y en los bucles **while**, que vamos a examinar al final.

En este ejemplo, vamos a considerar una escena como la que se muestra a continuación, en la cual tenemos a una abeja obrera que está buscando una nueva fuente de polen para alimentar a la colmena. En nuestra animación, queremos escribir un método que le indique a la abeja cuál es la flor más cercana para que vuele. Para simplificar las cosas, vamos a trabajar con dos flores: un roja y otra rosada.



Sin embargo, en Alice no existe alguna función que permita determinar si acaso un objeto está más cerca de otro. Ahora bien, el hecho que no esté, no significa que no se

pueda construir, ya que las herramientas predefinidas que nos provee Alice permiten hacerlo. Esta pregunta la vamos a construir en base a tres parámetros de entrada:[21]

1. **objetoComparador**. En este caso corresponde al objeto sobre el cual vamos a tomar como referencia para analizar las distancia a los otros que sirven de referencia. Para el caso puntual de este problema, vamos a considerar que el **objetoComparador** es igual a la abeja.
2. **primerObjetoDeReferencia.** Representa el primer objeto sobre el cual mediremos la distancia que haya con respecto al **objetoComparador**.
3. **segundoObjetoDeReferencia**. Representa al primer objeto sobre el cual mediremos la distancia que haya con respecto al **objetoComparador**. En la Tabla 13 se muestra el libreto de trabajo para esta pregunta:

Tabla 13. Libreto de trabajo para la pregunta de comparación de distancias.

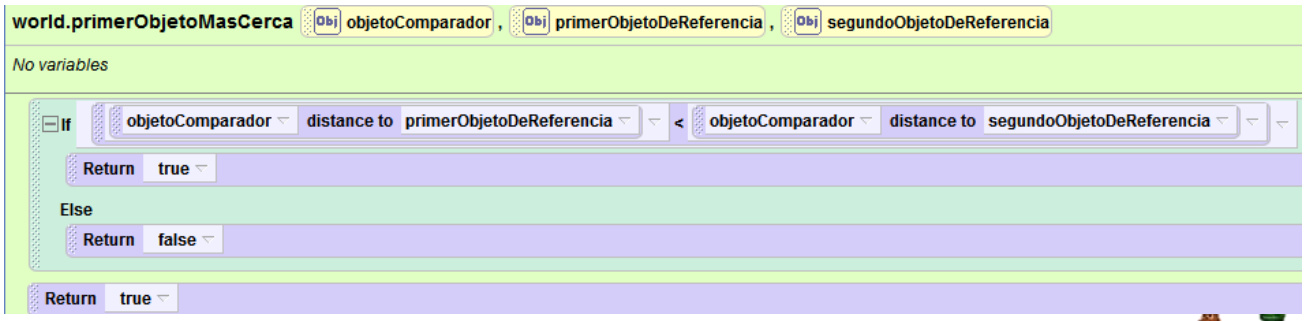
```
Parámetros: objetoComparador, primerObjetoDeReferencia,
segundoObjetoDeReferencia
      Si la distancia que hay desde objetoComparador hasta
      primerObjetoDeReferencia es menor que la distancia desde
      objetoComparador hasta segundoObjetoDeReferencia
            Retornar verdadero
      Sino
            Retornar falso
```

En el algoritmo anterior, la pregunta devolverá **verdadero** si el objeto comparador está más cerca del primer elemento que se haya tomado como referencia, o **falso** en caso contrario. Es importante dejar en claro lo siguiente:

El criterio que se empleó para decir cuándo es verdadero o cuándo es falso es completamente arbitrario, y queda estrictamente sujeto al criterio del programador.

Con esto podríamos decir que estamos prácticamente listos para programar esta pregunta en Alice, pero nos queda un detalle pendiente: el nombre de la función. Y como hemos dicho con anterioridad, esto es algo que queda a criterio del programador. Al igual que en el caso de los parámetros, lo que importa es que el rótulo que le ponga sea lo más explicativo posible, para efectos de depuración y de documentación del programa. Dicho eso, le pondremos **primerObjetoMasCerca**. El código se muestra en la Figura 69:

Figura 69. Implementación del algoritmo para comparar distancias entre dos objetos.



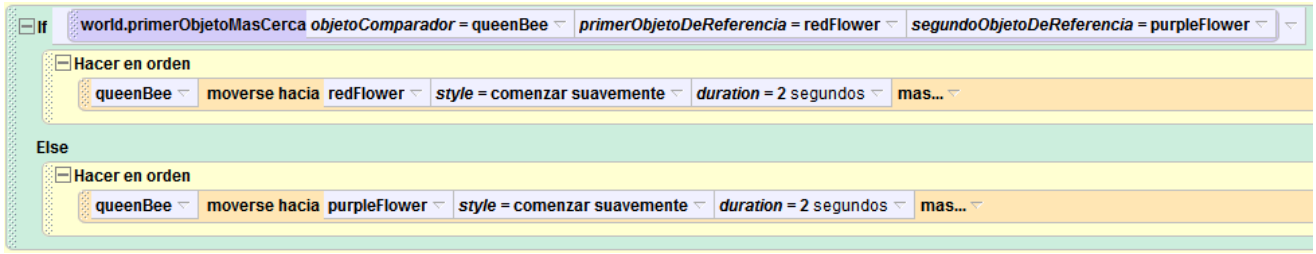
[21] Es importante destacar que esta clase de problemas se puede abordar de muchas maneras diferentes. Lo que acá se muestra es sólo uno de los tantos caminos que existen.

Tenga en cuenta que la gran diferencia entre un método y una función, es que esta última debe terminar con una instrucción de **retorno**, que a su vez es la responsable de **almacenar la respuesta a la pregunta**. Si la instrucción de retorno no se escribe como parte de la función, Alice no será capaz de generar respuesta.

Cómo llamar o invocar a una función

Una vez que la pregunta se ha escrito, puede ser invocada a través de algún método. Lo que ahí dice es que si la flor de color rojo está más cerca de la abeja, entonces deberá apuntar hacia ella, y si no, hacerlo en dirección a la rosada. Se ha agregado una línea en la cual la abeja dice a qué flor se va a dirigir

Figura 70. Invocando a una función dentro de una expresión condicional.



Observación: en la construcción de la función se han seguido estos pasos:

- Arrastrar una sentencia **if/else** hacia el editor, dejando como **true** el interior del **if**.
- Usted debe tener claro que la distancia se expresa a través de un número, y como tal, se puede comparar con otro, por medio de algún operador relacional como menor, mayor, menor o igual o mayor o igual. Entonces, por lo mismo, vamos a arrastrar la etiqueta **a<b** hacia el editor de código.
- El símbolo **a** se reemplaza por la función **distancia a**, y elegimos el valor de la distancia desde el parámetro **objetoComparador** hasta **el primer objeto de referencia**. Recuerde que tanto **a** como la distancia **son números**, y por lo mismo, podemos reemplazar uno por otro. Si tratara de mezclar números con cadenas de caracteres, Alice no lo dejaría. Haga la prueba.

Una pregunta más compleja

Un aeroplano y un helicóptero están sobrevolando el mismo espacio, a una cierta altura con respecto del suelo, como se muestra en la imagen de la derecha. Cuando dos naves comparten el mismo espacio de vuelo, el riesgo de colisión es más grande. Queremos escribir una pregunta que pueda ser utilizada para determinar si acaso el objeto no se encuentra en peligro de choque con otro. En caso que la distancia fuera muy estrecha, el aeroplano podrá invocar un método para evitar que esto suceda. Uno de los factores que vamos a emplear para

determinar si dos aeronaves están en peligro de colisión es la diferencia de altura que tengan con respecto al suelo. En este ejemplo, la pregunta **diferenciaDeAlturas** será diseñada para devolver **verdadero** si esta diferencia es menor a 10 metros. De lo contrario, la pregunta devolverá **falso**. Una posible solución que añade más ingredientes de lo que hemos comentado, es la siguiente:[22]

Tabla 14. Ejemplo que combina varias sentencias condicionale.

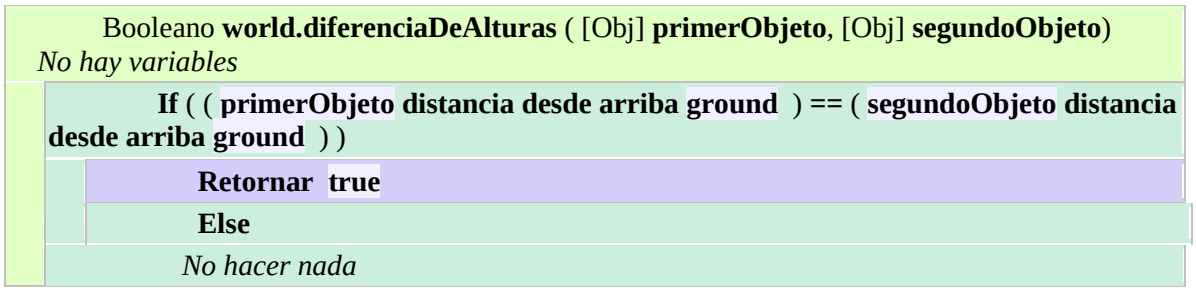
```
Función: diferenciaDeAltura
Parámetros: primerObjeto, segundoObjeto
      Si la altura del primerObjeto es igual a la altura del
      segundoObjeto
            Retornar verdadero
      Sino
            Hacer nada
      Si el primerObjeto está por encima del segundoObjeto, y la
      distancia entre el primerObjeto con el segundoObjeto es < 10
            Retornar verdadero
      Sino
            Hacer nada
      Si el segundoObjeto está por encima del primerObjeto, y la
      distancia entre el segundoObjeto con el primerObjeto es < 10
            Retornar verdadero
      Sino
            Hacer nada
      Retornar falso
```

En esta pregunta, los dos objetos — cuyas alturas se deben comparar — se transfieren a la pregunta como parámetros. Se presentan tres condiciones posibles:

1. Las dos aeronaves pueden tener la misma altura.
2. El aeroplano puede estar por encima del helicóptero, a menos de 10 metros.
3. El helicóptero puede estar por encima del avión, a menos de 10 metros.

Basta con que cualquiera de estas condiciones se cumpla (o que sea verdadera, que es lo mismo) para que la función devuelva **true**. Si ninguno de estos casos es cierto, la pregunta devuelve un valor predeterminado de **falso**, que es el que aparece en la última línea. El código de la pregunta definida por el usuario se presenta en la Figura 71:

Figura 71. Ejemplo de algoritmo con varias sentencias if/else.



[22] Tenga en cuenta que lo importante en esta etapa es «*soltar la mano*» con el uso de las funcionalidades de Alice, especialmente con el uso de múltiples **if/else** anidados.

```
If ( ambos ( primerObjeto está por encima de segundoObjeto ) y ( ( primerObjeto distancia desde arriba segundoObjeto ) < 10 ) )
    Retornar true
    Else
    No hacer nada
If ( ambos ( segundoObjeto está por encima de primerObjeto ) y ( ( segundoObjeto distancia desde arriba primerObjeto ) < 10 ) )
    Retornar true
    Else
    No hacer nada
Retornar false
```

El código para esta pregunta consta de tres sentencias consecutivas **if/else**. Si la primera de ellas se cumple, entonces **devuelve verdadero y termina la revisión de la función**. Si no se cumple, pasa a ejecutar lo que indique el **else**; sin embargo, como el código tiene la instrucción **hacer nada**, entonces simplemente pasará al segundo bloque de sentencias **if/else**. De esta manera, si las tres condiciones son falsas, la pregunta pasará a la última línea, donde está el comando para decirle que retorne el valor **falso**.

Algunos programadores prefieren un estilo **en cascada** para escribir las declaraciones **if/else**, como se muestra en la Figura 72. El código funciona exactamente de la misma manera anterior, salvo que a lo mejor la lectura no es tan directa. En términos algorítmicos, son equivalentes, y por lo tanto, ambas expresiones son igual de válidas.

Figura 72. Ejemplo de sentencias «*if/else*» anidadas.

```
Booleano world.diferenciaDeAlturas ( [Obj] primerObjeto, [Obj] segundoObjeto)
No hay variables
If ( ( primerObjeto distancia desde arriba ground) == ( segundoObjeto distancia desde arriba ground mas... ) )
    Retornar true
    Else
    If ( ambos ( primerObjeto está por encima de segundoObjeto mas... ) y ( ( primerObjeto distancia desde arriba segundoObjeto mas... ) < 10 ) )
        Retornar true
        Else
        If ( ambos ( segundoObjeto está por encima de primerObjeto mas... ) y ( ( segundoObjeto distancia desde arriba primerObjeto mas... ) < 10 ) )
            Retornar true
            Else
            Retornar false
        Retornar false
```

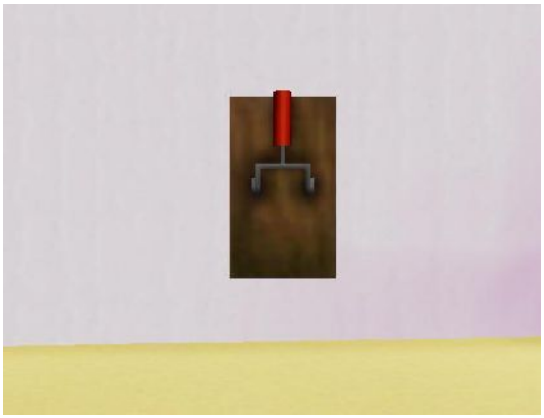
Ejercicios

1 Una función para el mundo de Zeus.

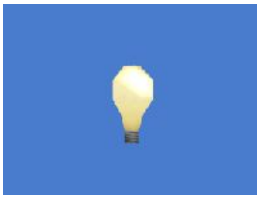
Este ejercicio es una modificación del mundo de Zeus de la unidad anterior. Diseñe una pregunta de tipo booleana llamada **esFilofoso**, que retorne verdadero si acaso el objeto que se pasa como parámetro es alguno de los cuatro filósofos griegos, y falso en caso contrario. A continuación, modifique el método **tragediaGriega**, de tal manera que aparezca incorporado en el código la pregunta **esFilosofo**, para saber si Zeus debe disparar el rayo sobre el objeto que el usuario haya cliqueado.

2 Interruptor.

Cree un mundo con un objeto de tipo switch o conmutador, como el que se muestra a la derecha. Diseñe un método llamado **subirBajarInterruptor**, que permita que el conmutador suba o baje cuando el usuario haga clic en él, produciendo un giro de media revolución. Para determinar la posición de la manilla, usted deberá construir una función **estaArriba** que retorne verdadero en caso que el mango esté hacia arriba, y falso en caso contrario.



3 Bombilla.

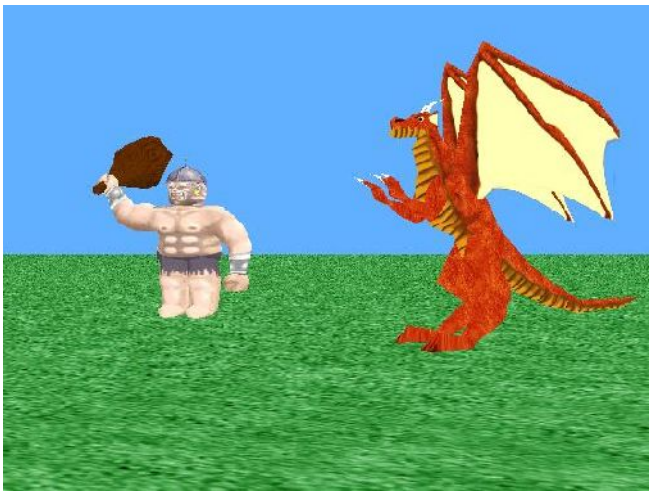


Cree un mundo con una ampolleta, que tenga asociada un método denominado **interruptor**, que permita encenderla y apagarla. Si está encendida, el color debe ser amarillo, o negro en caso contrario. Escriba una pregunta booleana **estaEncendida** que retorne verdadero en caso que la ampolleta esté prendida, o falso en caso contrario. Cuando se hace clic en el objeto, se debe encender o apagar.

4 Serpiente para el almuerzo.

Diseñe un mundo que contenga una pecera y una serpiente en el agua. La escena inicial debe parecerse a la imagen de la derecha. El pez está hambriento, y la serpiente parece ser un buen elemento del menú. Cada vez que el usuario presione la barra espaciadora, el pez se moverá una cierta distancia — que usted determinará — hacia adelante. Cuando el pez esté a menos de un metro de distancia de la serpiente, deberá comérsela. Para que parezca que el pez se ha devorado a la serpiente, haga que la serpiente desaparezca mediante la configuración de la propiedad **isShowing**.

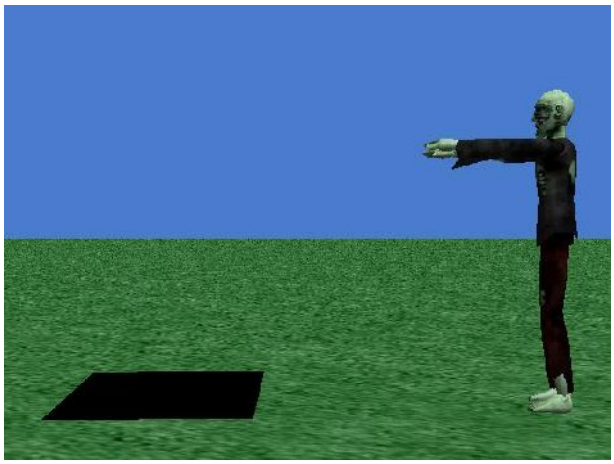
5 Troll, pero no tonto.



En la escena que se muestra a la izquierda, aparece un troll con un dragón. El troll trata de atemorizar a los dragones, pero al mismo tiempo es lo suficientemente astuto para no acercarse demasiado. Habiendo 5 metros de distancia entre uno y otro, el troll no corre peligro. Cada vez que el usuario presione la barra espaciadora, el troll deberá avanzar hacia adelante. Construya una pregunta que permita averiguar cuándo el hombre deba detenerse.

6 Mundo de Zombies.

Diseñe un mundo con un zombie, y una tumba abierta, que vamos a representar mediante un cuadrado negro en el suelo. En una escena de una película de miedo, el zombie camina hacia adelante y cae en la tumba. En esta animación, cada vez que el usuario presiona la barra espaciadora, el personaje debe caminar hacia adelante. Construya una pregunta booleana llamada **cercaDeTumba** que sea verdadera cuando el zombie esté a medio metro de ella. Cuando esto sea así, el objeto deberá caer dentro.



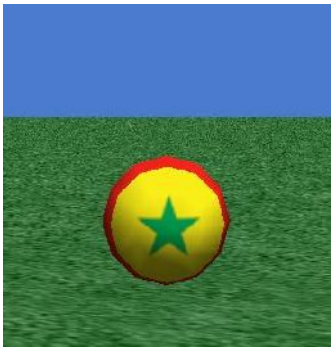
Otros tipos de preguntas

Como usted sabe, el tipo de pregunta se basa según el valor que devuelva, y hasta el momento hemos trabajado sólo con aquellas de naturaleza booleana. Veamos otros ejemplos, para que usted vez funciones de otros tipo que también son útiles.

Funciones de números

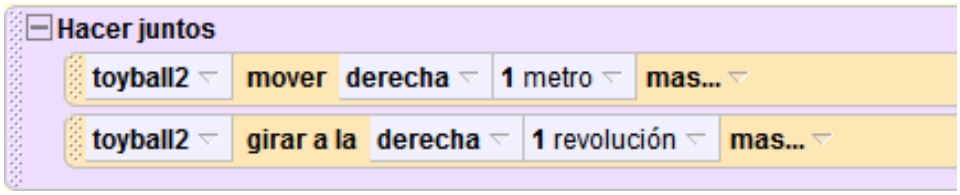
Hemos utilizado preguntas que devuelven un valor numérico, como por ejemplo la **distancia a**, que sirve para determinar la separación entre dos objetos. Las preguntas que devuelven un valor numérico serán útiles en muchas situaciones: en un videojuego, podemos construir una función que retorne el puntaje parcial que tenga un jugador, o la cantidad de «*vidas*» que le queden hasta que termine.

Siempre es útil comenzar con un ejemplo sencillo. Para ilustrar un tipo de función numérica, consideremos una pelota de juguete como la que se muestra a continuación:

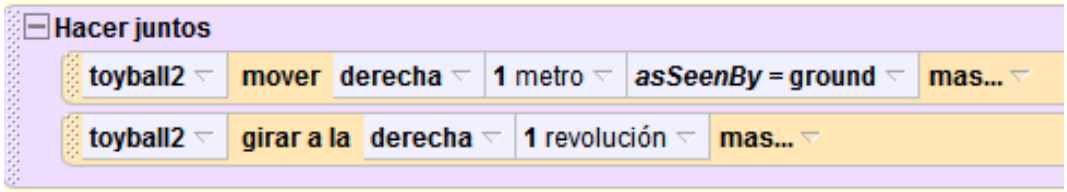


Lo que vamos a hacer es una simulación de cómo puede rodar esta bola. Piense cómo la pelota puede girar hacia su derecha: en este caso, la instrucción **roll** no nos sirve, porque lo que se consigue es que ésta permanezca fija en el suelo, pero sin moverse. La idea es que se mueva a lo largo del terreno.

Para hacer que la bola realmente se mueva, se requieren de dos acciones simultáneas: que se desplace hacia la derecha, y que gire en el mismo sentido, de acuerdo a lo que se muestra a continuación:



Sin embargo, las pruebas de este código también son poco efectivas. Queda la impresión que la bola avanzara hacia adelante, y que al final de la ejecución tendiera a retroceder. Una solución a este problema es **asSeenBy**. El efecto deseado es tener el balón sobre la tierra, y por lo tanto el suelo parece ser un probable objeto para utilizarlo como referencia, como se ilustra a continuación:



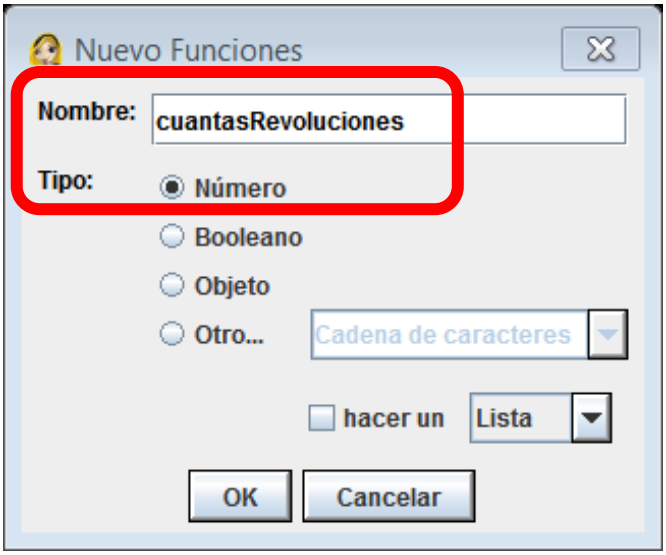
El balón fue hecho para avanzar más de un metro. Supongamos que la bola deba avanzar 10 metros. Entonces: ¿cuántas revoluciones debe hacer para cubrir una distancia de 10 metros en dirección hacia adelante? La creación de un movimiento rotacional que cubra esa distancia es difícil, ya que la cantidad de giros — o revoluciones — que tenga que dar es proporcional a su diámetro. Para cubrir la misma distancia en avance a lo largo de la tierra, una pequeña bola debe girar hacia adelante una mayor cantidad de veces que una más grande.

Por supuesto, el número de revoluciones necesarias para que la pelota ruede 1 metro a su derecha se pueden encontrar por prueba y error. O bien, el número de revoluciones puede ser calculado a mano, utilizando la siguiente fórmula:

$$\text{número de revoluciones} = \frac{\text{distancia}}{\text{diámetro} * \pi}$$

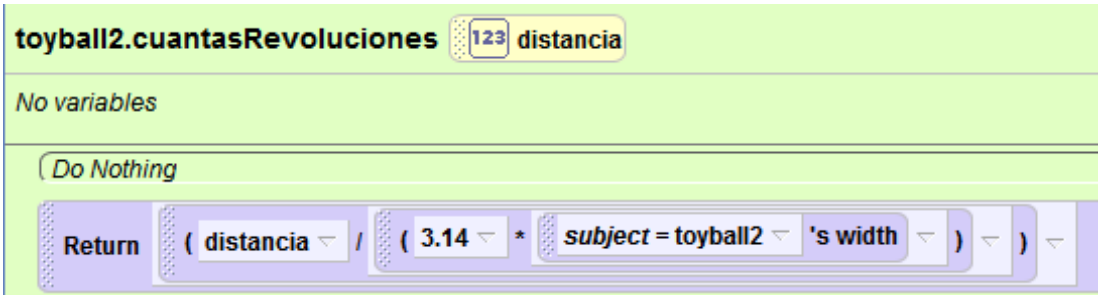
Pero, cada vez que el balón cambie de tamaño o la distancia, este cálculo tendría que llevarse a cabo de nuevo, y el código tendría que ser modificado. Este es el momento apropiado para introducir una pregunta de tipo numérica. A nosotros, más que importarnos el modelamiento físico de la situación, nos interesa el tipo de función que se tenga que definir. En este caso, como sólo aparece un objeto involucrado, vamos a trabajar con una pregunta de tipo objeto, que tiene la ventaja de poder ser reutilizada a futuro, al igual como ocurre con los métodos. De manera similar a las de tipo booleana, pediremos que la función **cuantasRevoluciones** retorne un número, tal como se muestra en la Figura 73:

Figura 73. Prototipo de una función numérica.



El parámetro que recibe como entrada le llamamos **distancia**, y el número de rotaciones se calcula dividiendo la **distancia** que se quiere recorrer, por el diámetro — que en Alice equivale al ancho del objeto — multiplicado por la constante π, que vamos a aproximar por el número 3,14. El valor calculado es la respuesta a la pregunta. La instrucción **retornar** le dice a Alice lo que tiene que generar como respuesta frente al cálculo realizado. La Figura 74 ilustra el código asociado:

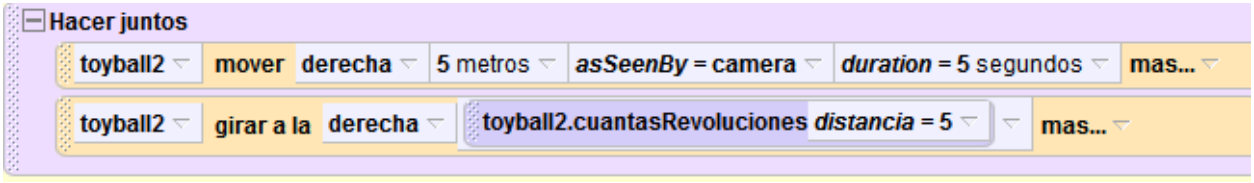
Figura 74. Implementación de la cantidad de revoluciones que da una esfera.



En las preguntas de tipo numéricas, el orden en que se ingresen los parámetros se tiene que organizar con cuidado, ya que Alice también respeta la jerarquía de los operadores aritméticos — paréntesis, potencias, multiplicaciones, divisiones, sumas y restas.

Las pruebas

Ahora que hemos definido la pregunta, podemos usarla en un programa. En la imagen adjunta se muestra una prueba de la función **cuantasRevoluciones**. Una distancia arbitraria de 5 metros se usa para el movimiento hacia adelante, y proporciona un parámetro para la instrucción **girar a la derecha**:



Utilizando una función con un operador relacional

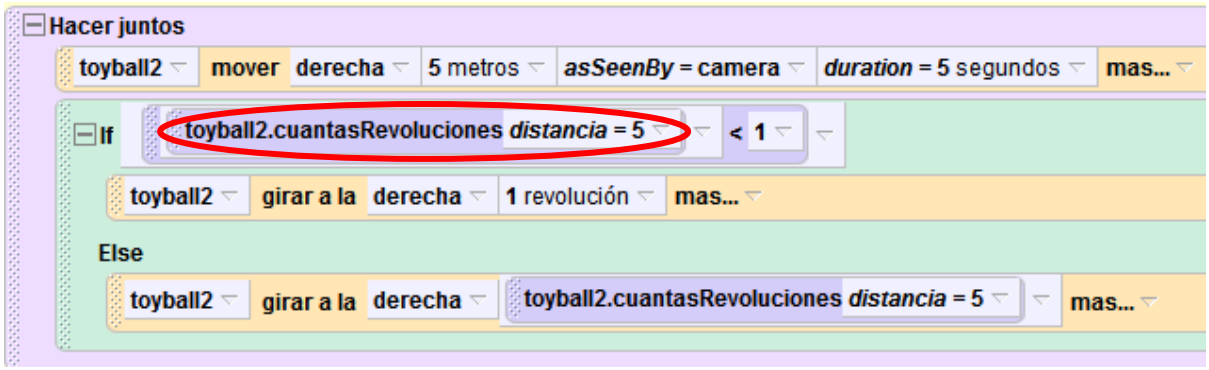
Si la distancia es pequeña y el balón es grande, es evidente que la pelota no podrá hacer una revolución completa. Entonces:

¿Existirá algún mecanismo para asegurar que el objeto alcance a dar por lo menos una vuelta?

Una solución consiste en usar una sentencia **if/else** para comprobar el valor de **cuantasRevoluciones**, en el sentido que si el número es menor que uno, podamos forzarla a que haga un giro completo, o que en caso contrario ruede tantas vueltas como lo indique el resultado de la operación anterior.

La Figura 75 muestra el código modificado mediante una llamada a **cuantasRevoluciones** como parte de una expresión lógica. En este caso, se ha empleado el operador relacional **menor que** para comparar el valor devuelto por **cuantasRevoluciones** con 1. Como el resultado de la función **menor que** es de tipo booleano, entonces el **if** queda bien construido en términos lógicos.

Figura 75. Llamando a una función dentro de una expresión condicional.

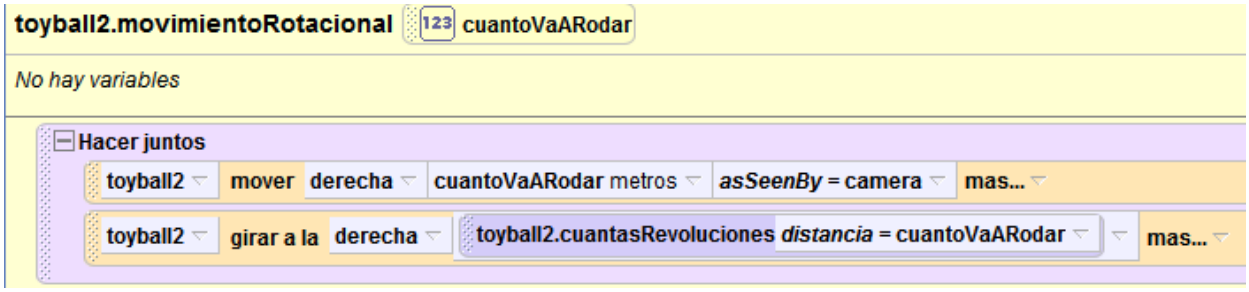


Abstraer un personaje de un método

Lo que hemos visto con anterioridad ilustra la manera de cómo implementar el movimiento real de una bola, que se construye a partir de varios pasos pequeños. Sin embargo, si pensamos en la construcción de un mundo más grande, donde la pelota fuera una de las tantas partes que lo componen, entonces podríamos encapsular todo lo que hemos escrito en un sólo método **movimientoRotacional** que opere a nivel de objeto. Entonces, como la pelota ya tiene una función definida, podemos crear el método pasándole como parámetro la distancia que tenga que recorrer. Al final, podemos guardar esta bola como

un nuevo objeto, y luego agregamos al mundo tantas copias como sean necesarias. La Figura 76 muestra el código del método **movimientoRotacional**.

Figura 76. Ejemplo de método que llama a una función definida para un objeto.



Funciones de uso genérico

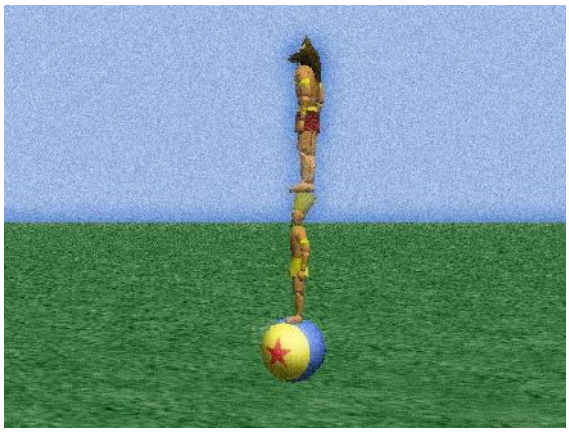
En el ejemplo de la pelota, notemos que una pregunta y un método de nivel de objeto fueron diseñados y construidos. Pero, está claro que la pregunta para calcular el número de rotaciones para la bola puede ser empleada en cualquier objeto de esta naturaleza ya que estamos haciendo uso de **funciones del mundo, que se caracterizan por estar disponibles para ser usadas por los objetos que lo compongan**. Las preguntas que se dan a este nivel se consideran de tipo **genéricas**.

Ejercicios

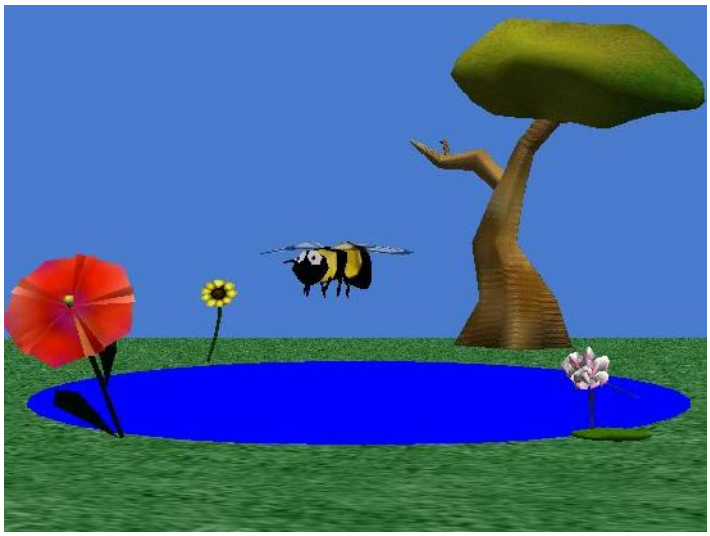
1 Acróbatas.

Diseñe un mundo que contenga una pelota de juguete, que tenga dos veces su tamaño original, y dos acróbatas a su elección que deben estar situados en la parte superior de la bola, como se muestra en la imagen adjunta:

Utilice el editor de vista cuádruple para asegurarse que los acróbatas estén de pie directamente uno encima de otro y estén centrados con respecto a la bola. Diseñe una animación de circo, donde los acróbatas puedan moverse con el balón, permanezcan encima de él, a medida que roda por el suelo. De manera simultánea, los artistas deben poner sus brazos a media altura, para ayudar a su equilibrio.



2 Abeja Scout.



Ha sido un caluroso y seco verano, y una colmena de abejas está desesperada por la necesidad de una nueva oferta de polen. Una de ellas tiene espíritu aventurero, y ha decidido salir a buscar el preciado elemento. Recree una escena inicial como la que se muestra en la imagen de la izquierda. Escriba un programa para que la abeja de vueltas alrededor del estanque de flores. En concreto, el insecto debe volar alrededor del perímetro de la laguna, que se ha modelado a través de un círculo. Escriba un método para hacer la abeja gire alrededor del perímetro del estanque, donde la circunferencia sea utilizada para guiar el movimiento. La fórmula para calcular el perímetro de una circunferencia es

Diámetro * π

El valor de π lo vamos a redondear en 3,14, y el diámetro corresponde al ancho del objeto. Escriba una pregunta que calcule y devuelva el perímetro del estanque. Luego, tiene que hacer que la abeja vuele alrededor del lago la cantidad de metros devuelto por la función de perímetro, mientras la abeja gira sobre su propio eje una revolución a la izquierda.

Resumen

Hasta el momento, la unidad presenta los conceptos fundamentales que tratan sobre sentencias condicionales, y de preguntas o funciones que el usuario puede definir. La instrucción **si** juega un papel importante en la mayoría de los lenguajes de programación, porque permite la ejecución de una determinada porción de código, en función del cumplimiento de ciertas condiciones lógicas que están asociadas con expresiones **boolenas**, que son las que gobiernan el curso que debieran seguir las acciones del programa o de la animación. Posterior a eso, revisamos algunos operadores relacionales que permiten establecer relaciones entre las propiedades de los objetos, que tienen como objetivo ampliar el conjunto de herramientas posibles para desarrollar con la mayor fiabilidad posible lo que el usuario necesite. Después, vimos cómo podíamos combinar expresiones booleanas con operadores relacionales, para así lograr construcciones de software cada vez más potentes.

También hemos estudiado cómo escribir nuestras propias preguntas, en función de las necesidades del libreto de trabajo, donde el principal beneficio corre por cuenta del nivel de abstracción que podemos lograr cuando queramos insertar nuestros nuevos objetos en mundos cada vez más complejos. Otro de los beneficios del uso de preguntas es que permiten la escritura de un código mucho más limpio, porque los cálculos están encapsulados en la pregunta. además, el uso de parámetros permiten formular preguntas genéricas que puedan ser aplicables a un mayor espectro de objetos.

Conceptos importantes vistos hasta el momento

- La sentencia **if/else** es un bloque de código de programa que permite la ejecución condicional de instrucciones.
- El parámetro de una sentencia condicional es una expresión lógica que determina si el código se puede ejecutar, en función de su cumplimiento. En caso de no ser así, se pasaba a las instrucciones del grupo **else**.
- Los operadores lógicos **and**, **or** y **not** se pueden usar para combinar condiciones booleanas simples en expresiones más complejas.
- Los operadores relacionales **<**, **>**, **>=**, **<=**, **==** pueden ser utilizados para comparar valores dentro de una expresión booleana.
- Las preguntas definidas por el usuario pueden ser escritas para devolver un valor booleano y también pueden ser escritas para calcular y devolver otros tipos de valores.

Proyectos

1 Portero.

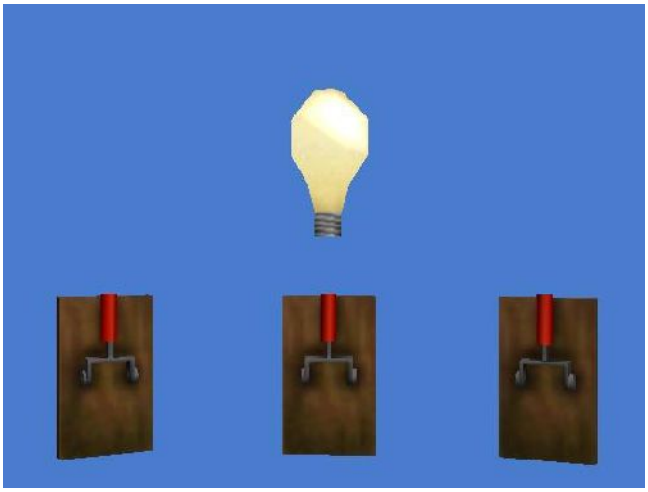
Construya un mundo con cuatro personajes a su elección, de tal manera que estén alineados a la misma distancia unos de otros. En este juego, uno de los objetos va a desempeñar el papel de portero, que deberá manejar una contraseña secreta para que el usuario abra una puerta oculta en la pirámide. Para encontrar la clave, el usuario debe reorganizar los objetos de la línea, hasta que el portero se encuentre en el extremo derecho de la misma.

Esto significa que si los objetos se cuentan de izquierda a derecha como 1, 2, 3 y 4, el portero deberá moverse hacia la posición 4. Cuando un objeto es cliqueado, se debe intercambiar con otro personaje, según estas reglas:

- El 1 se intercambia con el 4, si cualquiera de los se cliquea.
- El 1 se intercambia con el 3, si cualquiera de los se cliquea.
- El 2 se intercambia con el 4, si cualquiera de los se cliquea.

Haga que uno de los objetos que estén en la posición 1, 2 o 3 sea el portero (la elección queda a gusto del programador). El programa debe usar alguna pegunta de tipo booleano que devuelva verdadero cuando los objetos estén alineados y el portero se encuentre en la parte derecha de la línea, y falso en caso contrario. Cuando el portero está en la posición indicada, deberá mostrar un texto 3D con la contraseña, que usted como programador tiene que inventar.

2 Juego Binario.



Construya un mundo con tres interruptores y una ampolleta, tal como aparece en la imagen. Debajo de la palanca de cada interruptor, coloque una esfera invisible. Cuando la ampolleta esté apagada, tiene que quedar de color negro. En este juego, las posiciones de las palancas situadas en los interruptores representan un código binario. El número 1 se representa con una palanca levantada, en señal que existe un alto flujo de corriente eléctrica, mientras que el 0 se produce cuando algunas de ellas están abajo. De acuerdo a lo que se ve en la figura anterior, las tres palancas están configuradas, de tal manera que se puede leer el número 111, ya que todas están levantadas.

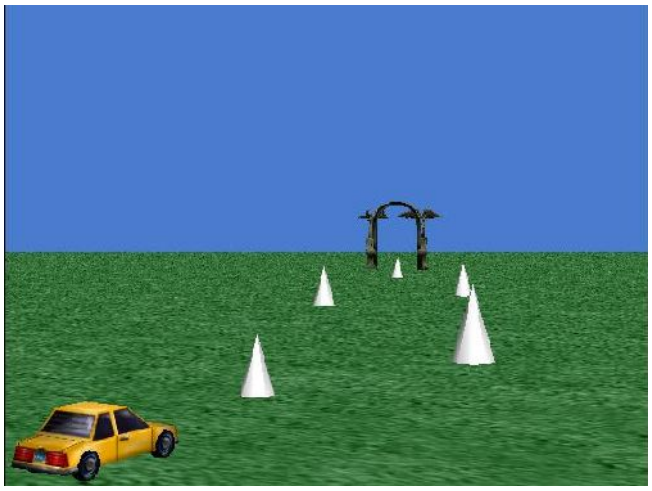
El código binario se determina al inicio del juego de forma completamente aleatoria, por medio de la función de mundo **número aleatorio**. La idea de este juego es que el usuario intente adivinar el código para que se encienda la bombilla. Para adivinar, el usuario debe hacer clic en las palancas para cambiar su posición: hacia arriba o hacia abajo según la ubicación actual que tenga. Cuando los tres interruptores estén en la posición correcta, la ampolleta se enciende.

Cada interruptor debe responder a un clic de ratón en su palanca.: si está hacia abajo, tiene que dar la vuelta hacia arriba, y viceversa. Para hacer un seguimiento de la posición actual de la palanca de un conmutador, se puede colocar una esfera invisible debajo del interruptor, de tal manera que se mueva en la dirección opuesta de la palanca.

El código del proyecto debe incluir una pregunta booleana que determine si el interruptor de la palanca está en la posición de arriba. La función debe utilizar un parámetro de objeto que especifique el interruptor que haya que controlar: no vale si hace una función para cada conmutador. Como sugerencia, puede asignarle un color a cada esfera que haga las veces de indicador sobre la correctitud de la posición de la palanca.

3 Prueba de Manejo.

Cree un mundo que simula una prueba de conducción, que debe estar conformado por un coche, 5 conos, y una puerta, tal como lo muestra esta imagen:



También construya dos textos 3D que digan «*Aprobado*», e «*Inténtelo de nuevo*». Utilice la propiedad **isShowing** para que ambos sean invisibles en la escena inicial. En esta prueba, el usuario maneja el vehículo con las flechas de dirección para ir hacia adelante, izquierda o derecha. La idea es que pueda oscilar alrededor de los conos sin que los golpee. Basta con que el conductor impacte a uno, para que no pase la prueba, y se active el letrero de intentarlo de nuevo. Por el contrario, si el usuario sortea con éxito los conos y pasa a través de la puerta, entonces aparece el mensaje de aprobación.

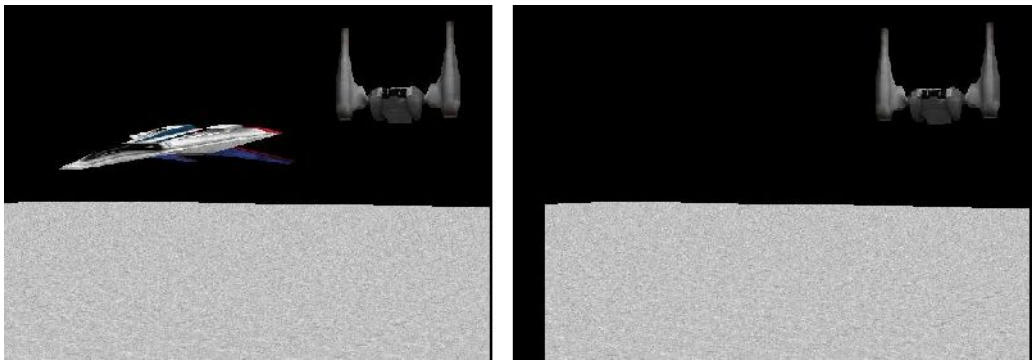
Escribir una pregunta definida por el usuario con nombre **muyCerca**, que devuelva verdadero cuando la distancia desde el auto hacia el cono sea inferior a los dos metros, y falso en caso contrario. Además, escriba una pregunta llamada **testAprobado**, que retorne verdadero cuando el usuario haya pasado la prueba, lo que ocurre cuando el vehículo haya traspasado la puerta. Trabaje bajo el supuesto de que el usuario es honesto para jugar, y que no va a pasar por alto los conos.

3.3 Consejos y Técnicas

isShowing vs. opacidad

En los programas de juegos, es frecuente el caso cuando los objetos se vuelven invisibles por efectos de camuflaje. Por ejemplo, una nave espacial es capaz de mimetizarse en el espacio estrellado para que no sea visible al radar del enemigo. La Figura 77 muestra la escena espacial antes y después que la nave use camuflaje.

Figura 77. Antes y después de un camuflaje.



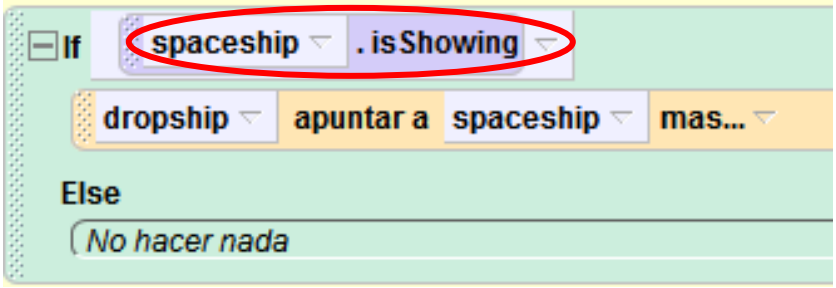
Como se explicó con anterioridad, Alice ofrece dos formas para hacer que un objeto sea invisible:

1. Haciendo que la propiedad **isShowing** sea **falso**.
2. Ajustando la **opacidad al 0%**.

En este ámbito del espacio, el código para encubrir la nave espacial puede ser parecido a este:

En los mundos donde la visibilidad se utiliza como parte de la animación, es común verla dentro de alguna sentencia condicional **if/else**, como en este caso:

Figura 78. Ejemplo de uso de la propiedad «*isShowing*» en una sentencia condicional.



Lo que no es tan obvio sobre el uso de visibilidad en una expresión booleana es que el código debe ser coherente en la utilización de cualquier **isShowing** o la **opacidad**. El punto es que **isShowing es una condición booleana**, mientras que la **opacidad es de naturaleza numérica**. Por lo tanto, **isShowing es diferente de la opacidad**. Cuando un objeto tiene una opacidad del 0%, es invisible ante nuestros ojos, pero eso no significa que Alice transforme internamente el valor de **isShowing** a falso. Del mismo modo, cuando se hace **isShowing** igual a falso, Alice no asume que la opacidad sea del 0%. La moraleja de esto es la siguiente:

Si utiliza isShowing para hacer invisible algún objeto, vuelva a ocupar isShowing para comprobar su visibilidad, y no se guíe por el porcentaje de opacidad. O bien, si va a trabajar con la opacidad, siga empleando esta propiedad, y no la mezcle con la otra.

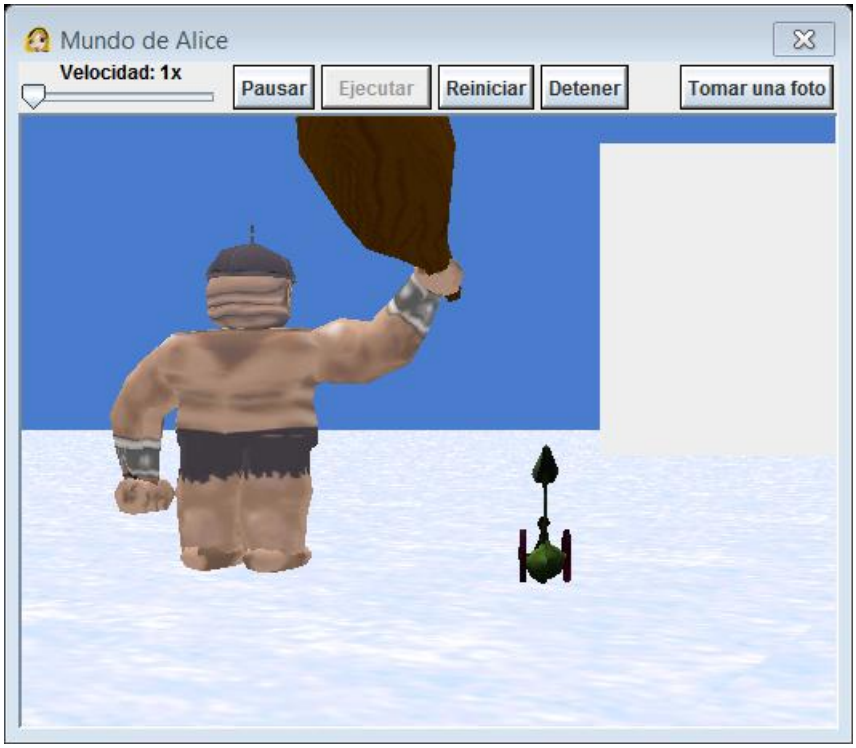
Vista desde la parte posterior

La imagen adjunta muestra un mundo de fantasía donde los objetos han sido añadidos a la escena y se han organizado crear una escena inicial de una animación cualquiera. Desde nuestra perspectiva, la cámara nos permite tener un acceso a la parte frontal.

En algunos mundos, puede que sea necesario mover la cámara alrededor de la escena inicial, para ver la parte posterior de la misma. Para esto, sabemos que la casa en el árbol corresponde al objeto que está más alejado de ella. Entonces, en el editor de código pedimos que se desplace hacia adelante la distancia que la separe de la casa, de tal manera que a continuación proceda a hacer un giro de media revolución hacia la izquierda. Después. Usted se puede desplazar todo lo que estime necesario para dar un vistazo posterior a la escena:

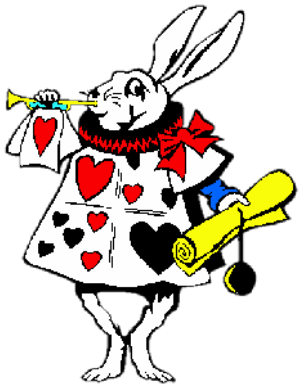
```
camera  mover  hacia adelante  ( camera  distancia a  treeHouse  + 20 )  mas...
camera  girar a la  izquierda  0.5 revoluciones  mas...
camera  mover  hacia adelante  100 metros  mas...
```

Los 100 metros hacia adelante se colocaron para dar un valor referencial, pero nada más que eso. Si ejecuta la animación, podrá ver algo como esto:



Iluminando la vista trasera

En algunas ocasiones, puede que sea necesario agregar más luz a la parte posterior de una escena. Para eso, basta con agregar una ampolleta, y notará la diferencia.



Segunda Parte

Bucle «*para*» o «*for*»

Al comienzo de esta unidad, hemos considerado **if/else** para apoyar la toma de decisiones durante el tiempo de ejecución de un programa. En esta parte, nos fijamos en un segundo tipo de instrucción de control, que se conocen bajo el nombre de **bucle** o **lazo**, que consisten en permitir las repeticiones de ciertas porciones de código las veces que el programador estime necesario.

En Alice, un bucle es un bloque de una o más instrucciones que se ejecutan una y otra vez una cierta cantidad de veces. En los lenguajes de programación, se trabaja con variables de tipo **contador**, que permiten almacenar la cantidad de veces que se está ejecutando el **cuerpo de la repetición**. Examinaremos en primer lugar los lazos que se llaman **para** o **for**, después veremos **recursión**, para terminar con los **mientras** o **while**.

3.4 Bucles

Introducción

Cuando diseñamos mundos más sofisticados, el código tiende a ser más largo. Nos encontramos también con la necesidad de repetir una o varias instrucciones durante la animación, para que las mismas acciones se repitan una y otra vez. Es posible que usted haya construido mundos donde haya experimentado la necesidad de arrastrar un método en reiteradas oportunidades.

Cuando esto ocurría, la única manera hasta el momento era arrastrar la porción de código hasta el editor la cantidad de veces que fuera necesario. El enfoque de esta parte apunta a enseñarle cómo funciona el tema de la repetición, y cuáles son los grandes beneficios que conlleva su uso.

La necesidad de repetición

El mundo que se muestra en esta figura presenta a un pequeño conejo se coló en el jardín del vecino. El animal se percata que ha estado creciendo el brócoli que plantó el dueño de casa hace unas semanas. Esto le ha abierto el apetito a nuestro amigo, quien no va a encontrar nada mejor que ponerse a saltar y devorar la plantación que con tanto esfuerzo cultivó el esforzado vecino. Pero justo en el momento en que el pequeño estaba a punto de comer este vegetal, aparece su papá justo en la puerta del jardín, después de lo cual el travieso animal baja su cabeza con tristeza, y se retira raudo del lugar. En esta descripción del programa, supongamos que ya hemos escrito un método para el conejo, que le llamaremos **saltoConejoChico**, y que se muestra en la Tabla 15:

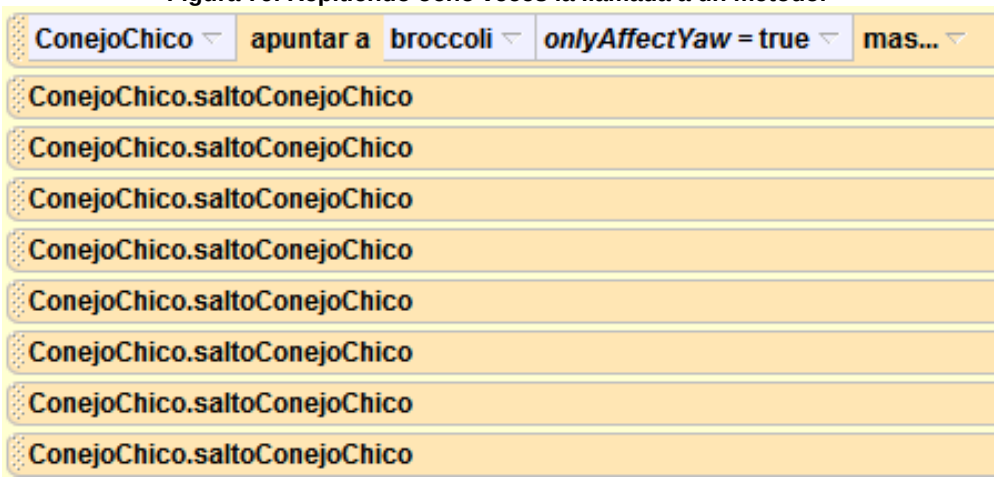
Tabla 15. Implementación del algoritmo del salto del conejo.

ConejoChico.saltoConejoChico ( )
No hay variables
Hacer juntos
Hacer en orden
// El conejo se mueve
Hacer juntos
ConejoChico mover arriba 0.5 metros duration = **0.25** segundos **mas...**
ConejoChico mover hacia adelante 0.4 metros duration = **0.25** segundos **mas...**
Hacer juntos
ConejoChico mover abajo 0.5 metros duration = **0.25** segundos **mas...**
ConejoChico mover hacia adelante 0.4 metros duration = **0.25** segundos **mas...**
Hacer en orden
// La pata derecha del conejo simula un salto
ConejoChico.hipR.footR girar a la hacia adelante 0.12 revoluciones duration = **0.25** segundos **mas...**
ConejoChico.hipR.footR girar a la hacia atrás 0.12 revoluciones duration = **0.25** segundos **mas...**
Hacer en orden
// La pata izquierda del conejo simula un salto
ConejoChico.hipL.footL girar a la hacia adelante 0.12 revoluciones duration = **0.25** segundos **mas...**

ConejoChico.hipL.footL **girar a la** **hacia atrás** **0.12** revoluciones duration = **0.25** segundos **mas...**

En esta parte, el método no es lo principal, sino que vamos a suponer que estas acciones las repite un total de 8 veces antes de ser descubierto. Una posible manera de animar esto es como se exhibe en la Figura 79. Ahí, se ha colocado un bloque **Hacer en orden** dentro de **my first method**, que contiene las ocho repeticiones de **saltoDelConejoChico**.

Figura 79. Repitiendo ocho veces la llamada a un método.



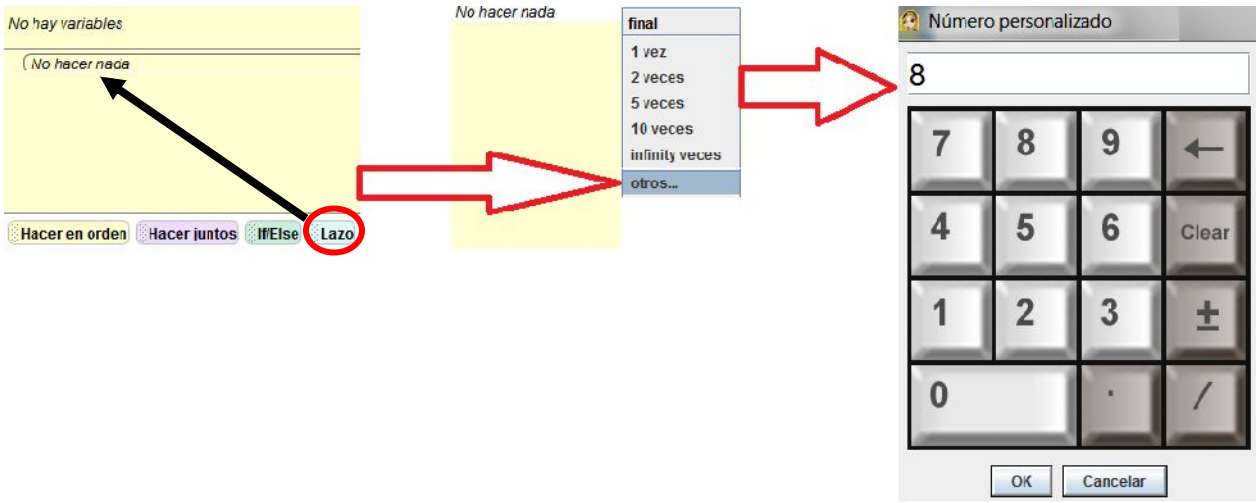
En realidad, no hay nada de malo en este código, ya que hace lo que se tiene previsto. Pero, es tedioso para arrastrar ocho instrucciones al mundo. No por flojera, sino porque esta cantidad perfectamente podría haber sido 100, 200 o 500, lo que de por sí justifica con mayor peso la aseveración anterior.

En este sentido, Alice ofrece una instrucción especial denominada **bucle**, que permite la inclusión repetitiva de código sin tener que hacer mucho trabajo. En materia de programación, el código del programa que se repite «*una y otra vez*» se llama **iteración**.

Usando un bucle

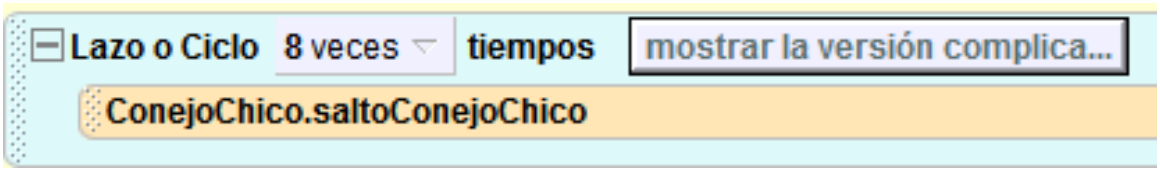
Para crear un bucle en un programa, se debe arrastrar el ícono **Lazo** hasta el editor. A continuación, aparece un menú emergente que pregunta por la cantidad de veces que se vaya a repetir. En este caso, 8 fue lo que se escribió, tal como se observa en la Figura 80.

Figura 80. Incorporación de un lazo o bucle al editor de código.



Entonces, se invoca al método y se arrastra el método del salto del pequeño conejo hacia dentro del bloque, como se muestra en la Figura 81. Cuando se ejecuta el programa, las acciones serán las mismas que en el caso anterior, con la diferencia que en este último caso, la instrucción se ha escrito de una manera mucho más rápida y sencilla de escribir.

Figura 81. Simplificación de las llamadas a un método, gracias a un bucle.



Recuento

En este ejemplo, el bucle le dice a Alice que debe ejecutar el método **saltoDelConejoChico** 8 veces. Tenga en cuenta que por razones lógicas, **la cantidad de veces que se repita un bucle debe ser un número positivo**.

Observemos además que una de las opciones que ofrece el menú emergente es **infinito**, que veremos con posterioridad. A modo de adelanto, si usted la marca, el lazo se ejecutará hasta que se detenga el programa. En este ejemplo, si infinito estuviera seleccionado, nuestro pequeño amigo nunca iba a parar de saltar, y por lo tanto, su papá no lo iba a descubrir, ya que esa parte del código no se iba a ejecutar.

Bucles y «hacer en orden» con «hacer juntos»

Hemos colocado una sola instrucción en un bloque de lazo para este ejemplo, pero no hay problema si queremos que varias instrucciones adicionales se agreguen dentro del **cuerpo del bucle** para que participen de la repetición. Asimismo, podemos anidar

estructuras **Hacer en orden** o **Hacer juntos** dentro de un lazo para controlar cómo las acciones se repiten. Es importante destacar que:

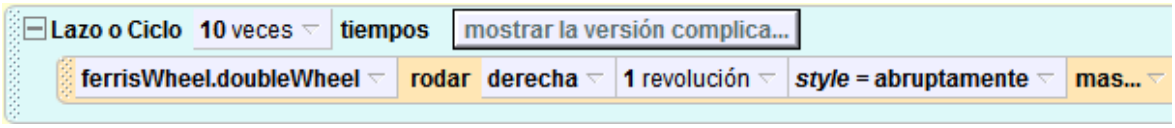
Si vamos a utilizar un bloque Hacer en orden o Hacer juntos dentro del bucle, Alice asume que las instrucciones se han de realizarán en orden secuencial.

Anidamiento de bucles

Supongamos un escenario que contiene una noria con dos ruedas, como se ilustra a la derecha. Se debe crear una animación que simule el funcionamiento de esta estructura: la doble rueda tiene que girar en el sentido de las agujas del reloj — o hacia la derecha, según la terminología de Alice ---, mientras que por separado cada una de las ruedas tienen que hacerlo hacia la izquierda. En la

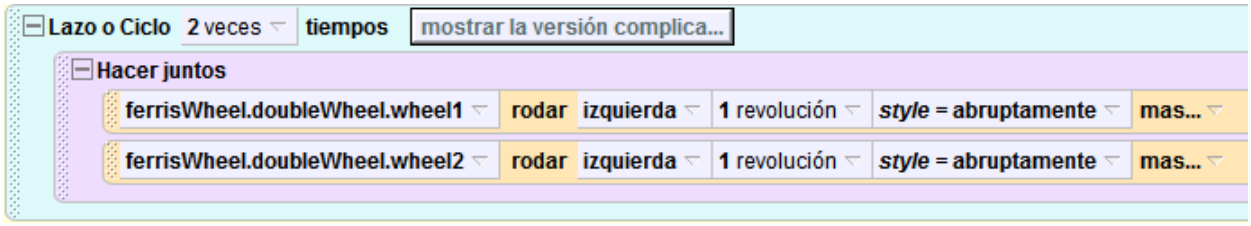
Figura 82 se puede apreciar el código de un bucle que permite hacer rodar la doble rueda de la noria diez veces. La razón por la que se eligió el estilo abrupto es porque si se hubiese trabajado con el predeterminado — que es más suave —, la animación se habría tornado más lenta.

Figura 82. Movimiento de una noria en el sentido de las agujas del reloj.



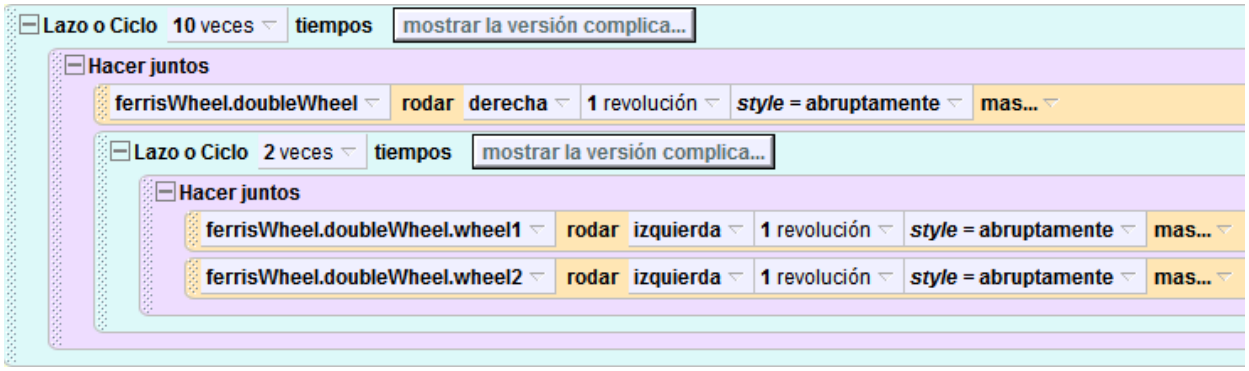
El código que se muestra en la Figura 83 permite la rotación simultánea de las ruedas interiores de la noria durante dos vueltas.

Figura 83. Incorporación de instrucciones en paralelo dentro de un bucle.



Ahora, los movimientos rotatorios que hemos definido — doble rueda y de las dos ruedas — se pueden combinar, tal y como se aprecia a continuación:

Figura 84. Ejemplo de anidamiento de bucles.



Es interesante observar el funcionamiento de los bucles anidados. Cada vez que el bucle externo se ejecuta una vez, el interno se ejecuta dos veces. De acuerdo a lo que se ve, el giro de la doble rueda toma dos segundos para completar su rotación, mientras que las ruedas requieren de uno, que es la duración predeterminada. En total, las ruedas interiores girará 20 veces hacia la izquierda el doble que gira 10 veces en el sentido del reloj.

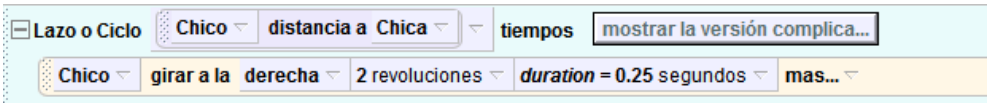
Es interesante observar el funcionamiento de los bucles anidados. Cada vez que el bucle externo se ejecuta una vez, el bucle interno se ejecuta dos veces. Tenga en cuenta que doble rueda toma dos segundos para completar su rotación, y las ruedas interiores requerirán 1 segundo (la duración predeterminada) para completar sus rotaciones, pero se gira dos veces. En total, las ruedas interiores girará 20 veces hacia la izquierda el doble que gira 10 veces en el sentido del reloj.

Nota técnica sobre el bucle

En los ejemplos presentados hasta el momento, el número de veces que un bloque de código fue una cantidad específica: 2, 8 o 10, pero el recuento puede ser también una pregunta que devuelva un número. Por ejemplo, si tenemos a un niño que está a 6 metros de distancia de una niña, podríamos escribir

este código, que se va a ejecutar un total de seis veces:

Figura 85. Ejemplo de bucle que se repite de acuerdo al resultado de una función.



Ejercicios

1 Conejo atrapado.

Este ejercicio tiene como propósito completar el problema del conejo que quería comer los brotes de brócoli del vecino. Como usted recordará, la animación no se completó, ya que estábamos preocupados de los bucles, y la parte final cuando llega el papá y lo sorprende no la consideramos. Complétela.

2 Formando un cuadrado.

Este ejercicio apunta al uso de bucles anidados. Papá conejo le ha estado enseñando a su hijo algunos conceptos básicos de geometría, y la lección del día giró

en torno a los cuadrados. Para probar que el hijo haya entendido bien los conceptos, el papá le pide que dibuje un cuadrado en el suelo en base a saltos. Construya un mundo que contenga al pequeño conejo, junto con el método de salto que creamos al principio de esta unidad. Utilice un bucle para que el animal salte tres veces, de tal manera que cuando termine de hacerlo, haga un giro de ¼ de revolución hacia la izquierda. Después, agregue otro lazo para repetir las acciones anteriores, como se muestra a continuación:

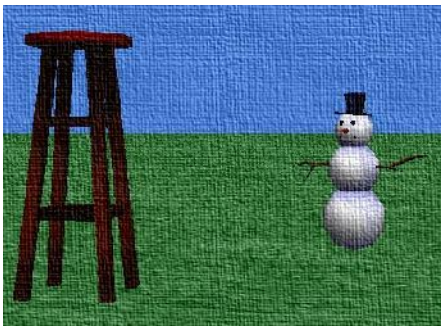
```
Iterar 4 veces
      Iterar 3 veces
            Hacer que el conejo salte
      Hacer que el conejo vire a la izquierda 0.25 revolución
```

3 Dragón curioso.

Cree una escena como la que se muestra a continuación, donde aparecen un dirigible y un dragón. La historia es que el dragón iba hacia el castillo, y en el camino se encontró con el dirigible. Los dragones son criaturas curiosas, y éste no ha sido la excepción, de tal modo que ha decidido volar y echarle un vistazo. El dragón decide volar y echar un vistazo alrededor el dirigible. Escriba un programa para que el dragón se mueva en dirección al dirigible, y que después gire a su alrededor durante tres veces, según la longitud del aparato. La animación debe incluir los métodos

- **moverHaciaElDirigible**
- **volarAlrededorDelDirigible**, junto a un bucle que repita este último método tres veces.

4 El Hombre de Nieve y el piso.



Este ejercicio es para practicar el uso de bucles con alguna pregunta numérica. Diseñe un mundo que contenga un muñeco de nieve y un piso grande (o taburete), como se observa en la imagen de la izquierda. Utilice un bucle para que el muñeco se mueva hacia el taburete, pero sin que choque o se

tropiece. La idea es que utilice el operador **distancia a** para determinar la cantidad de pasos que tendrá que hacer el muñeco para llegar al piso.

5 Cartel de neón.

Este antiguo salón está en proceso de convertirse en una atracción turística. Se pide que utilice un texto 3D para crear un cartel de neón que cuelgue de la parte delantera del balcón. A continuación, utilice un bucle para que el signo parpadee 10 veces. El contenido del mensaje corre por su cuenta.

Resumen

El bucle se ha introducido en esta parte de la unidad para permitir la repetición de un cierto bloque de instrucciones, constituyéndose en el segundo tipo de declaración que hemos estudiado. Nuestro primer programa consistió en usar un bucle para evitar escribir ocho veces el método del salto del conejo. La ventaja de esto se hizo sentir de inmediato, ya que corresponde a una estructura de programación que es rápida y fácil de escribir y de entender. Por supuesto, es posible escribir instrucciones que incluyan bucles que sean mucho más complicadas.

Conceptos importantes vistos hasta el momento

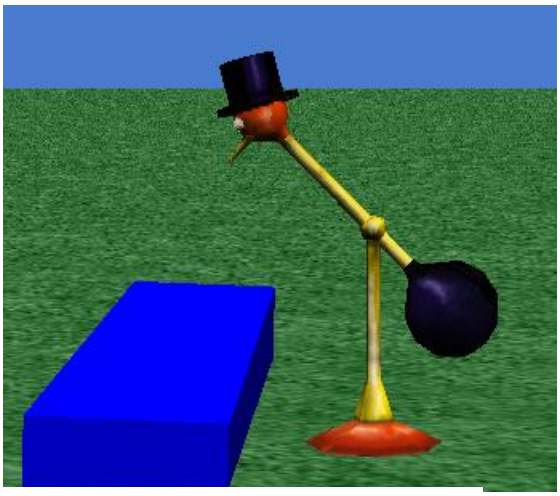
- Se puede usar un bucle para repetir bloques de instrucciones las veces que sea necesario.
- Un componente clave en un bucle es el **recuento**, o la cantidad de veces que deba repetirse.
- El recuento debe ser un número positivo o infinito. Si es igual a infinito, el bucle se repetirá hasta que el programa se cierra.

- Las instrucciones **Hacer en orden** o **Hacer juntos** se pueden anidar dentro de un bucle. También se pueden incluir bucles dentro de bucles, o hacer mezclas con sentencias condicionales.
- Cuando un bucle esta anidado dentro de otro bucle, tenemos uno interno y otro externo. La cantidad de veces que todo se ejecuta es igual al número de iteraciones del bucle externo multiplicada por la cantidad de iteraciones del interno. Por ejemplo, si el bucle exterior se desarrolla 5 veces, y el interior lo hace 10, entonces por cada iteración del externo, el de adentro lo hace 10.

Proyectos

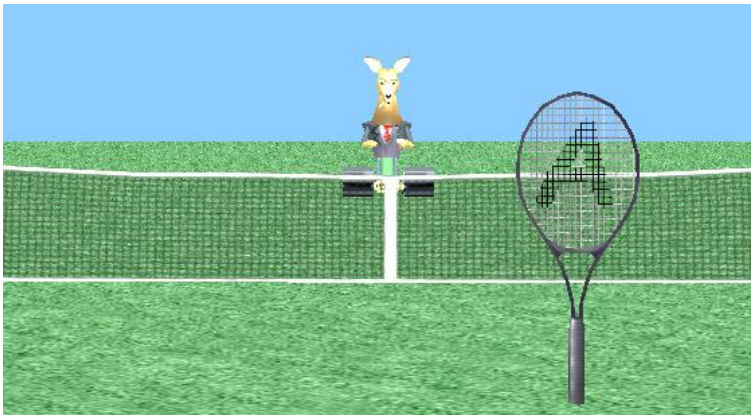
1 Loro que toma agua.

A Juan se le ha regalado un loro de juguete que puede tomar agua. En estos momentos, el aparato está frente a un recipiente de agua, y el cuerpo está impulsado al balde. Debido a la compensación de pesos en cada extremo de su cuerpo, el loro repetidamente baja su cabeza hacia el agua. Construya una animación que simule el funcionamiento de esta estructura, usando un bucle infinito para que el loro beba. La escena inicial se muestra a continuación.



2 Juego de Tenis.

Construya un juego de tenis, con un robot canguro, una pelota, raqueta y malla. Coloque la pelota de tenis inmediatamente delante del robot en el otro lado de la red (desde el punto de vista de la cámara), y la raqueta de tenis en este lado de la red (una vez más, desde el punto de vista de la cámara). La escena inicial debe ser parecida a lo que se muestra a continuación.



Establezca un evento en el controlador de eventos en el cual el usuario pueda mover la raqueta por medio del ratón. En este juego, el robot y la pelota de tenis deben moverse juntos a la derecha o a la izquierda, donde la distancia debe variar en forma aleatoria entre -1 y 1 metro.

A continuación, el robot «*lanza*» la bola a través de la red, y se debe mover hacia arriba una altura aleatoria durante un cierto periodo de tiempo que usted tiene que determinar, por medio de la técnica del ensayo y error. Cuando la pelota haya avanzado lo suficiente hacia la cámara, debe estar fuera del alcance de la vista, para simular que ha avanzado lejos.

El usuario moverá la raqueta de tenis para tratar de «*golpear*» la pelota, lo que se produce cuando la distancia que la separa de la pelota sea inferior a 0,1 metro. Se sabrá si el jugador consigue golpear a la pelota, porque el canguro deberá mover sus orejas, en el sentido que usted desee.

El verdadero desafío de este programa es determinar si el jugador realiza algún movimiento exitoso con la raqueta, para que ésta se acerque lo suficientemente cerca de la pelota, y logre golpearla antes que se le pierda de vista. Para realizar este trabajo, utilice un bucle, donde la ejecución de éste mueva la pelota hacia arriba y hacia adelante una distancia muy corta, del orden de 0,1 1 metro. Cada vez que se recorre el bucle, tiene que comprobar la cercanía entre ambos elementos, para saber si la pelota de tenis pudo ser golpeada. Cuando el bucle finaliza, tiene que hacer girar al robot canguro u de revolución a la izquierda, y mover la raqueta, señalando que el juego ha terminado.

3 El caballo y el toroide.

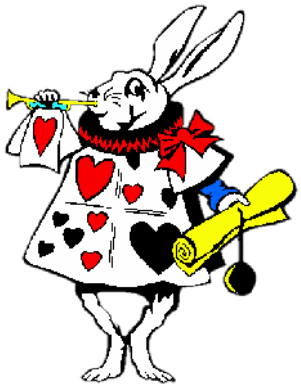
La escena de este ejercicio muestra a un caballo que está parado frente a un toroide, como parte de un espectáculo circense. El objetivo es que el animal deba dar un salto y pasar a través del toroide. Escriba un programa que emplee un bucle para que el caballo trote hacia adelante varias veces para que logre pasar a través del aro gigante. Utilice el método de prueba y error para determinar el número del bucle.

El proyecto debe incluir un método **trote** para hace que el caballo se desplace hacia adelante. En un desplazamiento, el caballo mueve la pata delantera hacia adelante, al igual que el resto de las extremidades. Para que la animación sea más realista, el animal tiene que

doblar las rodillas. Todo esto se tiene que hacer de forma simultánea mientras el caballo avanza. El bucle debe llamar al método trote un número determinado de veces.

4 Malabarismo.

Construya un mundo con un personaje a su elección, que tenga en su poder tres bolas para hacer malabarismo. Se pide que construya un método para que el artista pueda animar tres pelotas en el aire. Use un bucle para que el acto de repita cinco veces, y que luego las bolas caigan al suelo. Por supuesto, el personaje debe tener al menos dos brazos y ser capaz de moverlos para que el movimiento se vea más constante.



Tercera Parte

Bucle «*mientras*» o «*while*»

Las instrucciones tipo **mientras** se encuentran en la mayoría de los lenguajes de programación. En esta tercera y última parte de la unidad, veremos cómo podemos utilizarla como mecanismo de repetición. Al igual que con los lazos, usted descubrirá que este tipo de construcción es una potente herramienta para administrar segmentos de código que se tengan que ejecutar en reiteradas oportunidades.

El término **mientras** lo utilizamos a diario en nuestras conversaciones. Por ejemplo: «*mientras manejo, debo mantener las manos en el volante y la vista mirando al frente*», o «*mientras forme parte del equipo de fútbol, seré el arquero*». La palabra

mientras es un vocablo que cuando se dice, automáticamente uno como ser humano tiende a asumir que lo que se dice es **verdadero**. En el ejemplo del manejo, podemos asumir que **mientras sea verdadero el hecho que esté al volante**, **mis acciones serán** tener las manos sobre el volante y tener la vista al frente. Por lo tanto, podríamos pensar que si no estuviera manejando, ninguna de las actividades mencionadas con anterioridad se debiera hacer.

3.5 While (mientras)

La sentencia while es un **bucle condicional**, que hace uso de **operadores booleanos** al igual que las estructuras de programación **if/else**. La condición lógica que se ocupa actúa como si fuera una agencia de aduanas, en el sentido que si la condición de entrada se cumple, entonces esta «*aduana*» permite el paso hacia el interior del **while**. Por supuesto que en caso contrario, no podremos ingresar. Ahora bien, si estamos «*dentro*» de la ejecución de un bucle **while**, la condición de entrada se comprueba cada vez que se ejecutan las instrucciones que contenga. Por lo tanto: mientras la condición se cumpla, hay que repetir las instrucciones, de tal manera que cuando sea **falsa**, el bucle **termina** y Alice continúa con la ejecución del resto del programa. Revisemos esta importante instrucción a través de un ejemplo.

Ejemplo No. 1: Persecución en el agua

Esta animación simulará una persecución, que es bastante común en videojuegos o en películas animadas. En este mundo, un pez grande tiene hambre, y decide saciar su apetito a expensas de uno más pequeño, que se encuentra nadando en forma aleatoria de la manera más plácida posible. Nuestra tarea va a consistir en animar al pez grande para que persiga al pequeño (de color rojo), se acerque lo máximo posible — menos de un metro —, y se lo trague. Antes de construir el programa, es necesario que pensemos en los detalles de programación que aparecen involucrados. El pez de colores debe moverse al mismo tiempo que lo hace el pez grande. Supongamos que el pez grande siempre se mueve a una distancia constante. Uno de los problemas es que el pez de color rojo nade de forma aleatoria, lo que significa que no podemos saber cuántas veces se pueden repetir las acciones de nado de ambos especímenes.

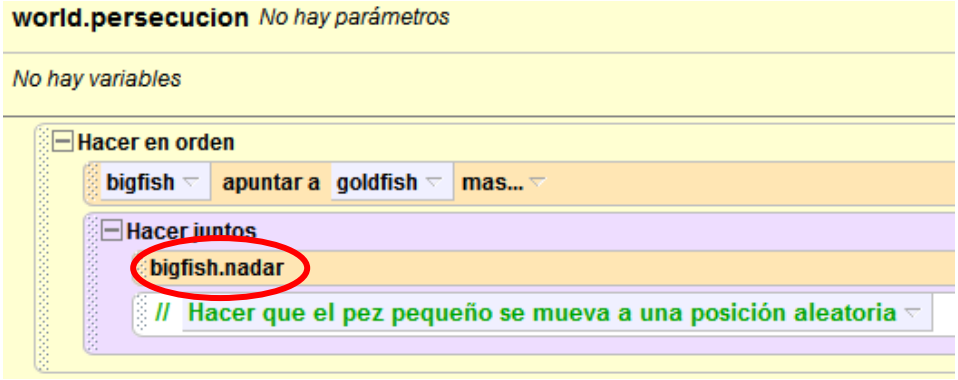
Si el pez de colores se mueve una distancia larga, el pez grande tardará una mayor cantidad de repeticiones en lograrlo. Por otra parte, si la distancia que recorre el pez pequeño es corta, entonces el pez grande demorará menos en capturarlo. En este problema resulta evidente que el número de repeticiones es desconocido. Otro problema que emerge es cómo lograr que el pez se mueva en forma aleatoria. Veamos cómo podría ser un guión textual:

Tabla 16. Simulación de una persecución.

```
Hacer en orden:
      1. El pez grande debe apuntar al pez pequeño.

      Hacer al mismo tiempo:
            1. Mover el pez grande hacia el pez pequeño a una
            distancia constante.
            2. Mover de forma aleatoria el pez pequeño hacia una
            nueva posición.
```

La diferencia entre los movimientos de los peces, es que el grande se mueve hacia el pequeño a una distancia conocida, pero el de colores se mueve a una nueva posición al azar. Pero esto es una persecución, lo que significa que las acciones anteriores deben repetirse hasta que el pez grande se acerque lo suficiente al pez pequeño para tragarlo, y poner fin a la animación. Pero, antes de que nos preocupemos de repetir las instrucciones, implementemos este conjunto de instrucciones como un método de mundo que le vamos a llamar **persecución**, que obedece a lo que escribimos en el guión textual:



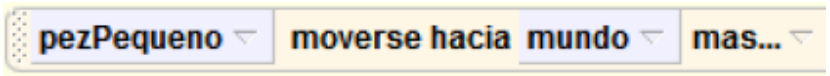
En la Figura 86 se muestra el método de nado que se ha construido para el pez grande, que por el momento no tiene mayor importancia, ya que el foco de la atención va a estar centrado en el movimiento del pez pequeño.

Figura 86. Estilo de nado del pez grande.

Cómo simular el movimiento aleatorio

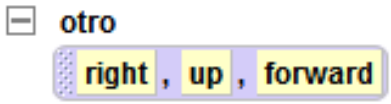
El verdadero reto es conseguir que el pez de colores tenga una ubicación aleatoria. La razón de por qué esto es un reto se debe a que siempre hemos trabajado con movimientos **fijos**, como por ejemplo: adelante, atrás, arriba, abajo izquierda o derecha una cantidad específica de metros. A lo más, en algunos ejercicios y proyectos se ha pedido que usted investigue sobre la función de mundo que genera números aleatorios, pero no posiciones. Pero, cada uno de los objetos de Alice poseen método llamado **mover hacia**, que se puede usar para mover a alguna ubicación específica, como vimos en el ejemplo del entrenador de basquetbol que debía encestar la pelota en el aro. Sin embargo, en este ejemplo, el pez debe moverse a una posición aleatoria que sea relativamente cerca de su ubicación actual.

Podemos utilizar la instrucción **moverse hacia** con movimiento aleatorio siguiendo unos sencillos pasos. En primer lugar, arrastramos la instrucción hacia el editor de código, y elegimos cualquiera de las opciones disponibles. Para efectos de este ejemplo, se empleó el mundo:

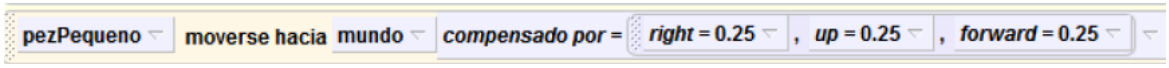


Después, apretamos en **más**, y seleccionamos la opción **Vector 3(0,0,0)** que viene dada en **position**:

A continuación, vamos a las **funciones del mundo**, vamos al grupo de preguntas que se llama **otro**, marcamos lo que diga **right, up, forward**, lo arrastramos hacia el editor y soltamos el puntero del ratón sobre **Vector 3(0,0,0)**.



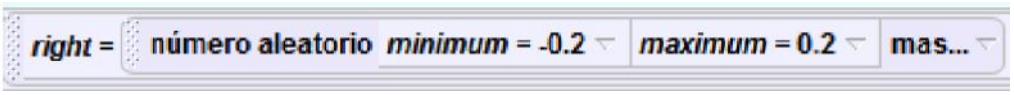
Luego, nos va a pedir que ingresemos valores para **right**, **up** y **forward**: de momento, seleccione los números que quiera. Hasta el momento — salvo diferencias numéricas — tendríamos esto:



Por supuesto que no queremos que el pez se muevan a la ubicación fija (0.25, 0.25, 0.25), sino que a una aleatoria. Para esto, fijémonos que tanto los nuevos valores son números, y como tales, pueden ser reemplazados por otros. Nuevamente, vamos a las funciones del mundo, escogemos la pregunta **número aleatorio** y la arrastramos hacia el código. De forma predeterminada, se genera un número al azar entre 0 y 1, pero tenemos la posibilidad de modificarlos para permitir un intervalo personalizado.

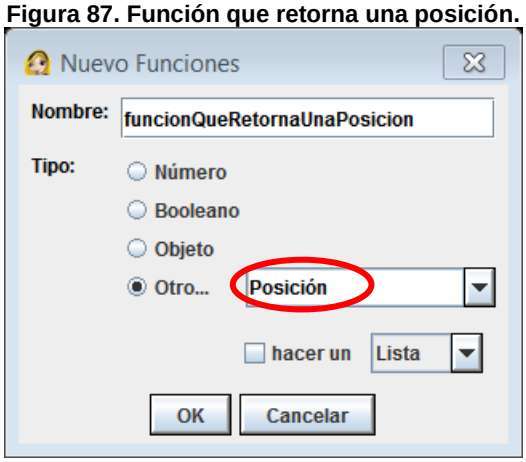
Sin embargo, lo que queremos es limitar el movimiento a un lugar cercano, para que la animación sea más real. Para esto, podemos permitir que el radio de movimientos varíe entre -0,2 y 0,2, para generar la sensación de cercanía. Esto se

logra alterando los valores de **mínimo** y de **máximo** que proveen las opciones adicionales de la generación de números aleatorios. En la figura xx se muestra la alteración del parámetro **right** en función del rango deseado. El mismo proceso se sigue para **up** y **forward**.



Recién podemos decir que el pez de color se ha movido de manera aleatoria de su posición actual. Se creó además una pregunta de mundo llamada **nuevaPosicion**, que nos va a retornar una determinada ubicación dentro del espacio para el pez pequeño. Como se había comentado con anterioridad, la posición de un objeto dentro del mundo se determina en base a **tres coordenadas**, que Alice le llama **right, up y forward**. En la Figura 87 se muestra la declaración de una función que va a **retornar una posición**, que es algo que hasta el momento no habíamos hecho:

Figura 87. Función que retorna una posición.



En el algoritmo que se presenta a continuación, vamos a trabajar con un número aleatorio que varíe entre -0.2 y 0.2, que nos permita lograr el efecto que el pez pequeño se desplace hasta una posición cercana de la que se encuentre actualmente:

Figura 88. Reajuste de la posición de un objeto en el espacio, por medio de números aleatorios.

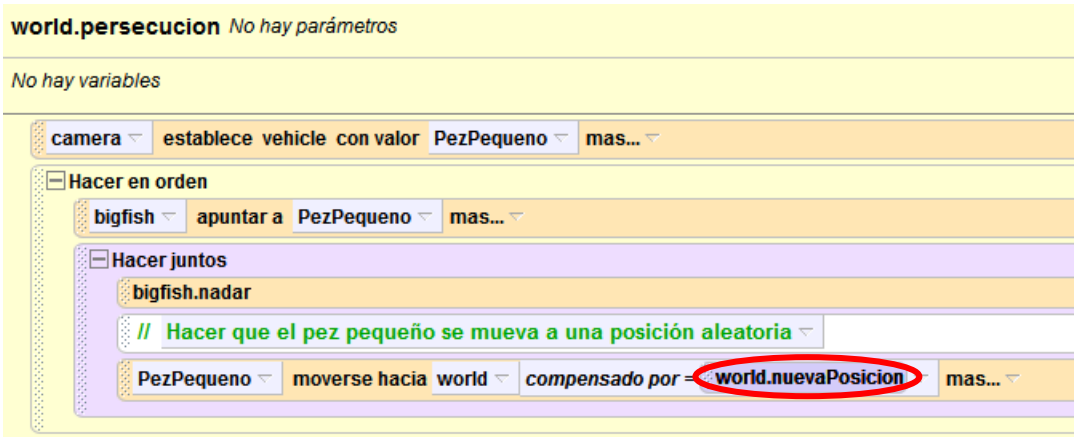
world.nuevaPosicion ( )
No hay variables

No hacer nada

Retornar ((**(** (**distancia a la derecha de** (**la posición del PezPequeno mas...**)) + (**número aleatorio** minimum = **-0.2** maximum = **0.2 mas...**) **)**) **,** (**(** (**distancia de** (**la posición del PezPequeno mas...**) **hasta**) + (**número aleatorio** minimum = **-0.2** maximum = **0.2 mas...**) **)**) **,** (**(** (**distancia en frente de** (**la posición del PezPequeno mas...**)) + (**número aleatorio** minimum = **-0.2** maximum = **0.2 mas...**) **)**))

El resultado del método **persecución** queda cómo se muestra a continuación:

Figura 89. Ejemplo de método que incorpora una llamada a una función que devuelve una posición en el espacio.



Si usted ejecuta este código, notará que sólo se ejecuta una vez, y que no se ha producido una comprobación que permita determinar si acaso será necesario correrlo de nuevo, en caso que el pez grande esté a más de un metro de distancia del pequeño. Veamos ahora cómo lo haremos.

Usando la instrucción while

Vamos a enriquecer el método **persecución** para permitir que en algún momento esto acabe. Piense en esta situación: **mientras el pez grande esté a más de un metro de distancia del pez pequeño**, haga que el primero apunte en dirección al segundo, y que después en forma simultánea el grande empiece a nadar y el pequeño se mueva a una posición aleatoria.

Entonces, nuestra condición lógica apunta a responder la pregunta booleana si acaso **la distancia que separa al pez grande del pequeño es mayor a un metro**. Si esta condición se cumple, el método **persecución** se ejecuta. Un posible guión textual se muestra a continuación en el algoritmo que se describe en la Tabla 17:

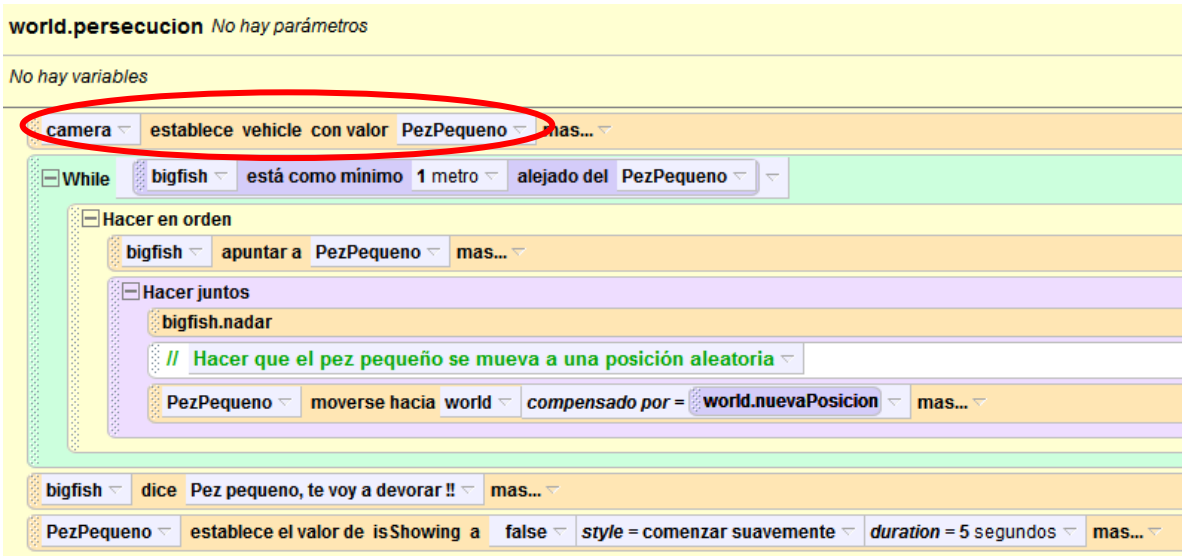
Tabla 17. Persecución a través de un bucle «*while*».

```
Mientras el pez grande esté a más de un metro de distancia del pez
pequeño
      Hacer en orden:
            1. El pez grande debe apuntar al pez pequeño.
            Hacer al mismo tiempo:
                  1. Mover el pez grande hacia el pez pequeño a una
                  distancia constante.
                  2. Mover de forma aleatoria el pez pequeño hacia
                  una nueva posición.
El pez grande devora al pez pequeño.
```

Se reitera la idea principal: si la condición de entrada se cumple, entramos en el ciclo del **while**, y una vez que se haya computado las instrucciones paralelas «*Mover el pez grande hacia el pez pequeño a una distancia constante*» y «*Mover de*

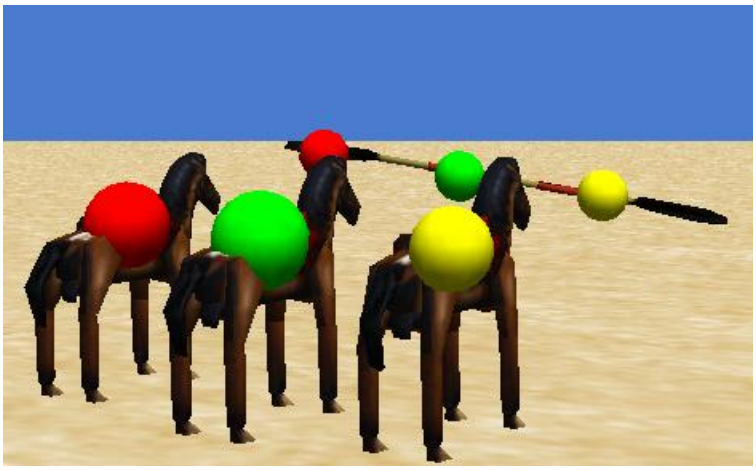
forma aleatoria el pez pequeño hacia una nueva posición», nuevamente hay que revalidar la pregunta inicial, para saber si continúa o no la ejecución de **while**. De esta manera, el método de **persecución** se ilustra en la Figura 90.

Figura 90. Implementación del algoritmo de persecución.



En el código anterior, se ha subrayado el hecho que el vehículo de la cámara pase a ser el pez pequeño. Si esa opción no estuviera disponible, entonces en más de una ocasión este objeto estaría fuera de nuestro alcance visual.

Ejemplo No. 2: Carrera de caballos



Este ejemplo es la versión de una carrera de caballos en un carnaval. En la escena inicial que se muestra en la imagen adjunta, se ha puesto un remo de kayak para representar la línea de meta. Por supuesto, usted puede emplear otros elementos para simularla. En la línea de meta se han puesto tres pelotas de colores para determinar el punto exacto al que tenga que llegar cada uno de los caballos al final de la carrera. A diferencia de las competencias típicas donde los animales tienen que correr en torno a una pista ovalada, en este carnaval los caballos se mueven hacia adelante a través de una vía mecánica.

Además, cada ejemplar tiene un color específico para su montura, de tal manera que el caballo con cinturón rojo tiene que llegar al marcador rojo, el de montura azul debe

alcanzar la pelota azul. Para el caballo con color amarillo es lo mismo. Los organizadores han dispuesto por lo menos dos opciones para ejecutar la carrera:

1. Todos los caballos corren de manera continua, en forma aleatoria hasta que uno de ellos llegue a la meta. **Esta modalidad quedará como ejercicio.**
2. Mover los caballos hacia adelante en función del valor que arroje un número aleatorio. En este caso, el juego termina cuando uno de los animales haya llegado hasta su marcador asociado.

Lo que haremos en esta parte es mostrar cómo se puede implementar la segunda alternativa. Para esto, usaremos un bucle **while** para codificar las distintas acciones que puedan tomar lugar durante la competencia. Antes de hacerlo, debemos tener claro cómo vamos a subsanar dos de los problemas de programación que se puedan presentar:

1. Cómo determinar cuándo termina la carrera.
2. Cómo **elegir al azar** uno de los caballos para simular el avance de cada uno de ellos.

Problema No. 1: Cómo determinar cuando la carrera ha terminado

La carrera termina cuando uno de los caballos se acerca lo suficiente a su marcador en la línea de meta para ser declarado ganador. Alice ofrece una serie de opciones posibles para averiguar cómo hacer que un caballo coincida con su destino o bola de color. En este ejemplo, vamos a usar la función

```
está como mínimo threshold alejado del objeto
```

Que está disponible dentro de la pestaña de preguntas de cualquier objeto que no sea el mundo. Entenderemos en esta parte que el **umbral** se refiere a la **mínima distancia que pueda existir entre el caballo y su marcador de destino**.

La escritura de una condición de una sentencia booleana en un bucle **while** se hace de la misma manera que se emplea para las construcciones **if/else**. Recuerde que **una condición es necesaria para saber si el bucle se sigue ejecutando o no**. En este ejemplo, no basta con decir que «*El objeto Caballo1 debe estar por lo menos a 1.1 metros de la meta*», porque tenemos un total de tres animales. Por lo tanto, **mientras el caballo1 esté a 1.1 metros de la meta Y el caballo2 esté a 1.1 metros de la meta Y el caballo3 esté a 1.1 metros de la meta**, la carrera **continúa**. En este caso, basta con que una de las tres condiciones **no se cumpla**, para que termine el bucle.

¿Cuáles son las tres condiciones?

Condición No. 1. El caballo1 está a 1.1 metros de la meta.
Condición No. 2. El caballo2 está a 1.1 metros de la meta.
Condición No. 2. El caballo3 está a 1.1 metros de la meta.

Al igual como lo hicimos en el ejemplo del mundo de Zeus, ahora tenemos que trabajar con tres condiciones **Y** en la formulación de la condición lógica. La Figura 91 muestra cómo sería el esqueleto del **while**:

Figura 91. Condición de término del bucle «*while*»para la carrera de caballos.

```
While ( ambos ( horse está como mínimo 1.1 metros alejado del destino1 ) y ( ambos ( horse2 está como mínimo 1.1 metros alejado del destino2 ) y ( horse3 está como mínimo 1.1 metros alejado del destino3 ) ) )
```

Problema No. 2: Cómo elegir al azar uno de los caballos para simular el avance de cada uno de ellos

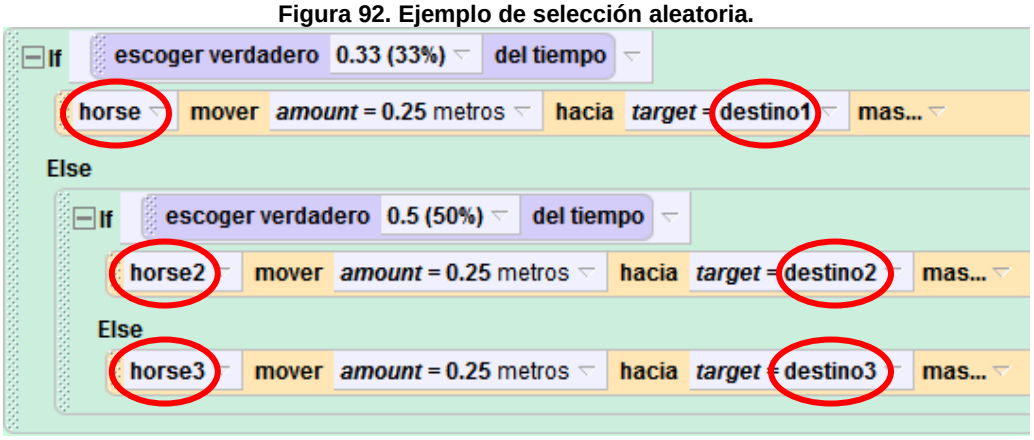
Ahora podemos abordar la tarea de decidir cuál será el caballo que se mueva. De acuerdo al enunciado del problema, debemos permitir que haya una selección aleatoria del animal que se vaya a desplazar. En el ejemplo de la persecución, habíamos trabajado con números generados al azar para generar una nueva coordenada para simular el movimiento del pez pequeño. La diferencia es que ahora tenemos que seleccionar aleatoriamente un caballo.

Además, pensando en términos estrictamente deportivos, sería bueno que cada animal tuviera **la misma posibilidad** de ser **elegido** que el resto. En términos matemáticos, buscamos que **la probabilidad de selección de cualquiera de los caballos sea la misma**. En Alice se puede trabajar con la función denominada

`escoger verdadero` `probabilityOfTrue` **`del tiempo`**

Tiene un nombre extraño, pero lo que hace es seleccionar «*algo*» en base a la probabilidad que algún objeto tenga para ser escogido. En este caso, tenemos tres caballos, y por lo tanto cada uno debe tener 1/3 de probabilidad de ser elegido, de tal manera que los 2/3 del tiempo restante no se mueve. Observe este código:

Figura 92. Ejemplo de selección aleatoria.



Lo que nos dice es que el 0.33 (≈ 1/3) del tiempo, el caballo1 va a ser el seleccionado. Pero si no es así, entonces nos quedan dos alternativas: caballo2 ó caballo3. Sin embargo, en este caso no tenemos 3 posibilidades, sino que 2: por lo tanto, volvemos a aplicar la función **escoger verdadero** probabilityOfTrue **del tiempo**, pero ahora con 50% de posibilidad para cada uno. También es fácil advertir que se ha ocupado la función **`mover`** `cantidad` **`hacia`** `objetivo`. ¿Por qué se trabajó con esa y no simplemente con **mover** hacia adelante?.

Ahora que hemos descubierto la manera de escribir la condición de tipo Boolean para la instrucción while y cómo elegir al azar uno de los caballos que avancen en cada iteración, podemos escribir el código final del método **carreraDeCaballos**, que se muestra en la Figura 93.

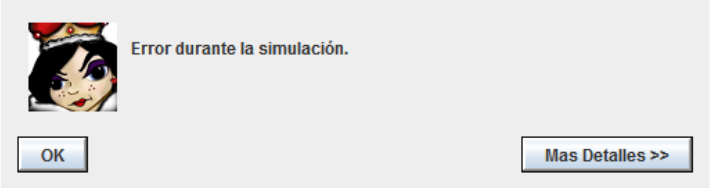
Figura 93. Implementación de la carrera de caballos.

```
world.carreraDeCaballos ( )
No hay variables
    While ( ambos ( horse está como mínimo 1.1 metros alejado del destino1 ) y ( ambos ( horse2 está como mínimo 1.1 metros alejado del destino2 ) y ( horse3 está como mínimo 1.1 metros alejado del destino3 ) ) )
        If ( escoger verdadero 0.33 (33%) del tiempo )
            horse mover amount = 0.25 metros hacia target = destino1 mas...
        Else
            If ( escoger verdadero 0.5 (50%) del tiempo )
                horse2 mover amount = 0.25 metros hacia target = destino2 mas...
            Else
                horse3 mover amount = 0.25 metros hacia target = destino3 mas...
    // El caballo vencedor se moverá hacia arriba un metro, como una forma de identificarlo
    If ( horse está dentro del rango 1.1 metros del destino1 )
        horse mover arriba 1 metro mas...
    Else
        If ( horse2 está dentro del rango 1.1 metros del destino2 )
            horse2 mover arriba 1 metro mas...
        Else
            horse3 mover arriba 1 metro mas...
```

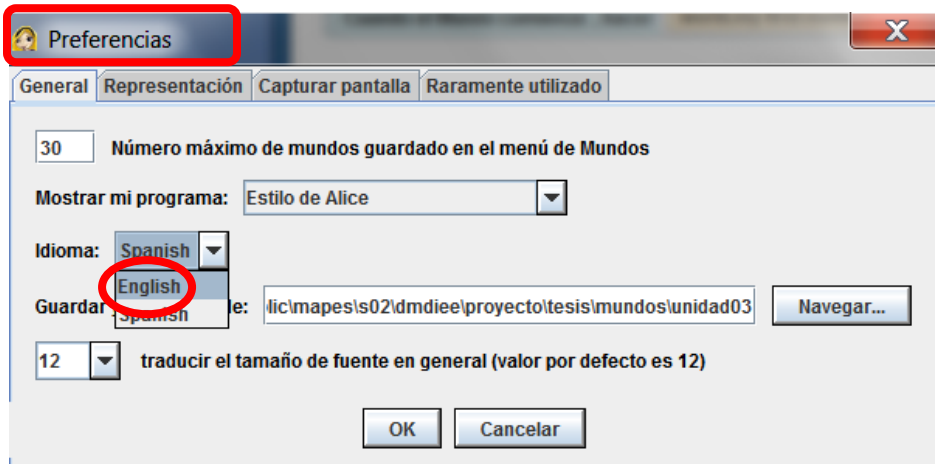
En este algoritmo, el caballo ganador se desplaza automáticamente 0.5 metro hacia adelante, que nos sirve para tener una mejor perspectiva del vencedor. Debido a la selección al azar de este programa, diferentes caballos debe ganar la carrera si se ejecuta el programa varias veces.

Nota técnica

Es posible que este mensaje aparezca al momento de intentar ejecutar el código anterior:



Una forma de solucionarlo, es **cambiando el idioma de Alice**. Para esto, vamos a **Modificar->Preferencias**, y en la primera pestaña seleccionamos la opción **inglés**:



Ejercicios

1 Carrera de caballos.

Modifique el mundo de las carreras de caballos vista en el ejemplo anterior, de tal manera que incorpore estos elementos:

1. Los caballos deben moverse en una animación más realista, en lugar de deslizarse a lo largo del suelo.
2. Los 3 caballos deben avanzar simultáneamente hacia adelante, pero desplazándose a una distancia generada al azar entre uno y otro.

2 El robot y el robot.

Cree un mundo con un robot gorila y una moto, como se ilustra a continuación. Escriba un programa interactivo que le permita al usuario arrastrar la moto a través de la escena, de tal manera que en todo momento el robot la persiga sin que colisione con ella. Trabaje con diferentes criterios de término del programa.

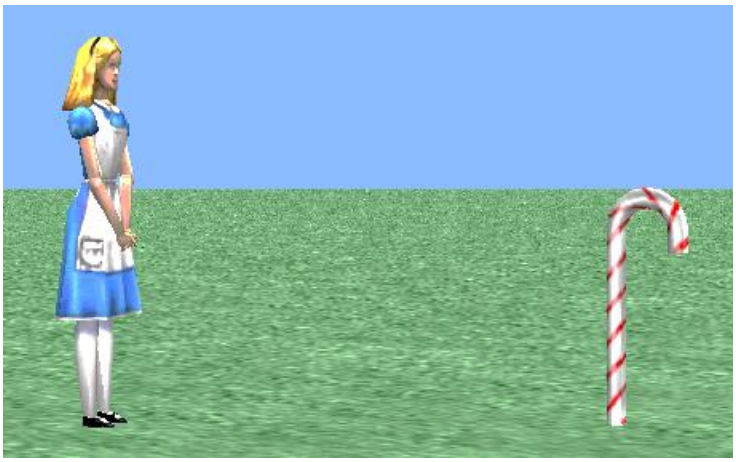
3 El conejo y la mariposa.

Considere la escena inicial que se muestra en la imagen de la derecha. En ella, un conejo blanco está persiguiendo una mariposa. El mundo que debe crear es similar a la escena del texto, excepto en dos aspectos importantes:

1. El conejo blanco siempre debe permanecer en el suelo, porque no vuela.
2. Con el fin de evitar que la mariposa vuele muy alto o muy bajo, debe coordinar que la mariposa se desplace de arriba hacia abajo un valor aleatorio entre 0 y 1, en lugar del valor actual que tiene por defecto.

La idea es que cuando el conejo se acerque lo suficiente a la mariposa, la capture con su red. Determine usted los criterios de acercamiento que le parezcan más apropiados.

4 El toque del Rey Midas. Parte I.



Un mago le ha hecho un conjuro a Alice, que consiste en que todo lo que ella toque, se convierta en oro. Alice está frente a un bastón dulce, como el que se muestra en la escena inicial. Escriba un método recursivo, denominado **comprobarBaston** que permita decidir si acaso la mano derecha de Alice está muy cerca del bastón dulce, en un alcance de 1 metro. Si es así, Alice se inclina para tocarlo, para que luego se convierta en oro. Si no, ella se deberá mover a una distancia pequeña de 0.1 metro, e invocar nuevamente al método **comprobarBaston**.

5 El toque del Rey Midas. Parte II.

Cree una segunda versión para el mundo del Rey Midas, en la cual el usuario pueda guiar el movimiento de Alice hacia el bastón dulce. Para hacer esto un poco más desafiante, Alice no debe estar mirando hacia el bastón dulce en la escena inicial, lo que significa que el método **comprobarBaston** se pueda alterar para que cuando ella esté más cerca del caramelo, pueda girar para ver si se puede agachar para tocarlo. Para esto, use las teclas de flecha izquierda y derecha para mover a Alice en esos sentidos. Construya los métodos **doblarDerecha** y **doblarIzquierda**, que deben provocar giros de 0.05 metros hacia la derecha o izquierda, respectivamente, que se deben activar con las teclas de flechas de dirección mencionadas anteriormente.

3.6 Repetición infinita

Como habíamos comentado en la segunda parte de esta unidad, podemos usar bucles que tengan una infinita cantidad de repeticiones, y que terminen cuando el usuario decida terminar la ejecución del programa. Estas técnicas son útiles cuando tenemos animaciones que siempre tienen que estar ocurriendo en un mundo. Por ejemplo, podemos tener un parque de entretenciones donde el carrusel deba estar girando de manera permanente mientras el usuario interactúa con otros objetos del escenario.

Métodos mutuamente invocados

La primera técnica que se presenta corresponde a un concepto bastante sofisticado en programación, donde los métodos se invocan mutuamente entre sí**.** Supongamos que tenemos un mundo virtual que contiene un helicóptero en la escena inicial que se exhibe a la derecha. La idea de esta animación es permitir que de manera alternada las aspas de la hélice del helicóptero giren en sentido horario y anti horario. Las figuras adjuntas nos muestran por separado la implementación de los métodos **aspas SentidoHorario** y **aspas SentidoAntihorario**, respectivamente.

Figura 94. Giro de las aspas en sentido horario.

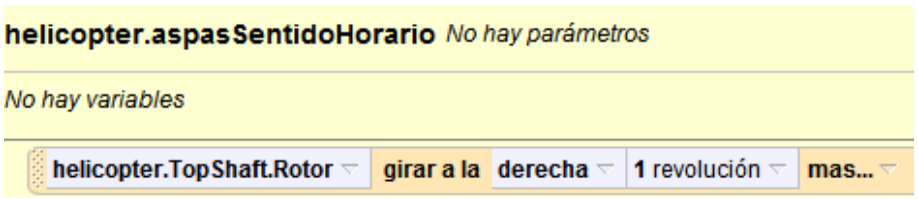
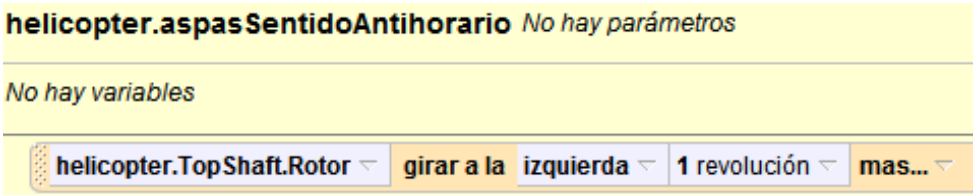
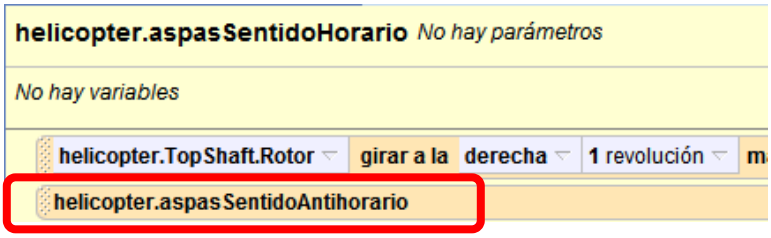


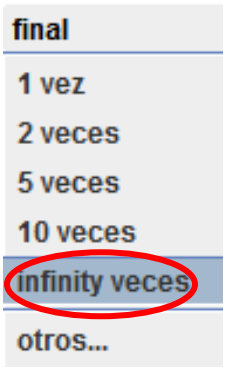
Figura 95. Giro de las aspas en sentido antihorario.



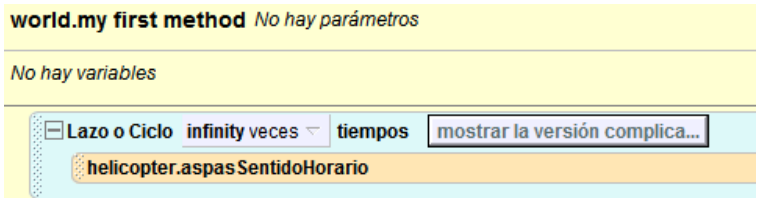
Por lo tanto, ¿cómo podemos hacer que estos dos métodos se repitan una y otra vez, mientras el mundo esté en marcha? De hecho, hay varias maneras de hacer que esto ocurra. En este ejemplo, la técnica que se va a utilizar es que el método que permite girar a la derecha invoque al que hace girar las aspas hacia la izquierda, como se muestra a continuación:



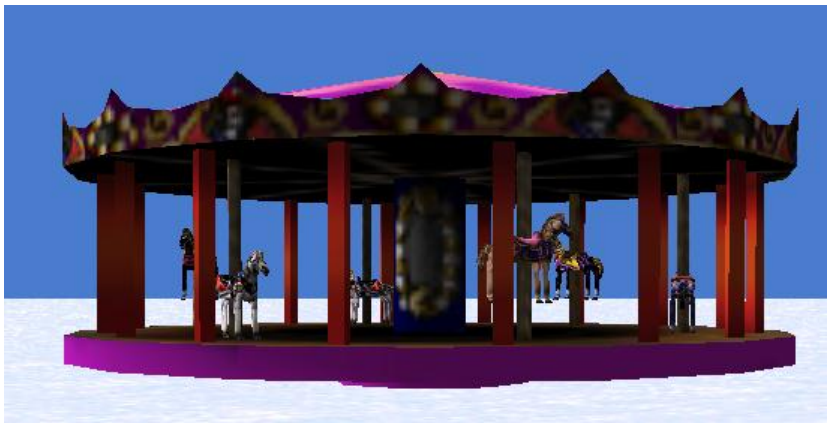
Entonces, para hacer que esto ocurra indefinidamente, basta con que arrastremos la instrucción **Lazo** hasta el editor, y elegimos la opción **infinidad de veces**, que aparece en el cuadro emergente:



Y por lo tanto, podemos tener algo como esto:



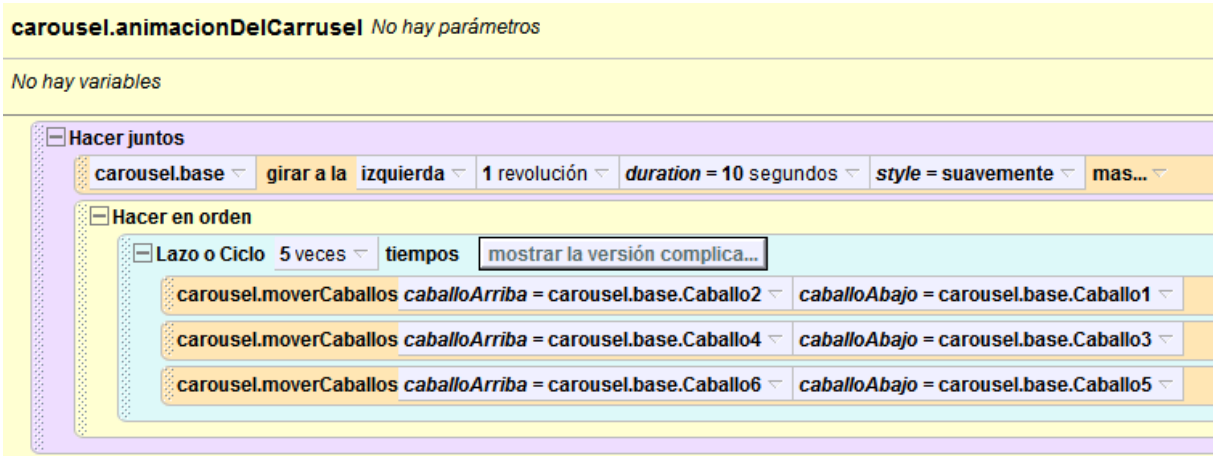
Infinitos bucles



La imagen adjunta muestra la escena de un carrusel que contiene caballos en su interior. Como usted imaginará, esta es una de las tantas atracciones que puede tener un parque de diversiones Siéntase libre de agregar todos los objetos que estime necesario para que la escena se vea más completa. Por ahora, sólo nos vamos a concentrar en el carrusel que tiene que dar «*muchas*» vueltas al ritmo de la música que suena de fondo. Además, los caballos repiten el ciclo de movimientos arriba – abajo – arriba en forma indefinida.

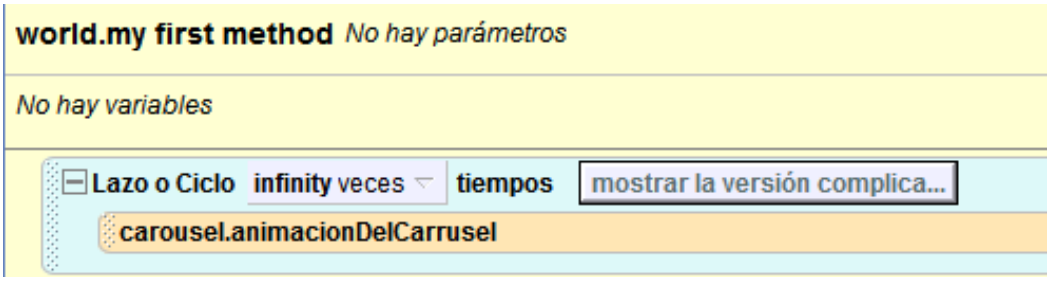
Las instrucciones para mover el carrusel van incluidas en un bloque **Hacer juntos**, y dentro encontramos las llamadas al método **moverCaballos**, que toma de a dos ejemplares y va alternando los movimientos para lograr el efecto cíclico arriba – abajo – arriba. Como se ha comentado en otras oportunidades, **cada programador es libre de recrear una animación de muchas maneras**. Por lo tanto, usted puede incluso escribir otro método que simule el funcionamiento de los caballos, en lugar del que se presenta acá:

Un posible código para animar al carrusel es el que se muestra a continuación:



Usted notará que los códigos exhibidos permiten hacer **un solo movimiento del carrusel y de sus componentes internas**. Para lograr que el aparato gire para siempre debemos emplear un **bucle infinito**, como se muestra en el método **carruselGirandoInfinitasVeces** que se muestra en la Figura 96:

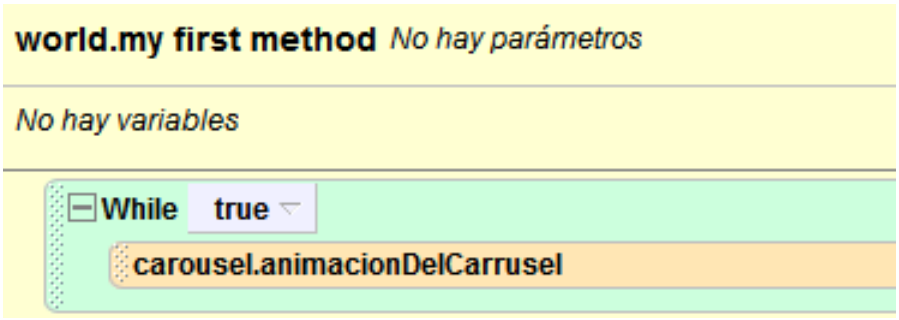
Figura 96. Un carrusel que funciona indefinidamente.



La técnica de while(true) también recrea infinitas repeticiones

Otra técnica que emplean los programadores para recrear infinitas repeticiones es por medio del uso de una sentencia condicional **true** dentro de la condición lógica del **while**. La Figura 97 muestra el método modificado. Lo que ocurre acá es que **mientras la condición del bucle while sea verdadera**, la animación se va a ejecutar. Sin embargo, dentro del código del giro del carrusel no hay sentencias o comandos que hagan que en algún momento la condición pudiera ser falsa. Por lo tanto, la acción del bucle while se mantendrá mientras la animación está en marcha.

Figura 97. Una forma equivalente de hacer una repetición infinita.



Evento «*begin-during-end*»

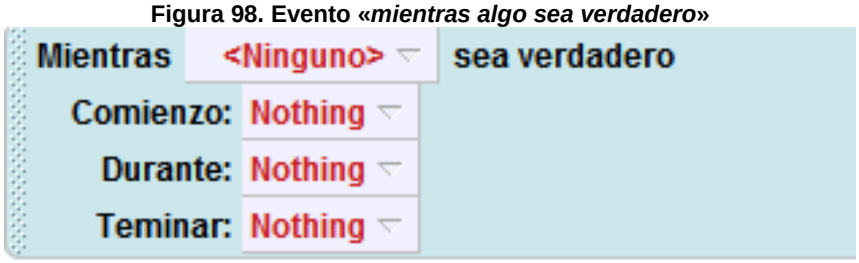
Hasta el momento, hemos visto que las acciones dentro de un bucle **while** se desarrollan, mientras la condición sea verdadera. Sin embargo en algunas oportunidades nos gustaría lograr que una acción pudiera ocurrir cuando alguna condición sea falsa, y el bucle termine. Por ejemplo, consideremos la escena que se muestra en la imagen adjunta, donde un helicóptero ha llegado a una isla con el objetivo de rescatar al conejo. El helicóptero está permanentemente dando vueltas en círculos alrededor de la isla. Queremos que el conejo esté siempre observando al helicóptero, y que siempre sus ojos estén apuntando a la máquina, a medida que pasa cerca de él. Por supuesto, que en el algún momento el helicóptero va a estar fuera del alcance visual del conejo. Cuando esto ocurra, haremos que el conejo quede mirando a la cámara. Vamos a considerar dos tipos de acciones repetitivas:

1. El helicóptero dando vueltas alrededor de la isla.
2. El conejo observando al helicóptero cuando éste de vueltas en círculos.

También, una acción debiera ocurrir cuando el helicóptero esté fuera del alcance visual del conejo, tras lo cual el animal se volteará y mirará hacia la cámara. El problema

es cómo hacer que las acciones se repitan con un bucle, y permitir que otras ocurran cuando una iteración haya terminado. Para esto, vamos a trabajar con un **evento** que permite trabajar con acciones repetitivas, que se llama **mientras algo sea verdadero**, como se muestra a continuación:

Figura 98. Evento «*mientras algo sea verdadero*»

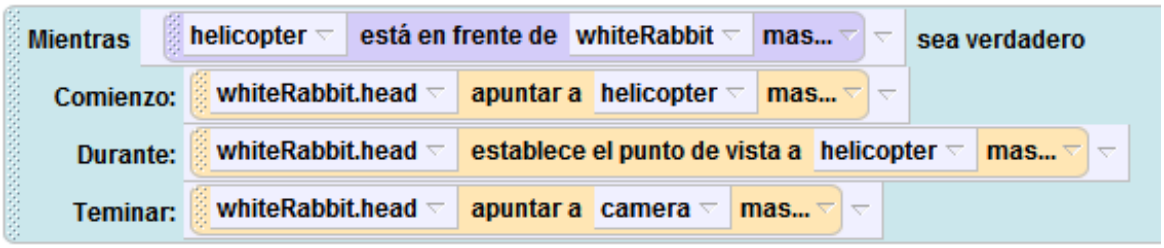


Cuando esto aparece, podemos especificar el flujo de acciones que se tienen que dar cuando el bucle empieza, durante su ejecución, y cuando termina. Hasta el momento sólo hemos usado la parte del **durante**. Veamos en qué consiste cada una de ellas:

- **Expresión condicional.** Por defecto, aparece la palabra **<Ninguna>**, pero la podemos reemplazar por alguna sentencia booleana, o por una función que devuelva verdadero o falso.
- **Comienzo.** Corresponde a la animación o al método que se tiene que ejecutar cuando la **expresión condicional** sea verdadera.
- **Durante.** Es una animación o método que se tiene que hacer en reiteradas oportunidades mientras la **expresión condicional** sea verdadera. Es importante destacar que al primer instante en que la condición sea falsa, terminará la ejecución del método, incluso si está en la mitad de una instrucción.
- **Terminar.** Es una animación o método que se tiene que ejecutar una sola vez, cuando la condición sea **falsa**.

En el ejemplo anterior, el **inicio** estará dado por el hecho que la cabeza del conejo esté mirando al helicóptero cuando éste se encuentre frente suyo. El **durante** es cuando el helicóptero permanezca estando al frente del conejo, mientras que el **final** se produce en el momento que el conejo pierde de vista al helicóptero y tiene que mirar a la cámara. El código de la Figura 99 ilustra esto:

Figura 99. Ejemplo de implementación de un evento «*mientras algo sea verdadero*».



La instrucción **establecer el punto de vista a** permite que un objeto ajuste su punto de vista hacia otro. En este ejemplo, se utilizó para lograr que la cabeza del conejo se mueva de manera continua hacia el helicóptero, mientras la condición del bucle **while** sea verdadera.

Importante

El uso de múltiples bloques **hacer juntos** dentro de una estructura **inicio, desarrollo y término** puede causar errores en tiempo de ejecución, si acaso esta estructura termina y quedara alguna de las instrucciones de **hacer juntos** en estado de ejecución.

Ejercicios

1 Paso a paso.

Cree un conjunto de movimientos reales para cualquier personaje que esté dentro de la galería personas. La idea es que emplee métodos que se llamen mutuamente para sincronizar correctamente los pasos hacia la izquierda, derecha, adelante y atrás.

2 Continuar y parar.

Elija algún objeto de la galería de atracciones que sea diferente del carrusel, y que se caracterice por tener un movimiento circular, como el que tiene la noria. Construya un método que permita moverlo de manera infinita, pero que pueda ser posible activarlo y desactivarlo a través del evento **mientras que algo sea verdadero**. Agregue un interruptor en el mundo que permita iniciar o detener la marcha del objeto.

3 Controlador de potencia para un ventilador.

Coloque un ventilador en mundo nuevo. La gracia que tiene este aparato es que cuenta con cuatro botones de potencia: alta, mediana, baja y apagado. Construya uno o varios métodos — según lo que estime conveniente — que permitan controlar la velocidad a la que las aspas del ventilador giren, en función del botón que se haya seleccionado. El ventilador debe proseguir la marcha hasta que la animación deje de funcionar, o hasta que el usuario haga clic en el botón de apagado.

4 Fonógrafo.

Cree un mundo que contenga un fonógrafo. El aparato debe tener definidos algunos métodos que permitan mover la manivela y hacer girar el disco. A continuación, cree un evento del tipo **mientras true es verdadero** que llame a un método que permita mover la manivela y hacer girar el disco al mismo tiempo.

5 Pingüino a cuerda.

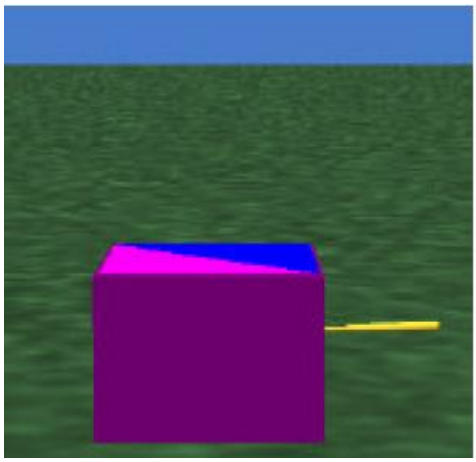
Cree un mundo con un pingüino y con una llave pegada en su espalda que sirva para darle cuerda. En este mundo, el pingüino camina en círculos alrededor del mundo mientras la llave siga girando.

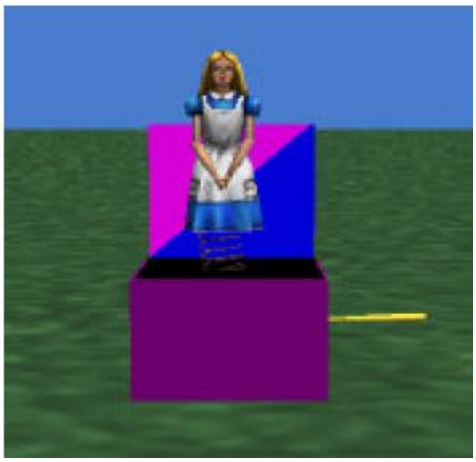
6 A la hora.

Coloque un reloj cucú en un mundo. La idea de esta animación simular el funcionamiento de esta clase de objetos — no en tiempo real. El minutero debe dar vueltas alrededor de la cara del reloj (tal vez una revolución completa dura 30 segundos en tiempo real), mientras que el péndulo debe oscilar hacia adelante y hacia atrás. Cuando el minutero haya hecho 1 revolución — desde las 12 hasta las 12 en la cara del reloj —, entonces el horario debe girar hacia el próximo número, se deben abrir las puertas, permitir que salga el pájaro, y hacer que suene por cada hora que haya transcurrido. Después, el ave debe ingresar al interior del reloj, a la espera de la próxima hora. La animación debe continuar hasta que el usuario decida lo contrario.

7 Alice en la caja.

Este mundo ilustra una caja de animación para niños. Como se muestra en la imagen inferior izquierda, la escena inicial contiene una caja cerrada, formada por un cubo con un cuadrado en su parte superior. Dentro de la caja, pero oculta a la vista, se encuentra Alice sujeta a un resorte. Se ha simulado la palanca de la caja con un bate de béisbol.





A principios de la animación, la caja está cerrada. La idea es que la manilla gire continuamente mientras reproduce un sonido. Cada 5 segundos, la parte superior de la caja se abre y Alice sale de ella., rebotando un par de veces por efectos del resorte. Cuando Alice se detiene, ella vuelve a descender en la caja y la parte superior se cierra, y todo vuelve a ejecutarse.

Resumen

Esta última parte de la tercera unidad nos muestra la potencia de las instrucciones **while**, como un efectivo mecanismo de repetición cuando no se tenga certeza de la cantidad de veces que se daba repetir un método o algún conjunto de instrucciones. Revisamos dos ejemplo: en el primero recreamos una persecución acuática, y en el segundo trabajamos con una carrera de caballos que resultó un poco más compleja, debido a algunas sutilezas matemáticas que hubo que emplear. A continuación, seguimos revisando algunas situaciones donde era necesario hacer infinitas repeticiones que permitieran lograr varias acciones en conjunto, junto con algunas técnicas que revisamos, y que resultaron ser equivalentes entre sí.

Conceptos importantes vistos en esta parte

- La sentencia **while** es un **bucle condicional**.
- La condición de tipo booleana utilizada para la entrada en una sentencia while es el mismo tipo de que se utiliza en las instrucciones **if/else**. La diferencia es la sentencia **while es un bucle**, y que la condición se comprueba de nuevo después que se hayan ejecutado las instrucciones que estaban dentro del bucle. Si es verdadero, continúa la ejecución del bucle, y si no, se sale.

- Dentro de los bucles **while**, se pueden anidar bucles finitos, instrucciones condicionales, bloques de tipo **hacer en orden**, **hacer juntos**, llamar a métodos, etc.

Proyectos

1 Whack-a-mole.

Debemos diseñar un juego que tiene como objetivo hacer clic en un objeto que constantemente aparece y desaparece de la pantalla. Si el usuario hace clic de manea exitosa en el objeto, deben ocurrir algunas acciones para que el jugador sepa que ha conseguido «*moler*» la cosa que aparece y desaparece. En el guión textual que se presenta a continuación, un hámster es el objeto que se mueve por todas partes, y una bandera se ocupa para señalar el éxito.

Supongamos que el hámster y el objeto bandera se han agregado al mundo, y que ambos están invisibles, dejando la propiedad **isShowing** en falso. El guión es el siguiente:

```
Mientras la bandera sea invisible
     1. Mueva el hámster aleatoriamente a una ubicación de la
     pantalla.
     2. Haga visible al hámster.
     3. Espere 0.25 segundos.
     4. Haga que el hámster sea invisible.
     5. Mueva al hámster 10 metros hacia abajo, de tal manera que
     el usuario no pueda hacer clic en él.
```

Cuando el usuario consiga hacer clic sobre el objeto, un método debe ser invocado para indicar el éxito. En nuestro ejemplo, la bandera deberá hacerse visible, haciendo que el juego se termine.

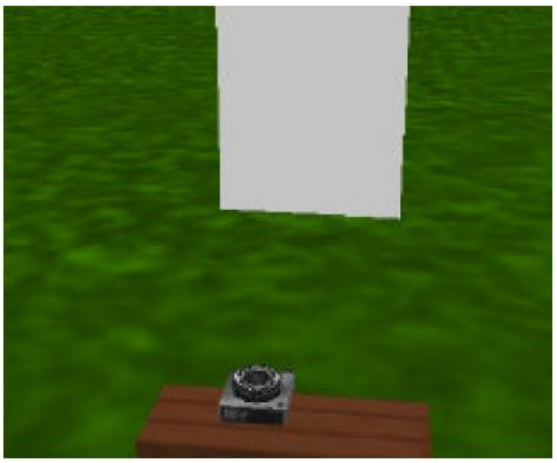
2 Autos chocadores.

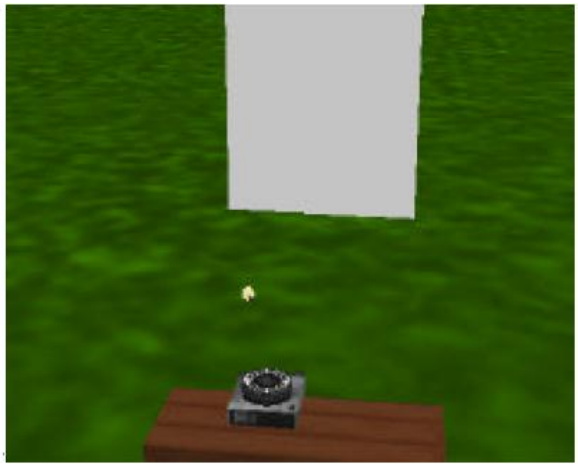
Cree una simulación de autos chocadores, donde los vehículos pasen juntos uno al lado del otro de manera continua dentro de la arena, como se muestra a continuación:

Tiene que incluir cuatro automóviles, y entre sí deben ir a una cierta distancia, o de la muralla, de manera tal que si uno se acerca a menos de 0.1 metro de otro o de la pared, debe cambiar de dirección para seguir avanzando. Utilice un interruptor de control para que el juego continúe o se detenga. En la medida que esté encendido, la animación se seguirá desarrollando.

3 La presentación sin fin.

Este mundo consiste en la creación de una interminable secuencia de diapositivas. En la simulación, emplee un cuadrado que hará las veces de telón, un proyector, una mesa y una ampolleta para provocar los efectos de iluminación de la pantalla. La escena inicial es la que se ilustra a continuación (nótese que la ampolleta está invisible):





Para un mayor sentido de realismo, comience con la luz del mundo y disminúyala antes que encienda el proyector. La presentación debe incluir al menos 4 diapositivas diferentes que deberán tener un número asignado. De esta manera, puede crear un método que le permita hacer la transición de una a otra. Cuando se ha terminado de mostrar una lámina, la luz del proyector debe disminuir ligeramente, y luego volver a encenderse para mostrar la otra diapositiva. Este proceso se debe repetir con todas las diapositivas, de forma indefinida.

Orientaciones Pedagógicas

Para conseguir los aprendizajes esperados, es necesario que los estudiantes tengan claridad con respecto a los siguientes puntos:

- Una estructura de control permite condicionar la ejecución de un conjunto o bloque de instrucciones, dependiendo del cumplimiento de ciertas condiciones.
- Una expresión consiste de alguna operación matemática donde intervengan valores numéricos, o de otra naturaleza.
- Una sentencia del tipo «*if-else*»:
  - Es un bloque de instrucciones que permiten un flujo de ejecución condicional.
  - Contiene condiciones de carácter booleana que determina en qué momento se va a ejecutar.
  - Los operadores lógicos se pueden emplear para combinar expresiones lógicas simples, para dar lugar a constructos más complejos.
- Las funciones:
  - Se pueden escribir para que retornen valores booleanos que puedan ser usados dentro una condición «*if-else*».
  - Se pueden escribir para computar y retornar otros tipos de datos.
- Un bucle tipo «*for*» requiere que el programador especifique la cantidad de veces que se vaya a repetir un bloque de sentencias.
  - En este sentido, se emplea cuando uno tiene absoluta certeza de la cantidad de veces que alguna acción — o conjunto de ellas — se vaya a repetir.
- Un bucle tipo «*while*» necesita que una condición booleana sea verdadera antes de entrar en el ciclo.

Para que estos entendimientos se puedan llevar a cabo de manera exitosa, se recomienda que los estudiantes desarrollen en las sesiones de práctica no sólo los problemas propuestos al final de la unidad, sino también que reflexionen en torno a estas preguntas, que se pueden distribuir en el tiempo que el profesor estime conveniente:

- ¿Cómo se determinan las estructuras de control que vayan a ser necesarias para poder resolver algún problema determinado?
- ¿Cuándo se puede emplear una función para ayudar en el proceso de toma de decisiones durante las etapas de diseño e implementación del programa?
- ¿Qué diferencia hay entre un método y una función?
- ¿Cuándo es más apropiado usar algún tipo de bucle dentro de un programa?
- ¿Qué tipo de bucle es más apropiado para alguna situación determinada?
- ¿Qué ocurriría si nunca se llegara a cumplir la condición de término de un bucle?

Unidad 4

Agrupación de elementos

En esta unidad revisaremos un par de estructuras de datos que nos permitirán agrupar elementos que sean de la misma naturaleza, y estudiaremos algunos ejemplos prácticos que dan cuenta de estas poderosas características. Terminaremos con el concepto de variable, y de cómo se puede integrar con la noción de herencia que se analizó en la segunda unidad.

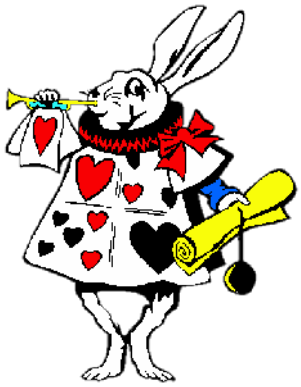
Sentido educativo de la unidad

Aprendizajes esperados.

- Identificar el concepto de lista y de arreglo.
- Reconocer las diferencias entre ambas estructuras de datos.
- Aplicar el concepto de herencia con la incorporación de variables asociadas con objetos.

Contenidos.

- Listas.
- Arreglos.
- Variables.



Primera Parte

Listas

Esta unidad abre una nueva característica que ofrecen los lenguajes de programación, que se denominan **estructuras de datos**. En las unidades anteriores, el objetivo era aprender a usar las instrucciones, métodos y funciones que le dan vida a los objetos que participen en el mundo virtual. Ahora nos vamos a centrar en la **organización de los elementos**, que vamos a combinar con todo lo que hemos aprendido en las unidades anteriores. Por el momento, habíamos trabajado con mundos que no tienen muchos objetos. Pero, ¿qué sucede si una animación requiriera tener 20 soldados marchando? En ese caso sería necesario crear por lo menos 20 instrucciones de animación (¿no será mucho?).

Una manera de manejar esta situación consiste en **recoger** o **agrupar** todos los objetos en una estructura que le permita al programador trabajar de manera más eficiente. Imagine por un momento que en lugar de controlar a 20 soldados, usted sólo tuviera que escribir unas pocas instrucciones para mover a todo el batallón. Para allá vamos.

En Ciencias de la Computación, hay muchas estructuras para organizar los conjuntos de datos que forman parte de un programa o de alguna animación. Por ejemplo, tenemos el caso de **listas, arreglos, matrices, pilas, árboles, bases de datos**, etc. En esta unidad aprenderemos el uso de listas y arreglos. Las otras estructuras quedan para cursos más avanzados de programación. De momento, una lista corresponde a una **colección de objetos del mismo tipo**.

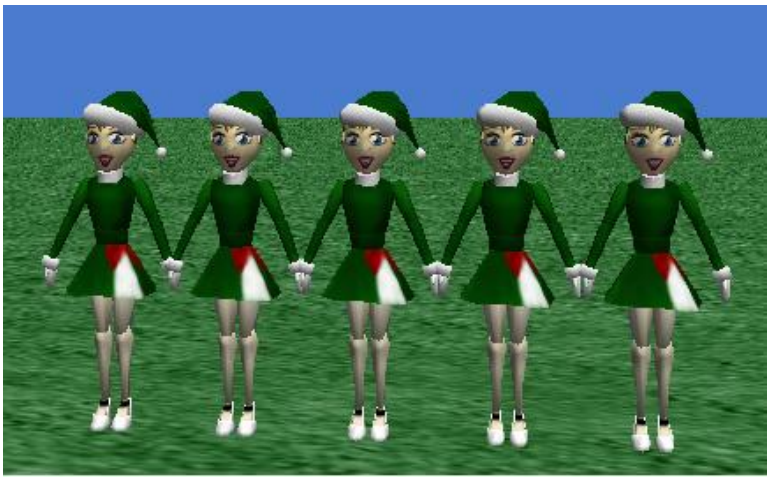
Partiremos viendo cómo se **crea una lista** en Alice, y cómo **iterar a través de ella**, por medio de grupos de instrucciones llamadas **para todos en orden** y **para todos juntos**, que ya veremos con más detalle. El grupo **para todos en orden** permite hacer una iteración secuencial, similar a lo que hace uno cuando da vueltas las páginas de un libro con los dedos. Como su nombre lo sugiere, **para todos juntos** permite hacer acciones simultáneas con todos los miembros de la lista.

Terminaremos esta parte mostrando aplicaciones prácticas de cómo usar una lista para aplicar algoritmos de búsqueda para encontrar algún objeto en base a una(s) propiedad(es) determinada(s).

4.1 Listas

Las listas son una de las estructuras de datos más usadas por los programadores para organizar la información. Ejemplos de listas especializadas son: **pilas, colas, listas enlazadas y doblemente enlazadas**, que se pueden encontrar en miles de aplicaciones computacionales.[23] Por lo general, en Alice las listas contienen objetos de similares características. En esta sección veremos cómo se crea una lista, y cómo se **itera** a través de ella, con el fin de buscar un elemento en particular.

Creación de una lista

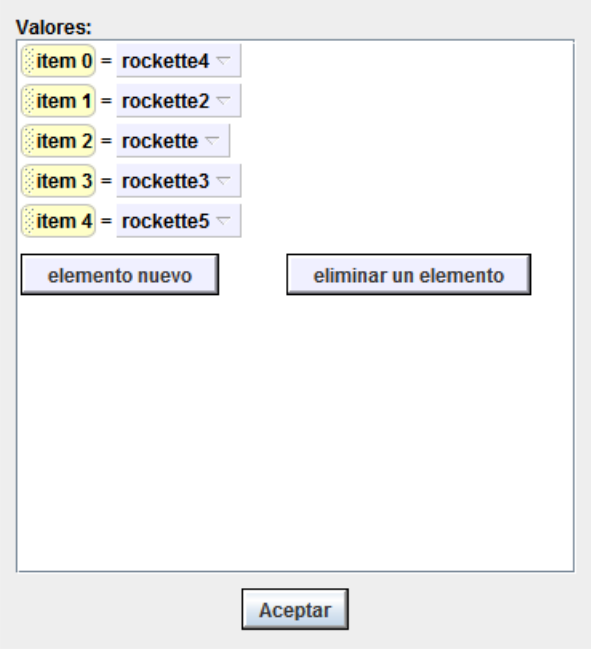


Antes de usar una lista, se tiene que crear. Supongamos que tenemos una escena inicial, donde aparece un grupo que contiene a cinco bailarinas iguales, como las que se muestran a la derecha. Nos gustaría crear una lista que represente a las bailarinas, que será muy útil cuando vayamos a crear alguna animación. Por ejemplo, podemos hacer que todas las bailarinas realicen una coreografía. Lo primero que haremos será incluir las cinco bailarinas a la escena, para que sea más fácil hacer la lista. Una vez que eso esté hecho, seleccione **mundo** en el árbol de objetos, y después presione el botón **crear una variable nueva** que se encuentra en la ficha **propiedades.**

[23] Por ejemplo, la administración del servicio de impresión se hace a través de una cola, mientras que las acciones de deshacer y rehacer se implementan a través de pilas y filas.

En el cuadro de diálogo emergente, ponemos un nombre para la lista. En este caso, la llamaremos **bailarinas**. Lo importante es que **deje marcada la opción hacer una lista.** A continuación, haga clic en el recuadro etiquetado como **agregar ítem** cinco veces, para acceder a cada una de las bailarinas, tal como se ve en la Figura 100:

Figura 100. Creación de una lista en Alice.



Iteración secuencial

Una de las operaciones más útiles que se pueden realizar con una lista es la **iteración secuencial** a través de ella. Esto significa que **a cada elemento de la lista se le entrega una instrucción** para que la realice, **una después de la otra**.

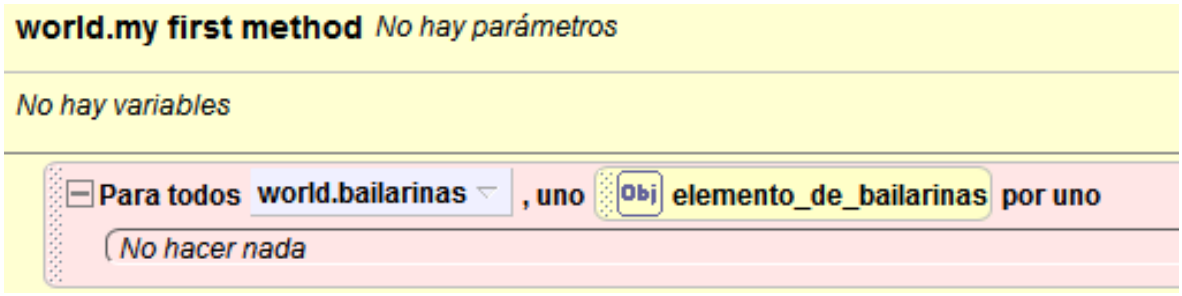
Este recorrido o barrido se puede lograr a través de la instrucción **para todos en orden**, como si fuera un cartero que va casa por casa repartiendo la correspondencia. Por ejemplo, vamos a hacer que cada bailarina — de izquierda a derecha — eleve la pierna derecha como parte de la rutina. El guión o código es el siguiente:

Tabla 18. Algoritmo para que todos los elementos de una lista hagan una acción en orden

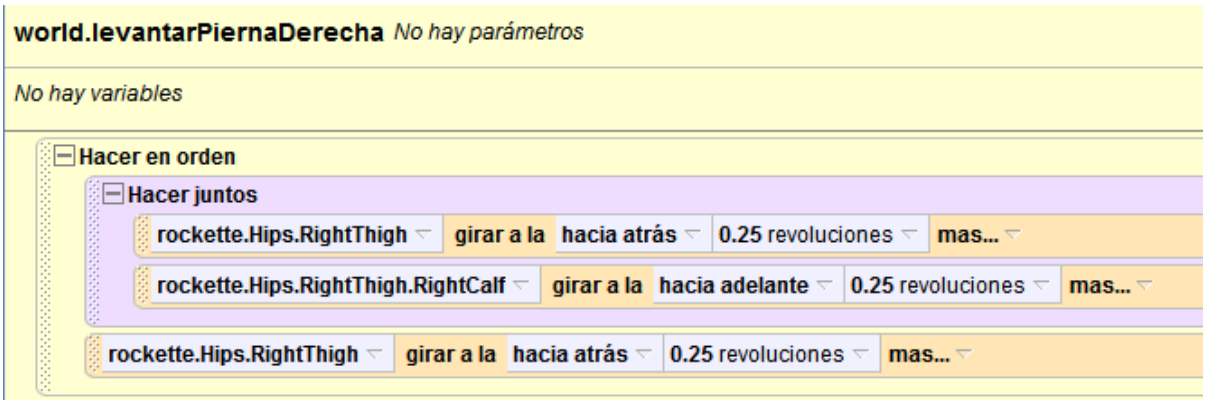
```
Para todas las bailarinas en orden
      elemento_de_bailarinas debe levantar la pierna derecha
```

En Alice, basta con arrastrar el bloque **para todos en orden** hacia el editor, seleccionando en el camino la lista con las bailarinas. El resultado se puede ver en la Figura 101:

Figura 101. Preparando un bloque de acceso secuencial a través de una lista.

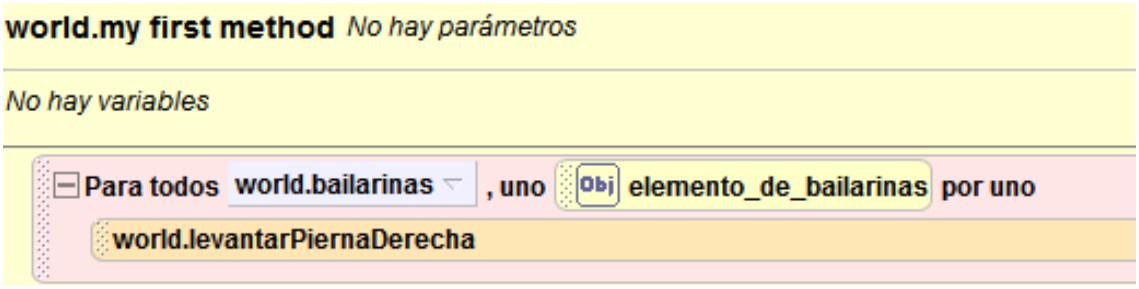


Ahora necesitamos escribir las instrucciones para que cada bailarina levante su pierna derecha. Para esto, vamos a construir un método para alguna en particular, y luego lo vamos a invocar en **para todos en orden**. La imagen adjunta muestra el código del método **levantarPiernaDerecha**. Se prefirió escribir un método, pero todo el grupo de instrucciones se podría haber escrito directamente dentro del bloque.



Ahora, siguiendo los pasos anteriores, hemos seleccionado un objeto bailarina como prototipo, y luego arrastramos el método **levantarPiernaDerecha** hacia el bloque que toma a todos los elementos de la lista en orden:

Figura 102. Ejemplo de algoritmo de iteración secuencial a través de una lista.

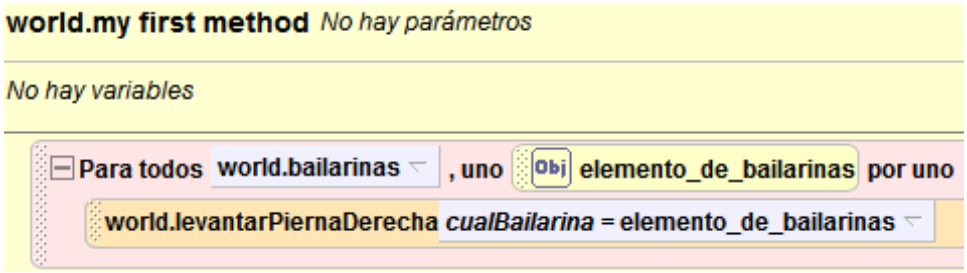


Si usted ejecuta el programa, se dará cuenta que sólo una de las bailarinas levantó la pierna, y que el resto no. Tampoco podemos arrastrar la variable **elemento_de_bailarinas** hacia el bloque, porque el método que creamos se aplica a una en particular. Entonces, vamos a tener que reinventar el método para permitir una bailarina arbitraria sea el parámetro.

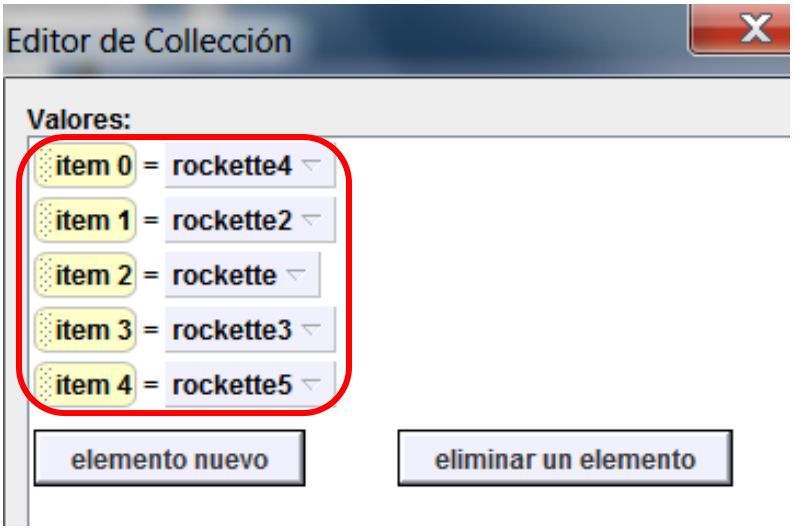
Para eso, crearemos un método con el mismo nombre, pero que opere a nivel de mundo, con la diferencia que le vamos a pasar un objeto bailarina como parámetro. Revise la nota técnica que está al final de este título para que comprenda mejor cómo se escribe el método:

Ahora, el método **levantarPiernaDerecha** lo trasladamos hacia dentro del bloque **para todos juntos**, como se ilustra en la Figura 103:

Figura 103. Todos los elementos de una lista efectúan una operación, de a uno a la vez.



Cuando se ejecuta el programa, cada bailarina de la lista levanta la pierna derecha. Tenga cuidado con el orden en el cual agregó a las bailarinas, ya que el recorrido de la lista se hace desde **ítem 0** hasta **ítem 4**. Entonces, si usted presiona a la derecha del signo igual en la lista **bailarinas**, puede tener algo como esto:



Según este catálogo, el orden en que debieran levantar la pierna es:

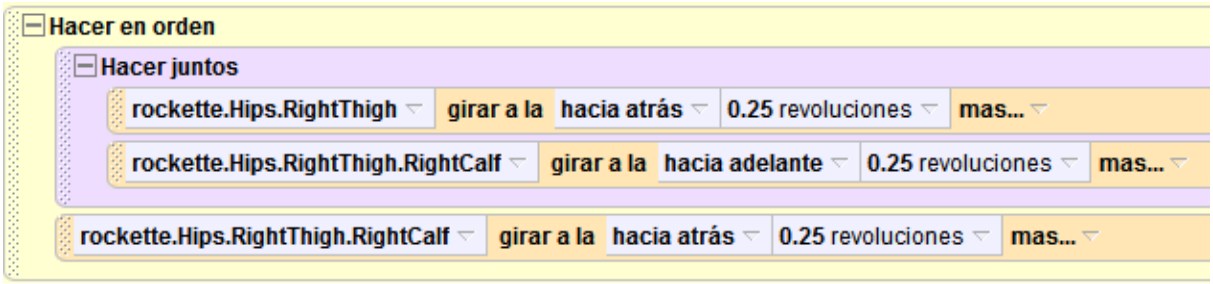
bailarina4 – bailarina2 – bailarina1 – bailarina3 – bailarina5

Si la animación se ve desordenada, puede reacomodar los elementos, haciendo clic en la posición que desea modificar, y eligiendo el objeto correcto.

Nota técnica

Usted habrá notado que al crear el método de mundo **levantarPiernaDerecha**, no tuvo la posibilidad de haber escogido exactamente cuál era la parte del cuerpo que debía elevar. La razón de esto es que no todos los objetos tienen piernas, y por lo tanto, Alice no puede asumir que usted quería trabajar con bailarinas. Esto no significa que el método no se pueda hacer, sino que el camino para escribirlo no es tan trivial. Veamos cómo hacerlo:

1. Vamos a crear un parámetro de tipo **objeto** llamado **cualBailarina**.
2. Hacemos el código del método como si estuviéramos tratando con una bailarina:



3. Tome nota de los nombres de las partes del cuerpo afectadas :

Hips.RightThigh y **Hips.RightThigh.RightCalf**

4. Elija una bailarina cualquiera del árbol de objetos, y arrastre desde la pestaña de funciones aquella que se llama **la parte del** `rockette` **se llama** `key` hacia el editor:

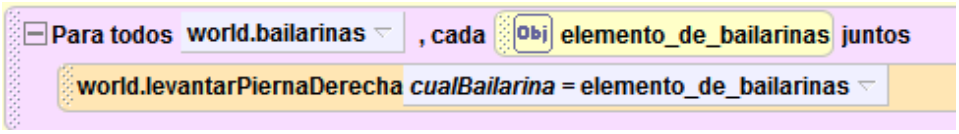
Una vez que suelte, tendrá que anotar los nombres de las partes del cuerpo del paso 3.

5. Repetimos la secuencia de pasos anteriores para completar el código, y llegar a esto:

Iteración concurrente

Alice también dispone de una construcción denominada **para todos juntos** que permite que todos los objetos de la lista realicen una misma acción de forma simultánea. Una manera alternativa de ver este bloque de instrucciones es lo que ocurre con los centros de llamados, donde todas las personas que están en la lista pueden estar hablando por teléfono al mismo tiempo. La Figura 104 muestra cómo quedaría el código del ejemplo de las bailarinas, si permitimos que todas ellas levanten la pierna derecha al mismo tiempo:

Figura 104. Ejemplo de iteración concurrente.



Cuando ejecute el código, todas las bailarinas levantan su pierna derecha. Es importante destacar en esta parte que **para este tipo de instrucciones simultáneas, no es necesario preocuparse por el orden que tengan los elementos de la lista.**

Ejercicios

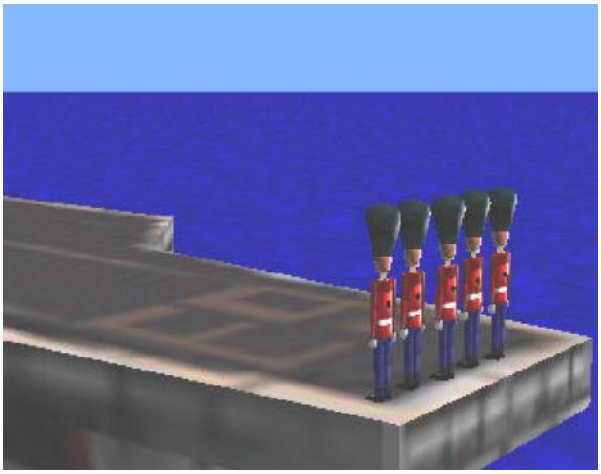
1 La ola.

Cree una animación que simule a unos personajes hacer «*la ola*» al interior de un estadio. Para estos efectos, vamos a confeccionar una escena inicial con cuatro personajes que usted puede elegir a su gusto.

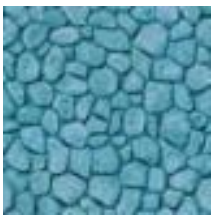
Utilice el **para todos en orden** con el fin de lograr que todos levanten sus brazos en orden.

2 Marcha de soldados.

El propósito de este ejercicio es practicar el uso de **para todos juntos**. Diseñe un mundo como el que se muestra en la imagen adjunta, que contiene a cinco soldados al borde de un portaaviones, que tienen que ejecutar una marcha sincronizada:

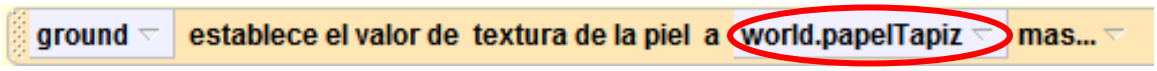


3 Mapas de texturas.



En varios sitios de la web, usted podría encontrar un fondo de pequeñas piedras azules que puede importar en Alice como mapa de textura, y luego cambiar el estilo del suelo por el diseño personalizado. Por ejemplo, podríamos emplear la imagen de la izquierda. Para esto, vamos al menú **Archivo**, elegimos la opción **Importar**, y terminamos con la elección del archivo de imagen en nuestro disco duro. La

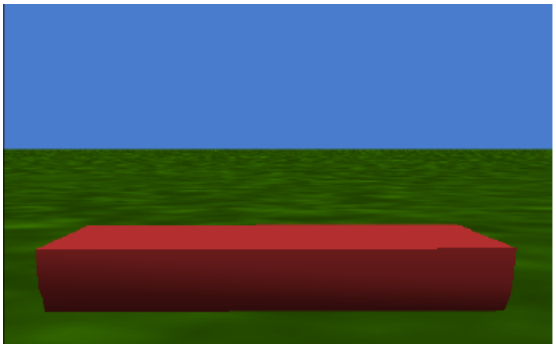
instrucción para alterar la textura del suelo se llama **textura de la piel**, que basta con arrastrar hacia el editor, y elegir el nuevo valor:



Descargue cinco tipos de suelos .En un mundo nuevo, importe los archivos de fondo como mapas de textura, y cree una lista con los cinco archivos que descargó. Escriba una animación que recorra las diferentes texturas, de a una a la vez, de tal manera que se recree el efecto especial de hacer que el paisaje cambie cada 5 segundos. Cada vez que la textura del suelo se cambia, el terreno cambia a un color que sea apropiado.

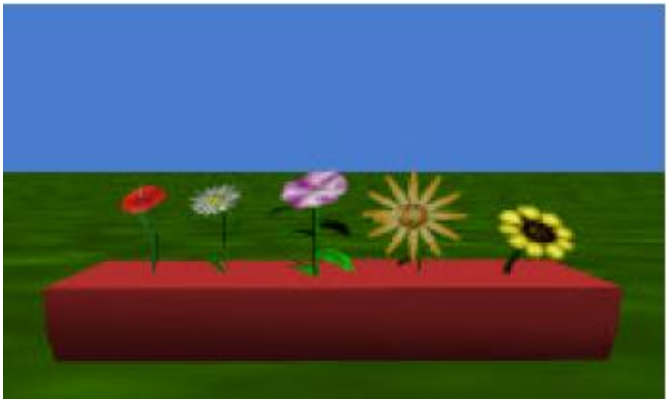
4 Flores de primavera.

La primavera ha llegado, y se espera con ansias la crecida de las primeras flores. Para esto, se pide que cree una escena inicial con 5 flores a su elección, de tal forma que se en la escena inicial no se vean. Haga una lista que contenga a las flores que agregó:



a). Cree una animación donde cada una de las flores de la lista crezca hacia arriba, de una en una.

b) Cree una segunda versión donde todas las flores crecen al mismo tiempo. En ambos casos, el resultado debiera ser algo como esto:



5 Patinaje Olímpico.

Cree una animación que simule la práctica de una patinadora de hielo para los próximos Juegos Olímpicos. Utilice una lista llamada **cuantasVueltas** que guarde el número de

veces que la patinadora hace la figura del ocho seguida de un giro completo. La patinadora debería patinar la figura una vez, dos veces, y así sucesivamente.

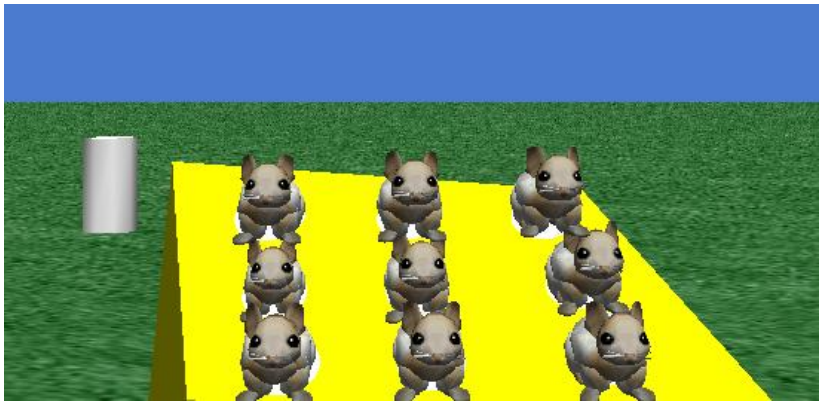
Haga que la patinadora levante las manos a la audiencia cada vez que haya terminado una rutina, lo que se debe interpretar como una señal de que un ciclo termina y que otro empieza. También, a la patinadora se le ha entregado un nuevo juego de zapatos de skate que realmente van a sorprender a los jueces, porque tienen la particularidad de cambiar de color mientras ella está en el hielo. Para hacer que los patines cambien de color durante la animación, construya una lista que contenga diferentes colores que se llame **colorDelPatin**. El guión de trabajo es el siguiente:

```
Para todos los elementos de la lista cuantasVueltas, hacer en orden:
    Hacer juntos:
        Iterar usando el elemento actual de la lista cuantasVueltas
        Para todos los elementos de la lista colorDelPatin, hacer en orden
            a) Cambiar simultáneamente el color de los patines derecho e izquierdo.
```

Búsqueda dentro de una lista

A menudo nosotros hacemos búsquedas empleando algún motor de la web, como por ejemplo Google o Yahoo, y en cuestión de segundos tenemos una lista de enlaces hacia los sitios que coincidan con los **criterios de búsqueda**. Una **búsqueda dentro de una lista** es una operación bastante común a nivel de Ciencias de la Computación. Lo que acá se hace es **iterar a través de una lista**, y comprobar si acaso cada elemento satisface o no lo que andamos buscando. En esta sección vamos a explorar cómo hacer una búsqueda dentro de una lista con Alice.

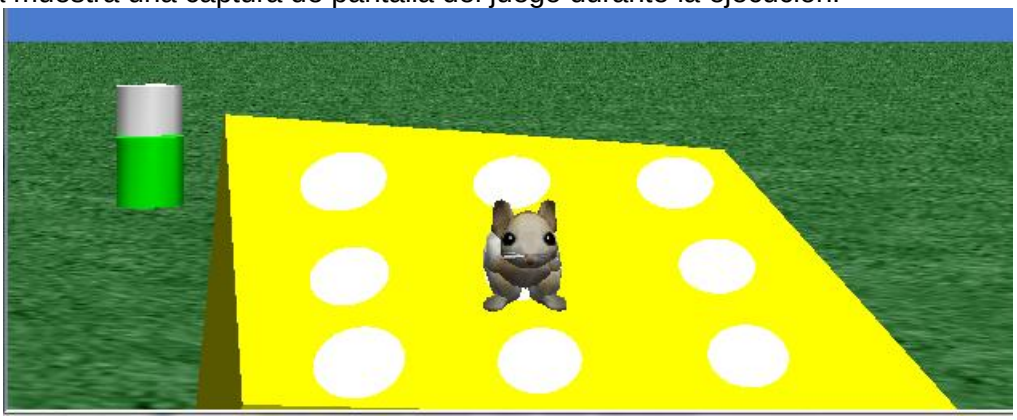
Simulación de una búsqueda dentro de una lista



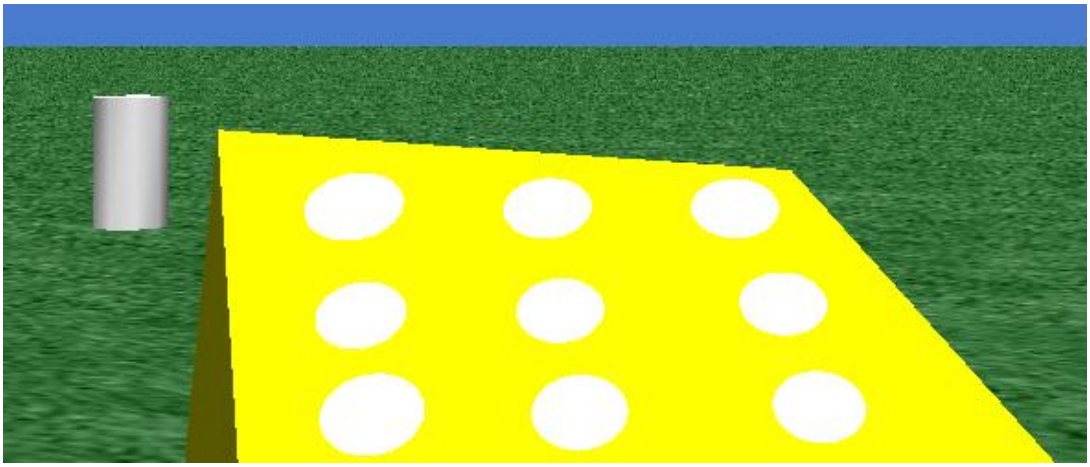
Como ejemplo de una búsqueda dentro de una lista, rememoremos el juego llamado **whack a mole**, que tuvo que implementar para uno de los proyectos finales de la tercera unidad. En resumen, se trata que un jugador tenga un martillo y golpee a un títere que aparece desde una caja en el stand de feria. Cada vez que el jugador golpea al títere que sale de los cilindros blancos, anota más puntos. En nuestro caso, vamos a machacar a un hámster. En la imagen arriba podemos apreciar un cubo amarillo con nueve círculos blancos que se han añadido a la parte superior. Nueve hámsters se han agregado a la escena, y se

colocan por encima de ellos. Finalmente, 2 cilindros denominados **puntaje total** y **puntaje del jugador** se han agregado en el fondo para que actúen como un marcador visual primitivo. El cilindro **puntaje total** representa el total de la puntuación, es de color gris y está situado sobre el suelo. El cilindro **puntaje del jugador** es de color verde y se ha dispuesto en la escena, con su parte superior por debajo de la superficie de la tierra, de tal modo que no alcanza a salir en la escena inicial. Además, este cilindro representa la puntuación actual del jugador, y es igual a cero al inicio del juego. La idea es que a medida que el jugador gane puntos, el cilindro verde vaya emergiendo del suelo, para hacer el efecto óptico del aumento del puntaje.

La idea es que un hámster aparezca y desaparezca. Cada vez que el jugador tiene éxito al hacer clic justo en un hámster mientras está en exhibición, el cilindro **puntaje del jugador** tendrá que subir, lo que indica una aumento en la puntuación del jugador. Una vez que el cilindro **puntaje del jugador** se haya elevado por completo, entonces tendrá la misma altura del cilindro **puntaje total**, y en este punto el jugador habrá ganado la partida. Cuando esto ocurra, los hamsters deben desaparecer hacia abajo. La imagen adjunta muestra una captura de pantalla del juego durante la ejecución.



En ella se aprecia que la altura del cilindro **puntaje del jugador** está bastante elevada, lo que significa que el usuario está bien encaminado para terminar con éxito la partida. Esta escena vamos a tener cuando el mundo se inicie. Para lograrla, debe hacer que todos los hámsteres tengan establecido el parámetro de visibilidad en **falso**:



En la planificación de una solución a este problema, podemos definir dos subtareas:

Subtarea No. 1. Creación de un bucle que permita que la subida y bajada del hámster sea al azar, **mientras que el juego está en marcha**, que en términos algorítmicos podemos expresar de esta manera:

```
Evento: Cuando el mundo empieza
Respuesta: Llamada al método mi primer método, que está en el
mundo.
Método: mi primer método
      Mientras el cilindro puntaje del jugador no esté por completo
      encima del suelo
            Hacer aparecer y desaparecer uno de los 9 hámster al
            azar.
```

Subtarea No. 2. Aumentar la puntuación cada vez que el jugador logre hacer clic en uno de los hámsteres. En forma de guión textual, tenemos el siguiente evento:

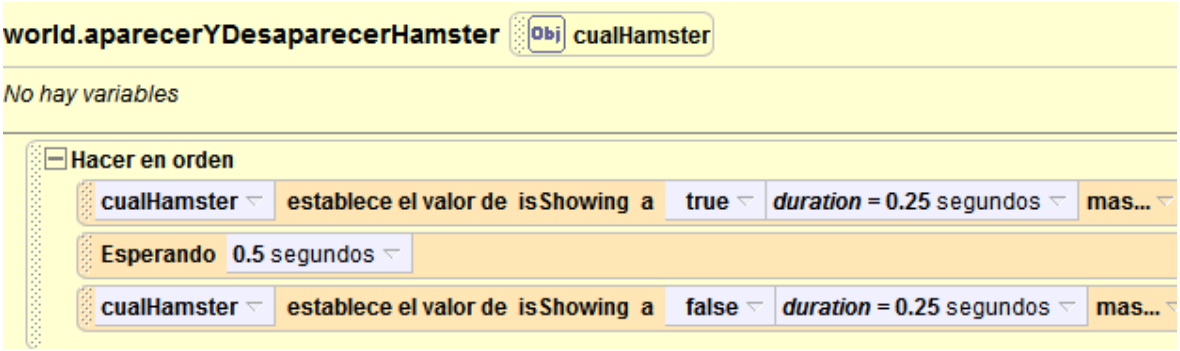
```
Evento: El usuario hace clic con el ratón
Respuesta: Llamada al método puntaje que está en el mundo.
Método: puntaje
      Si el ratón hace clic en uno de los hámsteres
            Mueva el cilindro puntaje del jugador hacia arriba
            En forma opcional, reproducir un sonido
```

Primera tarea: Dejar el juego en marcha

Cuando el mundo comienza, el juego tiene que hacerlo de inmediato. Una de las características que tiene Alice es que de **forma predeterminada parte ejecutando el método «*mi primer método*»**. Entonces, nos aprovechamos de eso, y ponemos el código en este método.

Por lo tanto, ahora nos queda crear el bucle para lograr que la acción sea continua a lo largo del desarrollo del juego. Recordemos que la condición dice que **mientras el cilindro del puntaje del jugador no esté por encima del suelo**, lo que significa que el usuario aún no ha ganado la partida. Podemos crear un método llamado **aparecerYDesaparecerHamster**. Ahora bien: ¿por qué es necesario crear uno? Simplemente porque es una acción que se tendrá que desarrollar muchas veces a lo largo del juego, y por un asunto de legibilidad es mejor tenerla «*empaquetada*» en un método, para que su uso sea más cómodo. Tenemos un total de nueve hámsteres, por lo que necesitamos un parámetro que nos indique a cuál animal estamos haciendo referencia. El nombre que le pondremos es **cualHamster**.

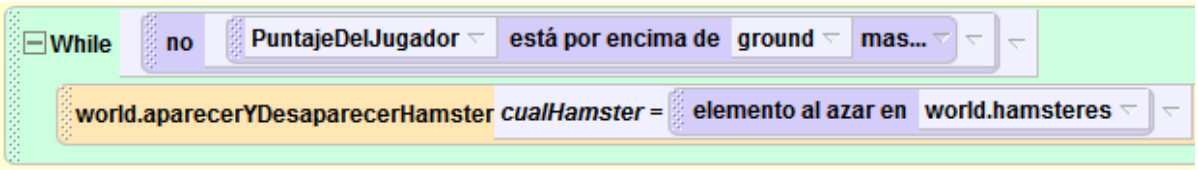
En esta imagen, podemos ver la codificación de las acciones que permiten hacer aparecer y desaparecer a uno de los objetos durante 0.25 segundo. Por supuesto, usted siéntase libre de modificar este valor como estime necesario.



Ahora tenemos que averiguar cómo pasarle a un hámster al azar al método **aparecerYDesaparecerHamster**. Una forma es **organizar los hámsteres en una lista**, y a continuación seleccionar al azar uno de ellos. Una lista de hámsteres se crea como se ha descrito anteriormente en la sección 4.1, y le pondremos el poco original nombre de **hámsteres**, pero que resulta bastante práctico por ser auto explicativo. **Recuerde la observación que se hizo con respecto al ordenamiento de los elementos en una lista**.

Hasta el momento hemos creado el método para hacer aparecer y desaparecer un hámster, junto con la lista **hámsteres**. Por lo tanto, podemos escribir un código similar al que se muestra en la Figura 105, en el cual la condición de permanencia del bucle **while** es que **el cilindro que representa al puntaje del jugador NO se encuentre por debajo del suelo**, de tal manera que si llegara a estar completamente por encima, el juego termina. Esta es primera vez que planteamos una condición lógica empleando el **not**, y lo hacemos porque si no estuviera puesto, el bucle se ejecutaría en forma indefinida, y el juego nunca terminaría. Piense que apenas el usuario hace un clic en el lugar correcto, inmediatamente el cilindro se empieza a levantar de la tierra, pero cuando esté por sobre el nivel del suelo, no es necesario seguir haciendo algún recuento de puntaje:

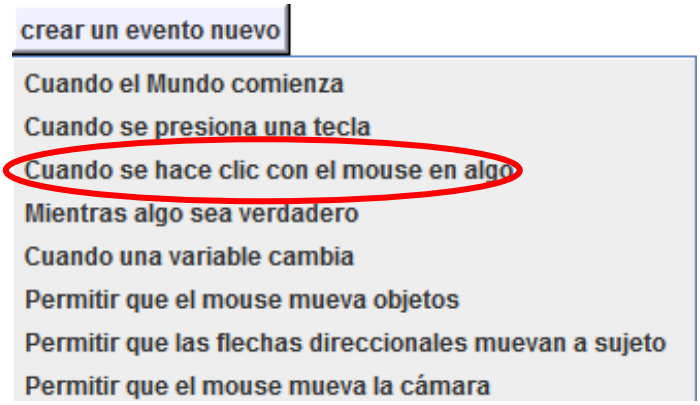
Figura 105. Un bucle «*while*» que contiene una negación dentro de su declaración condicional.



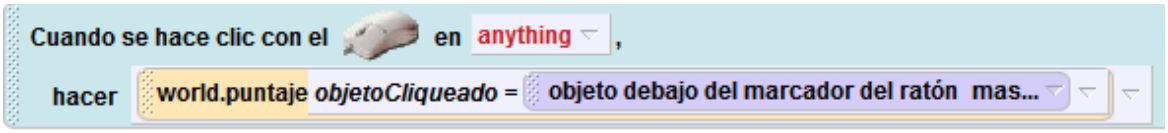
Para enviar un hámster al azar desde la lista creada, arrastramos **hamsteres** hasta el parámetro del método **subirYBajarHamster**, y luego elegimos **ítem al azar** desde el menú emergente. De esta manera, cuando el código se ejecute, un hámster al azar desde la lista se pasa como argumento al método **aparecerYDesaparecerHamsterv** cada vez que la sentencia **while** se repita. Cuando el cilindro **puntaje del jugador** sobrepase el del puntaje total, el bucle termina.

Segunda tarea: Llevar el recuento del puntaje

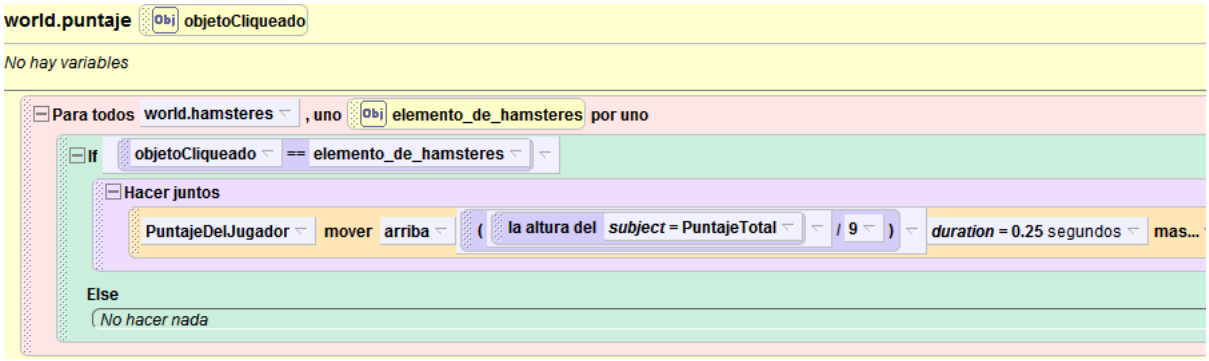
Vamos a escribir un método de puntuación para mostrar visualmente el éxito del jugador. Cada vez que el usuario haga clic sobre un hámster, el cilindro **puntaje del jugador** se va a levantar del suelo. Fundamentalmente, todo se basa en el evento de hacer clic con el ratón:



En respuesta al clic, vamos a llamar a un método que denominaremos **puntaje**. Cuando se produce este evento, se notifica a Alice que el ratón ha hecho clic en algo, y el programa puede determinar de qué se trata.



El método **puntaje** se muestra en el código adjunto, en el cual se pregunta si acaso el argumento que se está pasando como valor del parámetro es o no uno de los hámsteres.



En el método del puntaje, se ha utilizado una construcción **para todos en orden**, con el fin de **iterar a través de la lista**. De hecho, **esa es la forma en la cual se recorren los valores almacenados en esta estructura de datos**.

De esta manera, se **compara cada objeto almacenado en la lista con lo que el usuario haya pasado como argumento a través del clic**. Este es el momento donde la búsqueda dentro de la lista tiene lugar. Siguiendo con la idea: **si** lo entregado por el jugador efectivamente corresponde a un hámster, **entonces** el cilindro **puntaje del jugador** tiene que subir**,** y el hámster hace un ruido extraño.

Lo que se ha ilustrado en este ejemplo corresponde a la manera en cómo podemos buscar algún elemento dentro de una lista que satisfaga alguna propiedad en particular, que en este caso corresponde al nombre del objeto. La búsqueda se realiza mediante un bloque **para todos en orden**, para iterar o moverse dentro de la lista, con el objetivo de comprobar si alguno de los hámsteres coincide con el clic del jugador.

Resumen

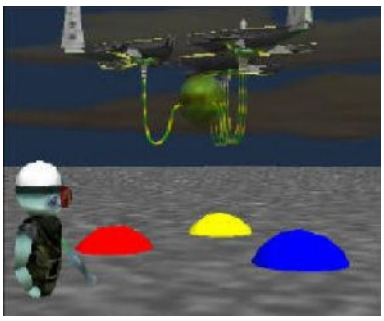
El principal objetivo de esta unidad era introducir el concepto de lista, y de la importancia que tiene como **estructura computacional que está pensada para organizar datos** y la información sobre ellos. Alice proporciona dos mecanismos para **iterar** a través de una lista: secuencial, a través de la construcción **para todos en orden**, y simultánea, gracias al **para todos juntos**.

Conceptos vistos hasta el momento

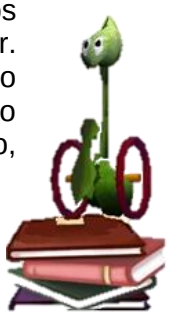
- Una **lista** es una colección de objetos que por lo general son del mismo tipo.
- Se puede **iterar** a través de los elementos de una lista de forma secuencial (**para todos en orden**) o simultánea (**para todos juntos**).
- Cuando una lista contiene objetos, y éstos tienen que hacer alguna acción, el elemento de la lista se suele pasar como parámetro de un método que va a afectar al resto de los miembros.

Proyectos

1 Transmisión de código.



Crear un mundo interactivo que permita al usuario enviar un mensaje a una nave espacial que está en una misión. El usuario debería ser capaz de hacer clic en los módulos rojo, amarillo y azul en alguna secuencia particular. Cuando termine de hacerlo, deberá hacer clic en el casco de color blanco del robot, para que éste a su vez lo retransmita hacia la nave. Si su computador tiene sonido,

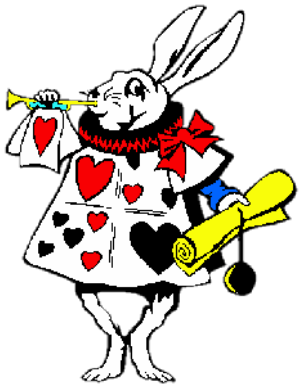


asígnele una pista de audio a cada color, y que suene mientras se transmite el mensaje.

2 El tesoro del fantasma.

En este juego, Ghost está fuera de su casa porque está participando en una cacería del tesoro en la noche de Halloween. En este juego, el fantasma se encuentra en un parque cementerio, en el cual se ha dejado un tesoro por debajo de cada lápida. Cuando el fantasma se acerca a una tumba, la lápida se abre y el tesoro se revela ante sus ojos. La tarea de Ghost consiste en visitar todas las tumbas y descubrir todos los tesoros. Pero no todo es tan simple: hay tres esqueletos que están patrullando el cementerio, y caminan muy cerca de las tumbas para proteger los tesoros. Si Ghost se acerca demasiado a un esqueleto, automáticamente se evapora, y el jugador pierde la partida.

Cree un juego para animar la búsqueda del tesoro de Halloween. Para estos efectos, use una lista que le permita almacenar 10 lápidas. Las lápidas están todas alrededor de la casa central (note que la captura de pantalla muestra sólo la parte delantera). El usuario tiene que usar las flechas del teclado para mover a Ghost por los alrededores del cementerio. Hacer que la cámara siga a Ghost, para que siempre esté a la vista. Ocupe una segunda lista donde guarde el estado de cada lápida: verdadero si fue visitada, y falso en caso contrario. Si Ghos rescata todos los tesoros, el juego termina. Por otra parte, si Ghost se encuentra con un esqueleto a muy poca distancia, el juego llega a su fin y Ghost se evapora en el aire.



Segunda Parte

Arreglos

Un **arreglo** es una **estructura que nos permite agrupar elementos del mismo tipo** en un grupo. En la primera parte de la unidad, vimos que una **lista también corresponde a una agrupación de elementos**. Por lo tanto, cabe preguntar: ¿en qué se diferencia una estructura de la otra? La respuesta radica fundamentalmente en lo siguiente:

1. En un arreglo, se ocupa un elemento llamado **índice** que indica **la posición que tiene un elemento** dentro de esta colección.

2. Los arreglos tienen un **tamaño que es fijo**, mientras que en las **listas esta característica es variable**. El **largo de un arreglo** se define como **la cantidad de elementos que lo componen**.

Una de las ventajas que tiene el uso de arreglos, es que no es necesario **iterar** a través de toda la estructura para llegar al elemento deseado, sino que uno puede acceder **de manera directa a través del índice**, al revés de lo que pasa con las listas donde estamos obligados a hacerlo. Esto significa que sin perjuicio de lo anterior, también podamos recorrer el arreglo.

Por otra parte, **no es necesario que los arreglos contengan objetos**, pero en los ejemplos que revisaremos sí los tienen, para que se pueda visualizar mejor el poder computacional que nos provee esta herramienta. Una vez que hayamos construido un arreglo, veremos cómo podemos recorrerla para buscar algún elemento que tenga una propiedad determinada, tal como lo hemos hecho con las listas. Algunos de los mecanismos que estudiaremos se presentan como una manera de introducir el concepto de **variable**, que se presenta en la tercera parte de la unidad. Este último concepto nos ayudará a simplificar la escritura de métodos para hacer determinadas operaciones con arreglos, mediante el uso de variables.

4.2 Arreglos

Un arreglo es una estructura que sirve para organizar datos y objetos en una colección o grupo, que a diferencia de una lista sus elementos se pueden **acceder** a través de un **índice**, y que su tamaño es **fijo**. Vamos a hacer uso de una analogía con el mundo real para explicar este concepto. Digamos que un arreglo fuera un cd de música, en el cual **la etiqueta** contiene **el listado de todas las canciones**. Por otra parte, si usted utiliza el cd, puede escuchar **todas las canciones**, una por una — que sería hacer una **iteración completa dentro del arreglo** —, o bien puede **seleccionar específicamente el número de pista** que quiera escuchar. En este caso, **el número de pista corresponde con el índice** que se había mencionado en la introducción. Por lo tanto, si quere escuchar la canción número 7, basta con elegirla y escuchar. Si hacemos el ejercicio de **comparar un arreglo con una lista**, podemos pensar que ésta tenga un comportamiento similar a un **cassette**, en el sentido que puede tener todas las canciones grabadas, pero si quiere llegar a la número 7, tendrá que avanzar rápido a través de la cinta para llegar a la pista deseada. Entonces, tanto el cd — visto como un **arreglo** — como el cassette — visto como una **lista — contienen la colección de canciones, pero la forma de acceder a ellas es diferente**.

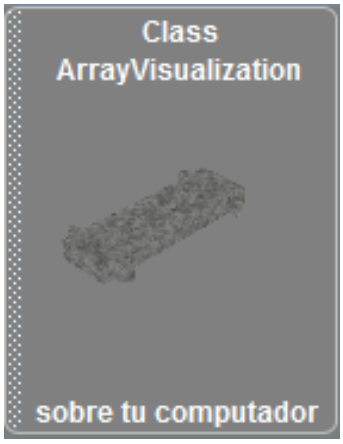
En esta parte, vamos a trabajar con arreglos de objetos, debido a que éstos son visibles en nuestros mundos virtuales. En algunos mundos se demostrará la acción de **iterar a través del arreglo**, de a un objeto a la vez. Veremos también cuáles son las operaciones que nos permitirán acceder a los elementos del arreglo.

Creación de un arreglo

Para crear un arreglo, empezaremos por agregar objetos al mundo, tal como lo hemos venido haciendo hasta el momento. Después vamos a incluir una instancia de, que está dentro de la galería local. Este elemento define ciertas propiedades **ArrayVisualization** y acciones para un objeto que ha sido especialmente diseñado para representar una estructura tipo arreglo. Es importante destacar que **los arreglos no son objetos visibles**

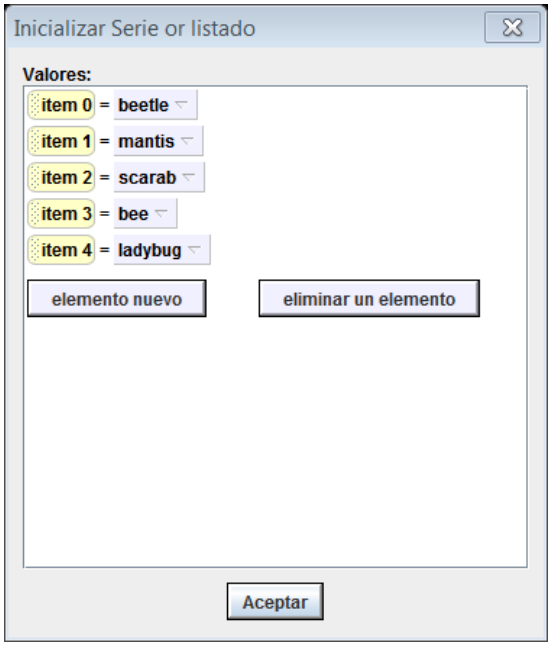
en el mundo. Por lo tanto, el uso de **ArrayVisualization** es para mostrar mejor cómo opera un arreglo. El último paso consiste en agregar los elementos hacia la estructura.

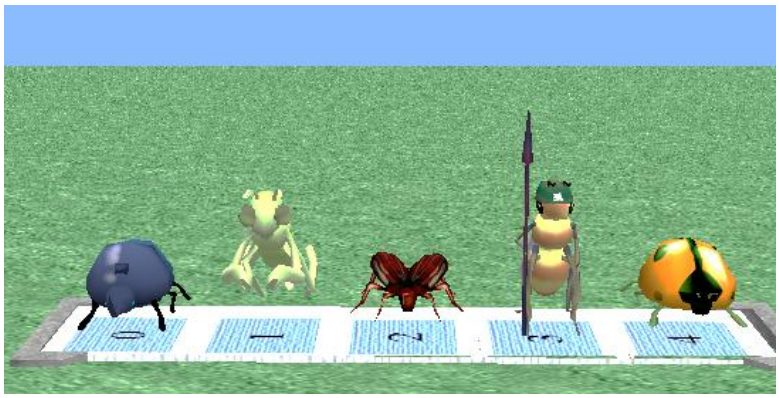
Para mostrar cada uno de los pasos involucrados en la creación de un arreglo, vamos a diseñar uno que contenga cinco bichos: escarabajo, mantis, otro tipo de escarabajo, abeja y china. Esta imagen nos muestra los cinco elementos que aún no han sido agregados al arreglo:



Después, vamos a la galería local y buscamos **ArrayVisualization**, como se muestra en la imagen de la izquierda. A continuación, se abre un cuadro de diálogo donde usted podrá ingresar cada uno de los elementos al arreglo, como se muestra en la Figura 106. Presione en el botón **elemento nuevo**, y agregue los elementos de a uno. Tenga en cuenta que **se debe agregar el objeto completo, no una parte**. Al final, debiera tener algo como lo que se muestra en la Figura 106. Es importante observar que el recuadro de abajo corresponde a la **estructura final** que le vamos a dar al arreglo, en términos del orden que le demos a sus elementos. Por lo tanto: **es completamente irrelevante la forma en cómo los objetos se hayan arrastrado inicialmente al mundo**; lo que cuenta es el orden que ahora se les dé.

Figura 106. Inicialización de un arreglo de objetos.

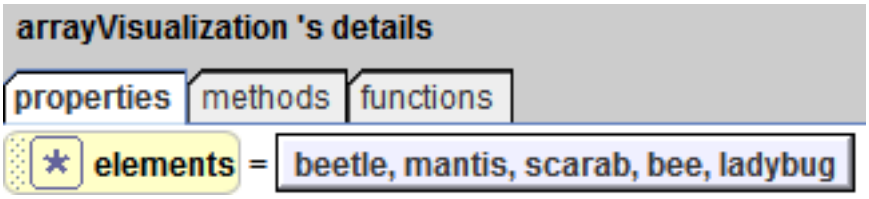




Por supuesto que es posible alterar la posición de alguno de los elementos del arreglo, en caso que nos hayamos equivocado. Al igual que en la lista, seleccionamos el elemento que queramos cambiar, y elegimos el nuevo destino a partir del menú que se despliega hacia abajo.

Note además que el arreglo **empieza en la posición cero**, de tal manera que si tenemos cinco elementos, **el último índice será igual a cuatro**. Cada uno de los números que acompaña a la palabra **ítem** se llama **índice**. Entonces, el escarabajo **está en la posición cero**, y así sucesivamente. En Ciencias de la Computación, **el primer índice es el cero, y no el uno como se espera**. Tenga en cuenta además que **ningún elemento puede estar simultáneamente en dos posiciones distintas del arreglo al mismo tiempo**. Por ejemplo, una mantis no puede estar en las posiciones 1 y 4, a no ser que tengamos **dos objetos distintos** mantis1 y mantis2.

Una vez que estamos listos con la inclusión de los elementos, presionamos el botón **OK**. Si usted cliquea en el objeto **ArrayVisualization**, podrá notar que se ha creado una **variable** llamada **elements** que contiene a todos los miembros del arreglo, en el orden que usted haya especificado:

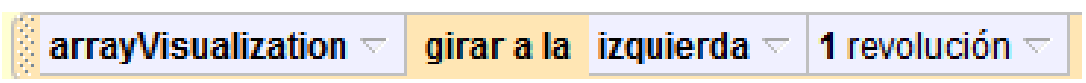


Cómo acceder a los elementos de un arreglo

Si queremos la abeja — que se encuentra en la posición 3 del arreglo — de una vuelta a la izquierda, podemos escribir la misma instrucción que hemos aplicado en los programas anteriores:

Para aprovechar las características que ofrecen los arreglos, podemos seguir estos pasos:

Paso No. 1. Seleccionamos **ArrayVisualization** desde el árbol de objetos, y arrastramos el método de giro hacia el editor:



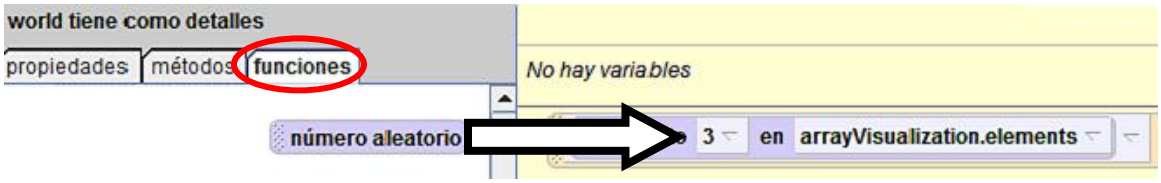
¿Qué ocurre si usted ejecuta el código que se acaba de mostrar?

Paso No. 2. Arrastramos la variable **elements** hacia el editor, y hacemos que sobre escriba la palabra **ArrayVisualization**. Después elegimos la posición 3, y llegamos a esto. Después de este paso, basta con correr el programa.

Figura 107. Acceso a una posición específica dentro de un arreglo.

Cómo acceder a un elemento aleatorio

En algunas ocasiones puede ser útil intentar obtener algún elemento de un arreglo en forma aleatoria. Para esto, lo único que se necesita es que el **índice** se obtenga a partir de un número aleatorio, lo que se obtiene arrastrando la función que entrega un **número aleatorio** desde el mundo hasta el editor de código, como se muestra a continuación:



Basta con pedir que **el número máximo sea igual al tamaño del arreglo**, y que éste **sea un número entero**. El tamaño se obtiene en el grupo de funciones asociado con **ArrayVisualization**, tal como se muestra arriba. Después de eso, el código debiera quedarnos así:

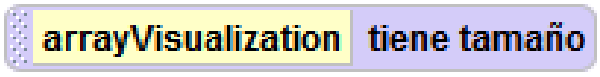
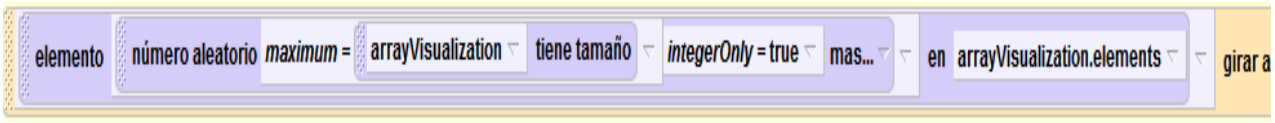
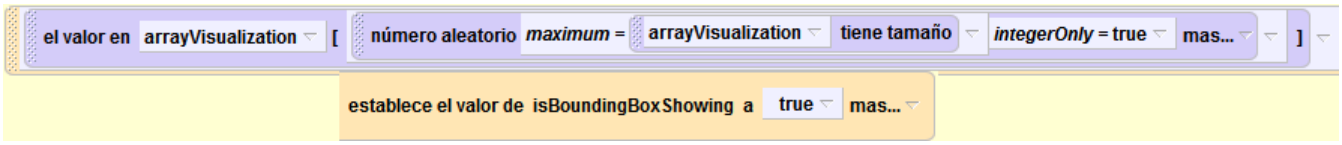


Figura 108. Acceso a una posición aleatoria de un arreglo.



Búsqueda de un elemento dentro de un arreglo

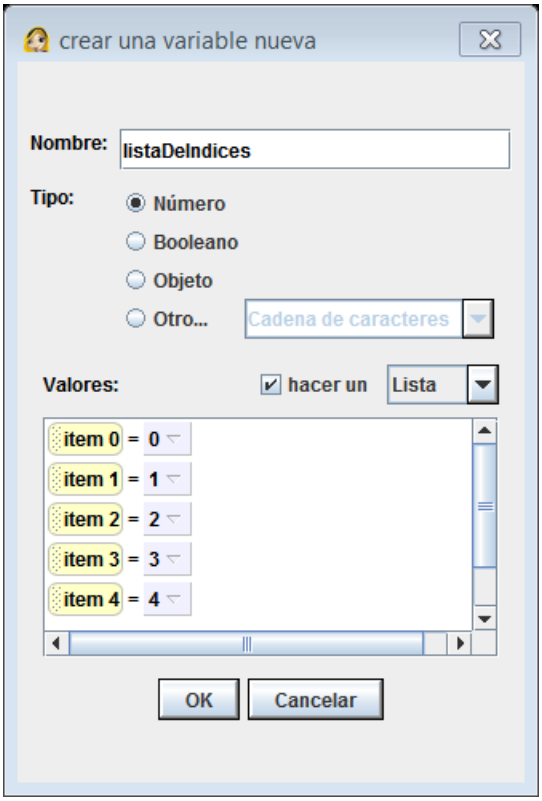
Una forma de trabajar con un arreglo consiste en **recorrer** todos los elementos que lo componen, **de a uno a la vez**, tal como lo hicimos con las listas. Para ilustrar el concepto de iteración, vamos a identificar a uno de los objetos que de forma aleatoria tenga una **caja que limita sus bordes (o bounding box)** encima. Para generarlo, basta con escribir este código:



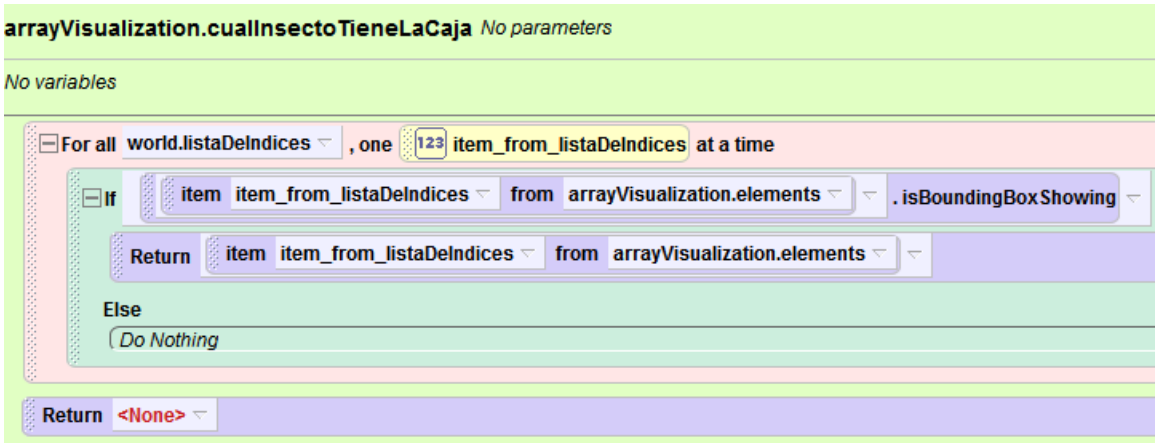
Al igual que en el ejemplo anterior, estamos pidiendo un número aleatorio que corresponderá con el índice del elemento que tenga una **caja** por encima. Este atributo se encuentra dentro del grupo de fichas **propiedades raramente usadas**. Esto es sólo para ejemplificar: usted puede usar cualquier otra.

Ahora, vamos a escribir una pregunta que devuelva el objeto que esta caja. Con una **lista**, podemos usar el bloque **para todos en orden** para comprobar si esta propiedad está activada o no. ¿Por qué trabajar con **para todos en orden** y una **lista**? Simplemente porque así resulta más fácil hacer el recorrido en el arreglo (también se puede con un **bucle**), y porque este bloque exige que se procese una lista, y no un arreglo. La lista que creamos **contendrá valores numéricos entre 0 y 4** (porque hasta ahí llegan los índices del arreglo), y le llamaremos **listaDeIndices**.

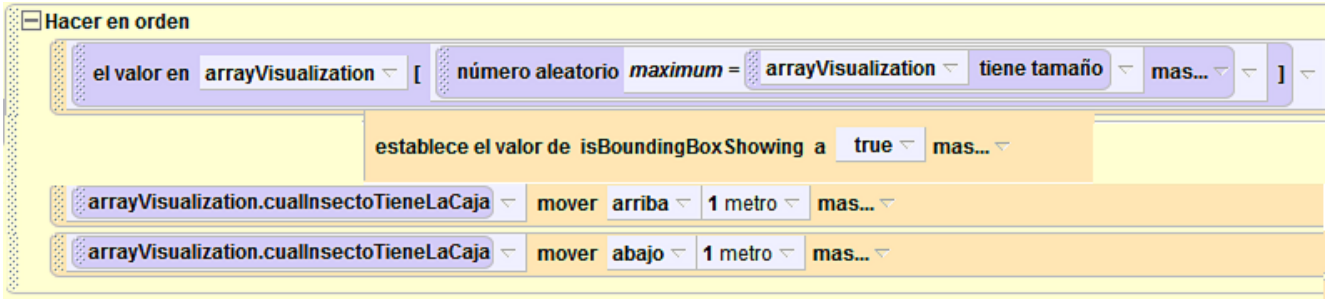
Figura 109. Creación de una lista de valores numéricos.



La instrucción **para todos, en orden** se utiliza para iterar a través de la lista de los números que contienen los índices del arreglo, de tal manera que cada uno de ellos representa la **dirección del elemento** dentro de la estructura. El código resultante se muestra a continuación. La función que creamos se llama **cualInsectoTieneLaCaja**, que fue definida para **ArrayVisualization**:



En la imagen adjunta se muestra un segmento de algún método cualquiera del programa que llama a la pregunta. El insecto al que se le asignó la caja se le pidió que se moviera hacia arriba y hacia abajo, como una señal que la iteración fue exitosa. Tenga en cuenta que tenemos que tener activada la caja a uno de ellos para que la función pueda retornar algo. De lo contrario, el programa se cae.



Búsqueda del elemento más alto en un arreglo

Como segundo ejemplo, supongamos que queremos encontrar aquel objeto de un arreglo que tenga la mayor altura de todos los que forman parte de él. Una posible solución a este problema es **suponer que el primer elemento es el más alto de todos**, de tal manera que **hasta el momento, el objeto de mayor altura es el que ocupa la posición cero**. Después podemos comparar este supuesto con el segundo, de tal manera que si el segundo es más alto que el primero, entonces ese elemento pasará a destronar al primero. A continuación, comparamos el tercero con el que **hasta el momento sea el «*nuevo elemento más alto*»**, que podría ser el primero o el segundo. Después, viene el cuarto, y así sucesivamente hasta llegar al final del arreglo, de tal manera que cuando eso ocurra vamos a tener la certeza absoluta de cuál es el más grande de todos. Lo que hemos hecho es pensar a través de una secuencia de pasos donde:

Marcamos un objeto de forma temporal como el más alto de todos.

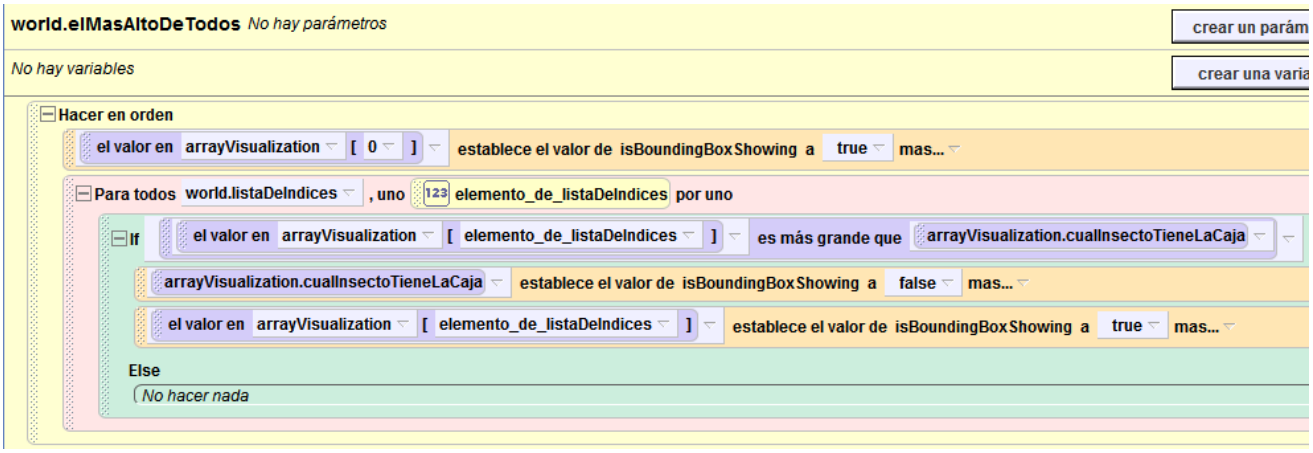
Veamos cómo quedaría un algoritmo o guión textual de lo que se acaba de exponer en los párrafos anteriores, suponiendo que el método que vamos a crear se llame **elMasAltoDeTodos**:

Tabla 19. Algoritmo para encontrar el mayor elemento de un arreglo, en forma de guión.

```
Método: elMasAltoDeTodos
1. Haga que el primer elemento del arreglo tenga la caja
delimitadora encima, en señal de ser el más alto de todos.
2. Iterar a través del arreglo:
     Si el i-ésimo elemento es más alto que el que se decía serlo
          a) Indicar que ese elemento es el nuevo más alto,
          poniéndole la caja alrededor del cuerpo.
          b) Hacer desaparecer la caja del otrora más alto.
```

El código de la Figura 110 muestra la implementación del algoritmo que se plantea en la Tabla 19:

Figura 110. Algoritmo para encontrar al elemento mayor dentro de un arreglo.



Esta solución, si bien correcta, es algo insatisfactoria porque tenemos que llamar en reiteradas oportunidades al método **cualInsectoTieneLaCaja**. El rectángulo de selección se utiliza como un marcador visual que efectivamente nos permite ver cuál es el más alto hasta ahora, pero cada vez hay que estar llamando al método **cualInsectoTieneLaCaja**. En la práctica, eso conlleva a que Alice tenga que iterar a través de la arreglo para indicar cuál de los objetos de insectos tiene la caja en su cuerpo.

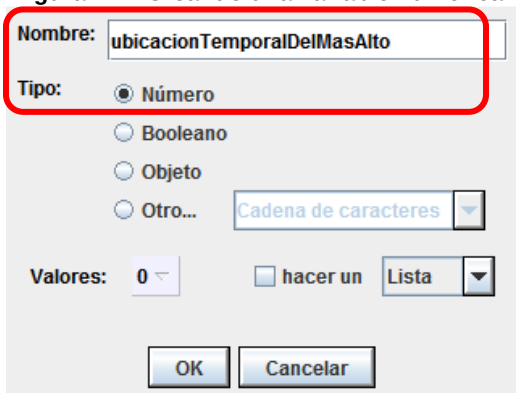
Agregar variables y replantear el método del elemento más alto

Si tan sólo pudiéramos **seguir la pista** de cuál insecto es el más alto hasta ahora, no tendríamos necesidad de llamar al método **cualInsectoTieneLaCaja** tantas veces. Acá es donde entran en juego las **variables, que corresponden a elementos de programación que permiten almacenar valores de manera temporal (a eso deben su nombre).** A diferencia de lo que hemos visto en Alice hasta ahora, las variables no son visibles porque no podemos acceder a su contenido, pero sí han aparecido en nuestros mundos virtuales, pero juegan un papel muy importante, que revisaremos en la tercera unidad.

Vamos a utilizar una variable para hacer un seguimiento de la ubicación del insecto más alto que hayamos encontrado hasta el momento. Gracias a esto eliminamos la necesidad de invocar al método **cualInsectoTieneLaCaja**, ya que la variable que usaremos se encargará de llevar **el registro del insecto de mayor estatura**. Para crear una variable, haga clic en el botón **crear una variable nueva**, que está situado en la parte superior derecha del método **insectoMasAlto** en el editor:

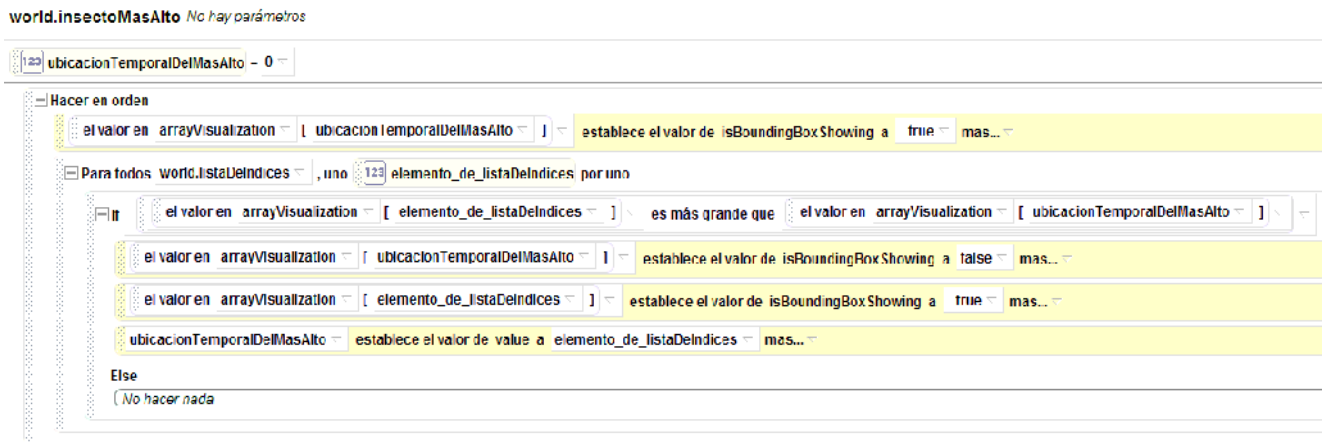
Encontrará un cuadro de diálogo donde se puede introducir el nombre de la variable y el tipo al que corresponde. En este ejemplo, le hemos llamado **ubicacionTemporalDelMasAlto**, cuyo tipo es un número de índice. Le dimos un **valor inicial** igual a 0, para partir diciendo que el elemento cero del arreglo es presumiblemente el más alto de todos. La figura Figura 111 da cuenta de esto:

Figura 111. Creando una variable numérica.



Gracias a la inclusión de esta variable, podemos reconstruir el método, como se muestra a continuación:

Figura 112. Implementación para encontrar el mayor elemento dentro de un arreglo, usando variables.

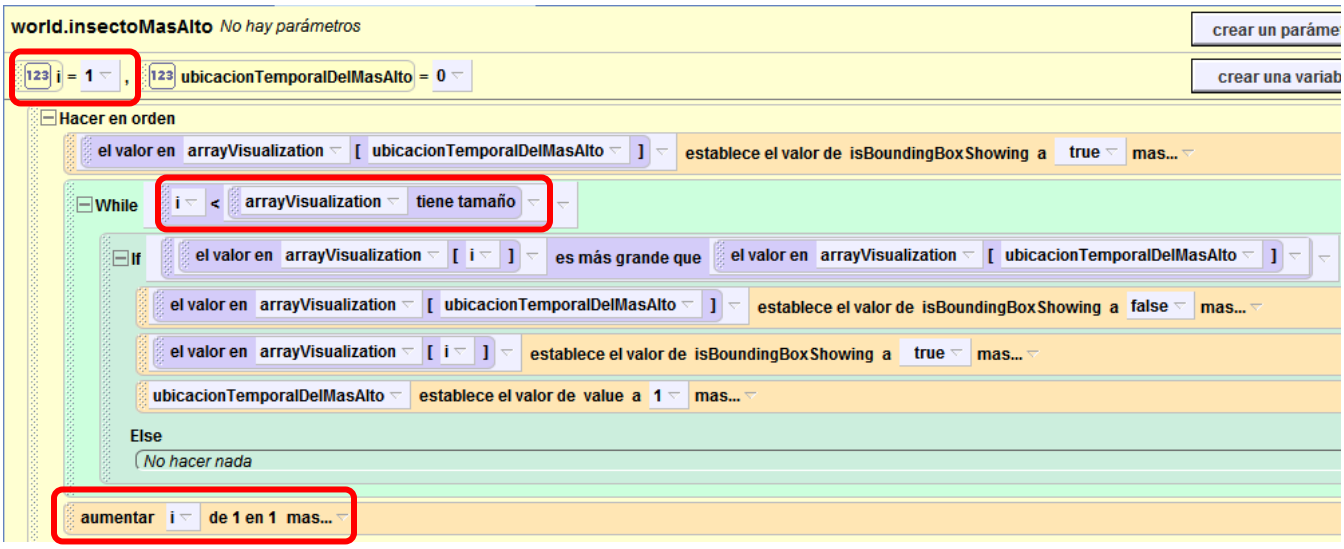


El valor de la variable **ubicacionTemporalDelMasAlto** se ajusta al momento de arrastrarla hacia el código, **asignarle** lo que vale **item_desde_ listaDeIndices**. El rectángulo de selección ya no es necesario.

Usando una variable como índice de localización

Podemos usar otra técnica para recorrer un arreglo, en lugar de los iteradores que empleamos hasta el momento. En su lugar, podemos trabajar con una **variable de tipo numérica** que podemos llamar **i**, para **representar la ubicación del índice** en el arreglo. Por lo general este nombre se escoge porque en la jerga computacional uno se refiere al **i-ésimo, j-ésimo ó k-ésimo** elemento de un arreglo. La variable **i** se usa con un **while** — podría haber sido también con un bucle finito — para realizar la misma tarea del **iterador, que se llama index**. El código resultante se muestra en la Figura 113:

Figura 113. Incorporación de índices para recorrer un arreglo.



Para empezar, la variable **i** se inicializa en 1, y la otra en cero. Al principio del bucle **while**, la condición de tipo booleana **i < tamaño del arreglo** tiene sentido, ya que es posible conocer la cantidad de elementos de esa estructura de antemano. La última declaración dentro del bucle es **incrementar i en 1**, por lo que al final de la primera iteración va a entrar al bucle con valor 2, momento en el cual se comprueba si esta **actualización del valor de i** hace que **siga siendo verdadero el hecho que i < tamaño del arreglo.**

Ejercicio

1 Mundo de insectos.

Cree un mundo que contenga un arreglo de 15 insectos, y modifique lo que se ha hecho hasta el momento en los siguientes aspectos:

1. Cambie la pregunta **cualInsectoTieneLaCaja** para que devuelva el índice del insecto que está en el rectángulo, en lugar del nombre. Escriba además el código para que el insecto se mueva hacia arriba y hacia abajo cuando se devuelva el índice.
2. Construya un algoritmo que permita encontrar al insecto menos angosto del grupo.
3. Construya una pregunta llamada **primerEscarabajo** que busque dentro del arreglo, y que devuelva la ubicación del primer escarabajo que haya. Asegúrese que haya al menos 3 de ellos. Una vez que retorne el valor, el bicho tiene que moverse hacia adelante 1 metro.
4. Escriba la pregunta, **ultimoEscarabajo**, que busque a través del arreglo y entregue la ubicación del último que haya. Un arreglo y devuelve la ubicación del último escarabajo en la arreglo Una vez que retorne el valor, el bicho tiene que moverse hacia atrás 1 metro.

Intercambio de elementos en un arreglo

La acción de cambiar el orden en que están dados los elementos dentro de un grupo se conoce en la jerga matemática como **permutación**. Por ejemplo, si tenemos los números 1,2 y 3, podríamos permutarlos de la siguiente manera:

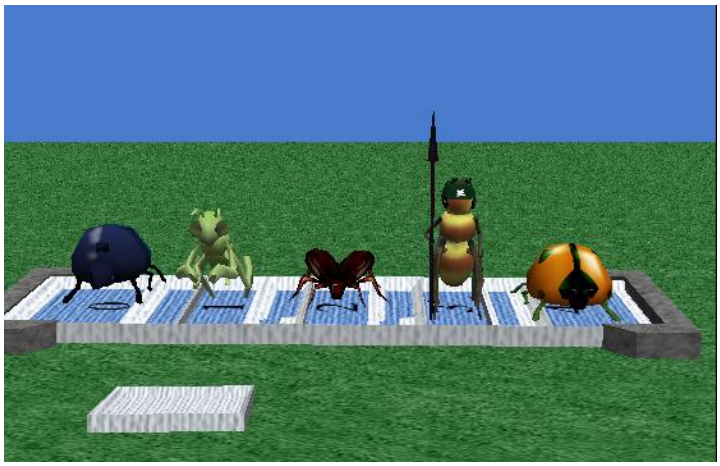
1,2,3
1,3,2
2,1,3
2,3,1
3,1,2
3,2,1

En programación, también podemos ilustrar este concepto a través del intercambio de elementos en un arreglo. Por ejemplo, intercambiemos el escarabajo

que está en la posición 0, con el mantis que está indexado con número 1. Una de las características que tiene un arreglo es que no se pueden colocar dos elementos en una misma posición (si fuera así, el problema sería muy sencillo de resolver). Tampoco podemos eliminar elementos del arreglo para hacer un intercambio, porque ese no es el espíritu de esta tarea. Por lo tanto, tendremos que hacer uso del concepto de **ubicación temporal**, que va a corresponder a un objeto que va a **albergar de manera transitoria** a un elemento, a la espera que el huésped quede reubicado en su nueva posición. El plan de ataque es el siguiente:

1. Ocupamos un objeto de almacenamiento temporal que guarde mientras tanto a uno de los elementos que se van a cambiar.
2. El segundo objeto se desplaza hasta el lugar que el primero deja vacante en el arreglo.
3. El que estaba en la ubicación transitoria se desplaza hasta el lugar que originalmente tenía el segundo.

Esta ubicación temporal la vamos a generar a través de un objeto que se llama **ObjectVisualization** que está en la misma galería de donde se obtiene **ArrayVisualization**, y que conoceremos como **cuadrado blanco**. La imagen adjunta muestra el arreglo inicial de los animales con el **cuadrado blanco que hará las veces de ubicación temporal**, que se ubica frente a la representación visual del arreglo con que vamos a trabajar.



Ahora que tenemos este espacio en blanco, podemos construir un método para hacer los tres pasos descritos anteriormente. Vamos a requerir de dos parámetros numéricos para identificar los índices de los elementos que vayamos a intercambiar (en este caso estamos trabajando con las posiciones 0 y 1, pero podemos generalizar el funcionamiento a más de una), que los vamos a denominar **primerIndice** y **segundoIndice**. Un guión para esto sería:

Método: permutación
Parámetros: primerIndice, segundoIndice
Hacer en orden:
1. Ubicar el elemento que está en la posición **primerIndice** en **ObjectVisualization.**
2. Ubicar el elemento que está en la posición **segundoIndice** en la posición dada por **primerIndice.**
3. Ubicar el elemento que está en **ObjectVisualization** en la posición dada por **segundoIndice.**

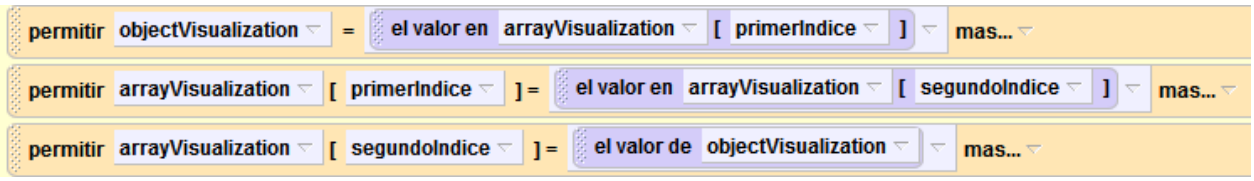
Lo único que nos va faltando es traducir el libreto en un programa. Una cosa bastante curiosa es que en ninguno de los tres pasos dados anteriormente hemos ocupado la instrucción **mover**. El término **ubicar** se usa para indicar que hay algo más

allá de un simple movimiento. Existe una instrucción en Alice que se llama **permitir** que se usa para simular el verbo **ubicar**, y que además se encarga de **actualizar la lista de elementos del arreglo para llevar un control de los cambios que se han hecho**. En primer lugar, vamos a reubicar el elemento que está en la posición **primerIndice** en el cuadrado blanco. Para esto, vamos a arrastrar la instrucción **permitir objectVisualization = item**, y en el menú emergente seleccionamos **arrayVisualization**:

Si esto lo dejamos tal como está, estaríamos tratando de colocar todo el contenido del arreglo en el **cuadrado blanco**, y claramente no es eso lo que queremos lograr. Tenemos que especificar exactamente cuál es la posición del arreglo que queremos pasarle. De acuerdo a nuestro algoritmo, se trata del índice dado por el parámetro **primerIndice**

Los corchetes **[ ]** se ocupan para guardar el nombre del índice que se esté empleando. Después, creamos las instrucciones para reubicar los elementos en el arreglo, tal como se ve en el código ya completado de la Figura 114:

Figura 114. Algoritmo para intercambiar elementos en un arreglo.



Ejercicios

1 Versión mejorada de la permutación.

Construya una versión mejorada del método **permutación** que permita recibir como parámetro un arreglo cualquiera, para que su uso sea más genérico. Además, tiene que **validar** los datos de entrada, según este criterio:

1. **Los índices no pueden ser iguales**. Si es así, entonces el programa tiene que permutar el primero con el último elemento.
2. **Los índices no pueden ser valores negativos, o ser mayores que el tamaño del arreglo**. Si es así, entonces el programa tiene que permutar el primero con el último elemento.

2 Todas las permutaciones posibles.

Construya un mundo con 4 objetos a su elección. Diseñe un programa que muestre todas las permutaciones que se puedan hacer. En cada iteración, los personajes involucrados tienen que decir en qué posición se encuentran actualmente, y al final tienen que hacer lo mismo con la posición de destino. ¿Cuántas iteraciones hizo su programa?

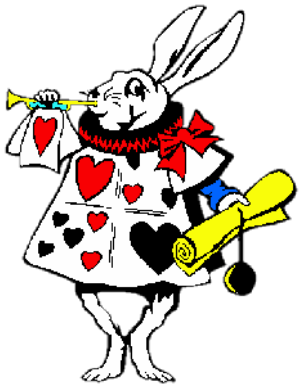
Resumen

El enfoque principal de esta parte estuvo centrado en los arreglos, que corresponden a estructuras de datos ampliamente utilizadas en el contexto de las Ciencias de la Computación. Estas construcciones permiten agrupar datos que sean de la misma naturaleza, y se pueden acceder a través de **índices**: ya sea a través de una iteración con un bloque **para todos en orden**, o bien por medio de algún **bucle finito o un while** que haga uso de una **variable** que se encargue de recorrer internamente esta estructura. Además, se dijo que el **tamaño** de un arreglo corresponde a la cantidad de elementos que guarda. También hemos aprendido a buscar objetos dentro de un arreglo que respondiera a alguna propiedad determinada, y a que era posible **intercambiar** dos elementos de posición, gracias a la utilización de una **estructura auxiliar** que nos servía para estos fines.

Conceptos importantes vistos en esta parte

- Un arreglo *es* una colección organizada de valores y objetos.
- Los elementos de un arreglo son todos del mismo tipo.
- Se puede **acceder** a un elemento de un arreglo a través de un **índice**, que determina la **dirección** del mismo.
- Podemos iterar a través de los arreglos, de a un elemento a la vez.
- Es posible valernos de una lista que guarde los índices del arreglo para poder hacer un recorrido con la estructura de programación **para todos en orden**.
- Una **variable** es una construcción de programa que permite almacenar valores de un cierto tipo.

- Las variables pueden ser utilizadas en conjunto con un **bucle o un while** para iterar a través de un arreglo.
- Es posible hacer operaciones con los objetos de un arreglo, como intercambiarlos, determinar cuál es el mayor, el menor, etc.



Tercera Parte

Variables

Esta parte vuelve a dar un vistazo a los objetos y a la herencia, que habíamos revisado en la segunda unidad cuando a la patinadora le enseñamos algunos pasos de baile, y que después guardamos con otro nombre para su posterior **reutilización** en otros proyectos. Vamos a combinar lo que hemos aprendido hasta el momento con el concepto de **variable** que enunciamos tímidamente en la parte anterior de esta unidad.

Herencia

En los programas y proyectos presentados en los capítulos anteriores hemos hecho uso de la técnica de diseño y creación de nuevos caracteres escribiendo nuevos métodos para ampliar la capacidad de un objeto existente. Luego, el personaje con sus nuevos métodos se guarda como uno nuevo, de tal manera que la creación **hereda todas las propiedades y los comportamientos de la original**, pero ahora está dotada con nuevos métodos que la clase madre no tiene definidos. En las Ciencias de la Computación, esto se conoce con el nombre de **herencia**. En esta sección, vamos a examinar la idea de una **variable** — conocidas también como **variables mutables** —, y la manera en que puede ser utilizada para apoyar la herencia y la creación de nuevos objetos.

Variables que afectan a los objetos

En el último capítulo, presentamos las variables como aquellas construcciones del lenguaje que permitían simplificar algunas tareas de iteración a través de arreglos. Se dice que son **mutables porque el valor que almacena va cambiando en tiempo de ejecución, mientras que lo que no cambia recibe el nombre de constante**. Desde la perspectiva de Alice, una variable corresponde a una propiedad de un objeto, o del mundo, las que pueden ser creadas para alguno en particular para ampliar sus prestaciones, tal como lo hacíamos con los métodos adicionales que se agregaron a la **PatinadoraInteligente**. De esta manera, si el personaje se guarda y se le pone un

nuevo nombre, se crea un nuevo tipo de objeto.

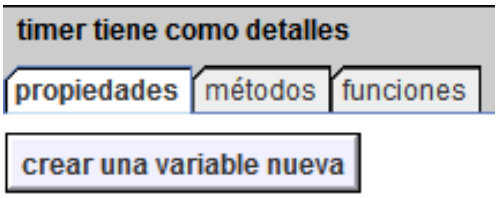
Ejemplo: Contador de tiempo

Un contador de tiempo o timer es uno de los elementos clásicos que aparece en los videojuegos, que se ocupa para llevar el registro de cuánto es lo que le queda de tiempo a un jugador durante una partida. Por ejemplo, en el fútbol tenemos un reloj que empieza con 45:00 minutos al inicio de cada tiempo reglamentario, de tal manera que a medida que el juego avanza, el valor va disminuyendo progresivamente hasta llegar a 00:00, momento en el cual finaliza uno de los dos tiempos. También, otro elemento que es bastante común son los marcadores de puntaje, que se van incrementando a medida que el jugador realiza acciones que tienen un efecto positivo en su partida. En el ejemplo que veremos ahora, se construirá un contador que avance hacia abajo, en segundos. La escena inicial que se muestra a la derecha contiene un timer que ha sido creado en base a un texto 3D.

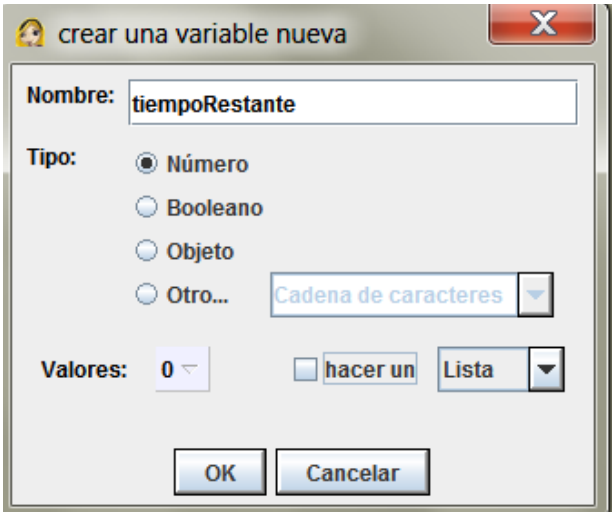
El valor inicial corresponde a una cadena de caracteres — o String — que vale **00:00**, para indicar que el contador está inactivo. Nuestra tarea va a consistir en darle otro valor inicial, para simular el tiempo restante en un juego (10 segundos), de tal manera que luego vaya hacia atrás contando lo que falte para llegar al final. El número de segundos que falten se va a desplegar y actualizar, a medida que transcurra el tiempo.

Creación de una variable

Una manera sencilla de poder llevar la cuenta de la cantidad de segundos consiste en usar una **variable que guarde la cantidad de segundos que actualmente falten para terminar**. Para esto, hacemos clic en **timer**, que está en el árbol de objetos. Luego, hacemos clic en las **propiedades**, y apretamos el botón **crear una nueva variable**, como se muestra a continuación:



A continuación, se abre un cuadro de diálogo que nos permitirá ingresar el nombre de la variable que queramos crear. En este ejemplo, le llamaremos **tiempoRestante**, que va a ser de **tipo numérico**, y le daremos un **valor inicial** igual a 0. Es importante destacar que **podríamos haberle dado cualquier valor, y que se eligió el cero por comodidad**.

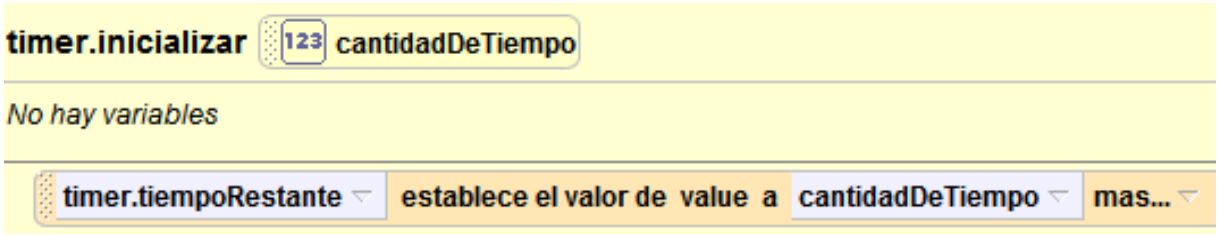


Debido a que el timer tiene un valor inicial de cero, sería útil crear un método parametrizado que se pueda usar para inicializar el valor del contador a algún valor arbitrario. Vamos a escribir un método llamado **inicializar**, cuyo algoritmo se describe en la Tabla 20:

Tabla 20. Algoritmo para inicializar un contador de tiempo.

Método: inicializar **Parámetro:** cantidadDeTiempo 1. La variable **tiempoRestante** se establece en un valor dado por **cantidadDeTiempo**

El código para implementar este método se muestra en la imagen adjunta. La instrucción para establecer un valor determinado se crea cuando arrastrando la variable **tiempoRestante** hacia el editor, y eligiendo **cantidadDeTiempo** como el valor:



El siguiente paso consiste en crear un método **contador** que haga que el contador vaya hacia abajo por segundos, y que al mismo tiempo actualice el mensaje que se lee a través del texto 3D. Un posible algoritmo es el que se muestra en la tabla Tabla 21:

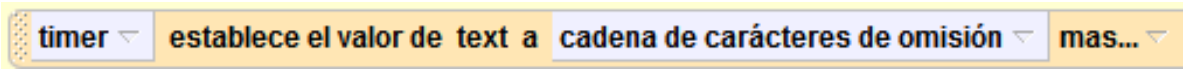
Tabla 21. Algoritmo para disminuir el valor de un contador de tiempo.

```
Método: contador
Hacer en orden:
      Mientras tiempoRestante sea mayor que cero
            Hacer en orden:
            1. Actualizar el texto 3D para mostrar el tiempo
      restante, en segundos.
            2. Disminuir en 1 el valor de tiempoRestante.
      Actualizar el texto 3D para mostrar 0 segundo.
```

A lo mejor le va a llamar la atención el contenido de la última línea del algoritmo. Piense lo siguiente: a medida que el bucle **while** se ejecuta una y otra vez, el valor de **tiempoRestante** va disminuyendo en una unidad por cada pasada. Sin embargo, si llega a cero, el **while** termina, y el cero no se despliega. Por lo tanto, fue necesario agregar esa línea para permitir que este valor se mostrara, en señal que el tiempo se acabó.

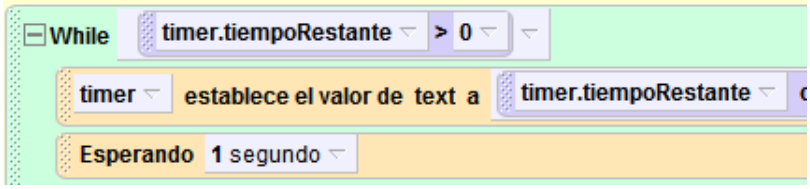
Ahora llegó el momento de traducir lo que tenemos arriba en el sistema Alice. En primer lugar, trasladamos el bucle **while** hacia el editor de texto, y dejamos establecido en la **sentencia condicional** que **tiempoRestante debe ser mayor que cero**:

Ahora nos falta completar el cuerpo del bucle **while**. Primero, agregamos un bloque **hacer en orden**, y después tenemos que arrastrar el **timer** para que despliegue el tiempo que falte. Para esto, debemos arrastrar la propiedad **texto** hacia el editor, y elegimos la opción **cadena de caracteres de omisión**:

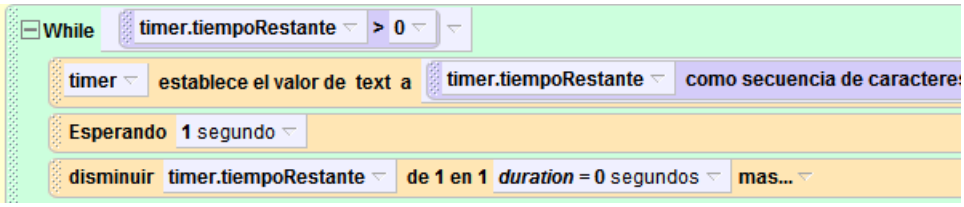


La **cadena de caracteres de omisión** es sólo un texto que se escribe, y que perfectamente se puede reemplazar. Lo que queremos es que **el texto 3D muestre el valor que actualmente tiene la variable tiempoRestante, que es de naturaleza numérica**. Esto se ha marcado con negrillas, porque es un verdadero problema: **no podemos mezclar texto con números** («*peras con peras, y manzanas con manzanas*»). Entonces, lo que necesitamos hacer es **una conversión de número a cadena de caracteres**, para que el valor se pueda mostrar correctamente. Para esto, podemos usar una **función de mundo** que se llama ***what* como secuencia de caracteres(string)**, que vamos a arrastrar hacia el editor, y la soltamos encima de donde dice **cadena de caracteres de omisión**. Cuando eso ocurra, elegimos como parámetro **tiempoRestante**, como se muestra a continuación:

Tenga en cuenta que la duración de la instrucción es igual a **cero segundo**, **para permitir que la actualización del texto 3D se ejecute de manera instantánea**. Ahora tenemos que hacer que efectivamente pase un segundo para que el texto se vuelva a modificar. Podemos usar una instrucción llamada **Esperando**, que le podemos dar el valor de 1 segundo:

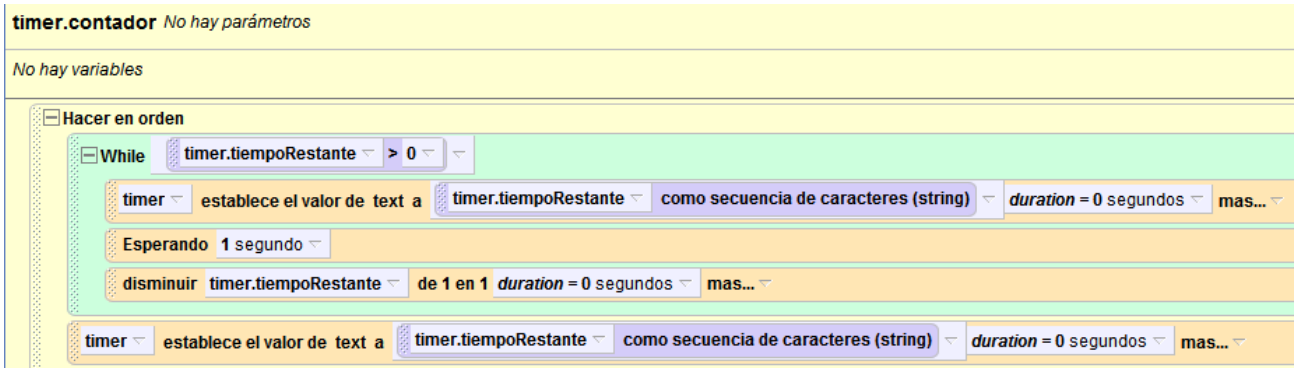


Después, tenemos que disminuir en 1 el valor de **tiempoRestante**. Para esto, arrastramos la variable hacia el editor de código, y pedimos que el valor baje en una unidad. Tenga presente que la duración de esto tiene que ser igual a cero segundo, para simular que todo se hace de manera instantánea:

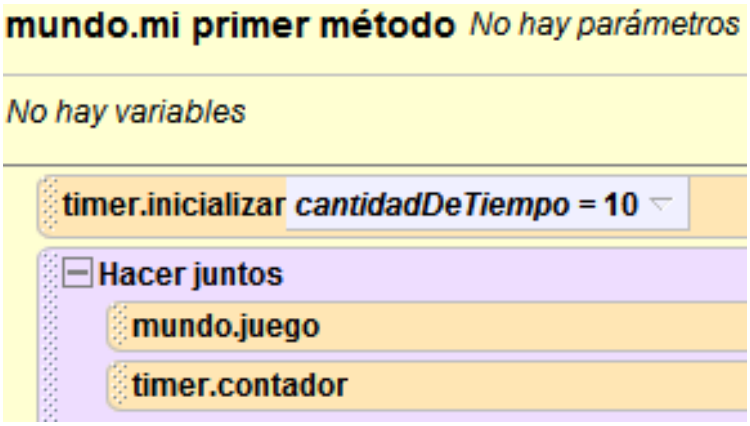


Lo último que nos queda es mostrar el momento en que la variable **tiempoRestante** sea igual a cero. Como ya se comentó, esto se tiene que hacer **fuera del while**, ya que ese bucle se ejecuta **mientras tiempoRestante** sea mayor que cero: si es igual, sale. El código del método se muestra completo en la Figura 115:

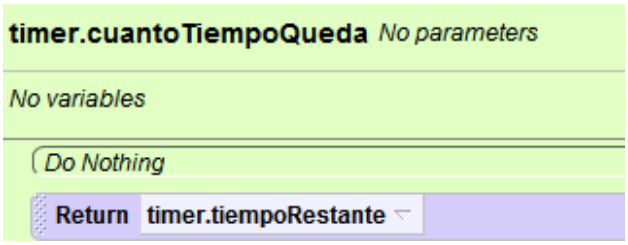
Figura 115. Algoritmo para implementar un contador de tiempo sencillo.



Para hacer una demostración de cómo funciona esto, en el método **mi primer método**, inicializamos el contador en 10 segundos, pasando 10 como el argumento de **tiempoRestante**. Después, usamos un bloque **hacer juntos** para que al mismo tiempo se ejecute el **timer** con un juego, que estará escrito con un método llamado **juego**:



El método **juego** necesita saber cuánto tiempo queda en el reloj, y para esto creamos una función **cuantoTiempoQueda** para el **timer**, que retorna este valor:



De esta manera, podemos escribir el método **juego**, para permitir el curso de las acciones **mientras** aún quede algo de tiempo disponible. En este caso, vamos a hacer que el pollo gire a la derecha una revolución. Es cierto: el juego es bastante fome, pero al menos sirve para ilustrar la idea de contador. Siéntase libre de hacer las modificaciones que quiera, para que sea algo más divertido.

Ejercicios

1 Conmutador.

Algunos ejemplos y ejercicios de este apunte han utilizado un interruptor como parte de un mundo interactivo. Varios de ellos se muestran a continuación. Un objeto interruptor le permite al usuario hacer un clic con el ratón para subir o bajar la palanca. La idea es crear el evento asociado con subir o bajar el interruptor cuando el usuario haga clic en la palanca.

Un evento de clic de ratón nos dice que el usuario ha presionado sobre la palanca, pero no brinda información sobre la posición actual que tenga la palanca conmutadora: si está en encendido o en apagado.

Dos técnicas se pueden utilizar para resolver este problema. La primera es colocar una esfera invisible en el extremo de la palanca, y luego ver si se encuentra por encima o por debajo del nivel medio del interruptor, de tal manera que su posición es inversa al estado del conmutador. La segunda técnica consiste en añadir una variable booleana en las propiedades del interruptor, de tal manera que sea igual a **verdadera** siempre y cuando el interruptor esté en posición de **encendido**, o **falsa** en caso contrario.

Se pide que construya un mundo que contenga un interruptor y otro objeto (una ampolleta por ejemplo), de tal manera que el estado del interruptor genere un efecto sobre el otro objeto que usted determine.

2 Conmutadores Binarios.

La electrónica digital se basa en el uso de altas y bajas intensidades para representar valores binarios. Una corriente alta que fluye a través de un circuito representa el dígito 1 y una corriente baja representa el dígito 0. Así, un sólo circuito representa un único dígito binario (**bit**). De esta manera, si juntamos 8 circuitos, formamos **8 bits**, que equivalen a **1 byte**.

Vamos a construir un programa que permita convertir un número escrito en base 10 — como los conocemos hasta ahora — a base 2, que sólo admite el 0 y el 1 para representaciones numéricas Un número escrito en base 10 se crea con los dígitos del 0 al 9. Por ejemplo: 5932 podemos escribirlo como:

5000+900+30+2

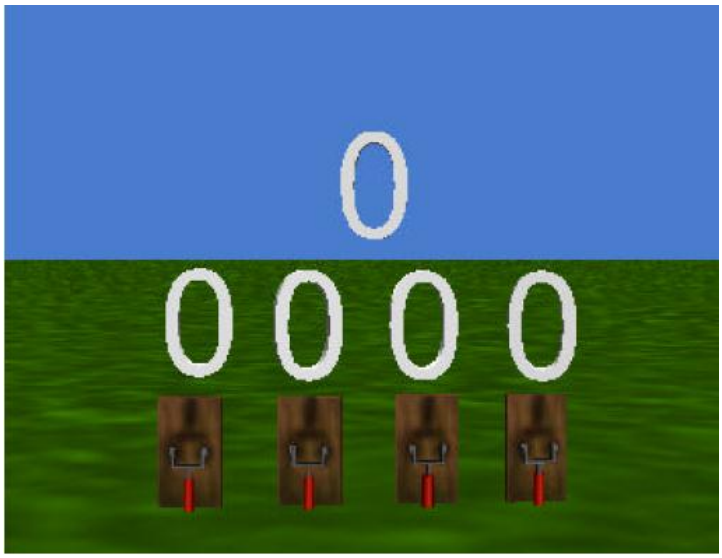
Que equivale a decir $\mathbf{5 * 10^3 + 9 * 10^2 + 3 * 10^1 + 2 * 10^0.}$

En base 2 ocurre lo mismo: si tenemos el número 1101, podemos representarlo como:

$$\mathbf{1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 8+4+1=13}$$

En lugar de multiplicar por 10, se hace por 2, y los exponentes que acompañan a las potencias parten desde el cero, en sentido de derecha a izquierda.

Cree un mundo con cuatro interruptores, cada uno con una variable booleana de encendido o apagado. Por encima de cada interruptor, inserte un texto 3D en la que se muestre el número 0. Cuando haga eso con los 4 interruptores, vuelva a colocar otro texto 3D encima de los cuatro ceros, como se muestra a continuación.



Inicialmente, todos los interruptores están apagados. Diseñe un evento que llame a uno o varios métodos, de tal manera que cuando el usuario baje una palanca, el conmutador quede en cero, y en 1 cuando la suba. Además, de manera simultánea los textos que están inmediatamente encima de los interruptores deben cambiar su valor

cuando el usuario haga clic en alguno de los conmutadores, y también el texto que está en la parte superior debe actualizar su estado para mostrar la representación en base 10 de lo que el usuario ha ido generando gracias a la interacción con los aparatos. . Crear un evento- método de controlador para ajustar el texto 3D por encima de un interruptor en la posición 0 cuando el interruptor este desactivado (la palanca hacia abajo) o a 1 cuando el interruptor está encendido (palanca hacia arriba). Puede usar todo lo que estime conveniente para resolverlo.

Ayuda. Puede ocupar esta fórmula:

(a*8)+(b*4)+(c*2)+(d*1)

Donde a, b, c y d representan a los 0s y 1s en su correcta posición, de acuerdo a la columna donde se encuentren. En el grupo de preguntas del mundo bajo la ficha **string**, va a encontrar la función **<what> as a string**. La utilidad que presenta es que puede convertir un número en una cadena que pueda ser mostrada a través del objeto de texto 3D.

3 Suma de números.

Construya un programa que le pida al usuario un número entero **n** cualquiera, de tal manera que muestre un texto 3D el valor de la suma de todos ellos. La aplicación tiene que funcionar mientras el número ingresado sea negativo.

Resumen

En esta parte, hemos trabajado con variables que operan a nivel de personajes, que se utilizan para almacenar información de algún objeto en particular, y que pasa a determinar parte del **estado** del mismo. Se les denomina también **variables mutables** porque su valor **cambia en tiempo de ejecución**. Al igual como ocurre con los métodos, un objeto que incorpore variables adicionales se puede guardar con otro nombre para ser **reutilizado** en otro proyecto de desarrollo.

Conceptos importantes vistos en el capítulo

- Las variables son útiles para extender las prestaciones de algún objeto en particular.

- Así como ocurre con los métodos de personaje, las variables también se pueden almacenar en nuevos objetos, para que éstos se puedan ocupar en otros mundos virtuales.

Orientaciones Pedagógicas

Para conseguir los aprendizajes esperados, es necesario que los estudiantes tengan claridad con respecto a los siguientes puntos:

- Una lista es una manera muy conveniente para organizar objetos.
- Las estructuras repetitivas se ocupan para poder iterar a través de los elementos que conforman una lista.
- Algún elemento en particular puede ser accesado y manipulado como si fuera un objeto cualquiera.
- Es posible buscar objetos dentro de una lista, a partir de ciertas condiciones específicas, empleando estructuras condicionales.

Para que estos entendimientos se puedan llevar a cabo de manera exitosa, se recomienda que los estudiantes desarrollen en las sesiones de práctica no sólo los problemas propuestos al final de la unidad, sino también que reflexionen en torno a estas preguntas, que se pueden distribuir en el tiempo que el profesor estime conveniente:

- ¿Por qué en algunos momentos una lista resultara ser el tipo de dato más apropiado para trabajar?
- ¿Qué clase de problemas se pueden abordar con un tipo de dato como las listas?
- ¿En qué se diferencia una lista de un arreglo?

Unidad 5

Transición a Java

Esta unidad tiene como propósito hacer una transición desde Alice hacia otros lenguajes de programación que se usan en cursos más avanzados, como el caso de Java, a través de un conjunto de analogías entre ambas herramientas.

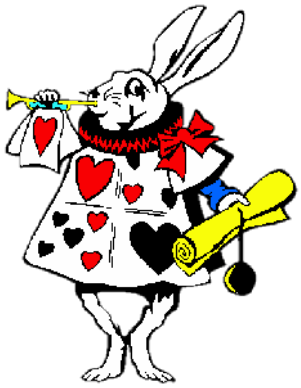
Sentido educativo de la unidad

Aprendizajes esperados.

- Reconocer las diferencias sintácticas entre Alice y Java.
- Identificar la estructura que tienen las clases, objetos y métodos en Java.
- Cómo se construye una clase en Java, y del por qué en Alice no es trivial.
- Conocer el concepto de encapsulamiento.
- Identificar cómo se usan los bloques clásicos de programación en Java.

Contenidos.

- Sintaxis de lenguajes de programación.
- Clases y objetos en Java.
- Instanciación de objetos, métodos constructores.
- Bloques de programación en Java



Primera Parte

Sintaxis

Esta unidad tiene como propósito brindar una transición desde Alice hasta otros lenguajes de programación, como **Java, C++ o C#**. Las principales diferencias entre Alice y otras herramientas se basan en la forma en cómo se presentan los conceptos tradicionales de programación. Lenguajes como los antes mencionados están pensados para el desarrollo de aplicaciones de **propósito general**, como pueden ser: sistemas **de información**, **gestión de comercio electrónico**, **protocolos de transmisión de datos a través de una red**, **juegos** o **software destinado a la investigación científica**, entre otros. En cambio, Alice se ha diseñado para lograr dos objetivos muy diferentes:

1. Proporcionar un entorno de programación de animaciones gráficas y videojuegos interactivos en el espacio tridimensional.
2. Servir de herramienta para que el estudiante aprenda a programar de una manera divertida y completamente alternativa a cómo se hace bajo el formato tradicional, que contempla desde el primer momento la inclusión de herramientas usadas por desarrolladores profesionales, como **NetBeans**, **Eclipse**, o **Microsoft Visual Studio**, por nombrar algunos

Lo que se cree es que muchas personas a aprendan a programar con Alice seguirán estudiando otros entornos de desarrollo, considerando que esta herramienta está pensada para que usted **desarrolle una manera de pensar que se base en el razonamiento algorítmico y sistémico**. Con esto en mente, el equipo desarrollador de Alice se ha preocupado para mantener un alto nivel de coherencia con otros lenguajes orientados a objetos.

Esta primera parte habla de la **sintaxis**. Alice ofrece una opción para poder visualizar la sintaxis de un programa, de tal manera que usted pueda ver cómo se van aplicando las distintas normas de escritura (llaves, paréntesis, comas, punto y comas, etc.) del proyecto que está realizando, como si estuviera en otro lenguaje de programación. En este caso, nos preocuparemos de hacer un paralelo entre la manera de escribir algo en Alice, y de cómo quedaría escrito en **Java.**

5.1 Comparación entre la sintaxis de Alice con la de Java

La **sintaxis** es un término técnico que se usa para **especificar la manera en cómo se deben escribir las instrucciones en algún lenguaje de programación**. Por ejemplo, considere la siguiente instrucción escrita en **lenguaje natural**, que es el que empleamos en nuestro diario vivir:

La rana se mueve hacia adelante 1 metro.

Una de las tantas reglas de sintaxis nos dice que una oración empieza con la letra de la primera palabra en mayúscula, y que debe terminar con un punto, o un signo de interrogación o de exclamación. Si es cualquiera de estos casos, entonces esos símbolos tienen que estar abiertos y cerrados, tanto al inicio como al final. Además, se incorporan otras que nos indica el **orden** en que se deben escribir las ideas. Por ejemplo, no podríamos haber escrito

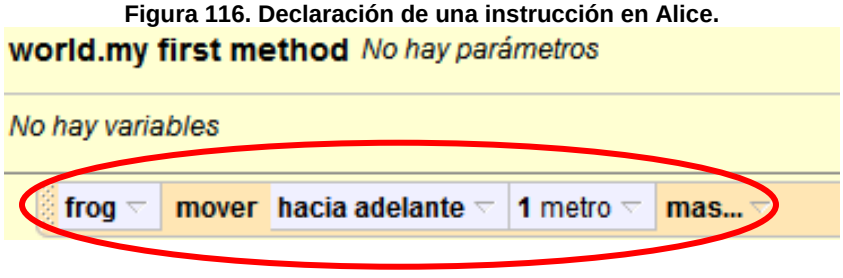
La se mueve rana hacia 1 metro adelante.

Porque el contenido de lo que se quería decir **pierde el sentido original**. Podríamos pensar que «*con un poco de buena voluntad*» podríamos hacer el esfuerzo de entender lo que se estaba tratando de decir — y que de hecho, se puede —, pero lamentablemente estas herramientas computaciones **son muy estrictas en este sentido**, ya que a veces un paréntesis mal puesto, o algún punto y coma que no se haya escrito da motivo suficiente para que el programa que estemos escribiendo no funcione.

Entonces, al igual como ocurre con los lenguajes naturales, los de programación también tienen **reglas de sintaxis**, ya que las declaraciones deben comenzar con ciertos tipos de palabras, continuar con otras, y así sucesivamente. Por otra parte, los signos de puntuación como comas, puntos, punto y coma y paréntesis están repartidos por todo el código que usted tenga que programar. Es importante comprender que los lenguajes presentan diferencias en cuanto a sus reglas sintácticas, que se caracterizan por ser **inflexibles**. Por ejemplo, si algún fragmento de código no cumples con estas normativas, el **compilador** — que es la herramienta que se emplea entre otras cosas para hacer las **validaciones** — simplemente rechaza el código, y usted no lo podrá ejecutar **mientras** no solucione todos los problemas pendientes.

Las reglas de sintaxis a menudo ayudan a generar sentimientos de frustración entre las personas que están iniciando sus aprendizajes en el ámbito de la programación. En Alice, el editor para construir las instrucciones está diseñado para aliviarle a usted esta preocupación. La Figura 116 muestra un ejemplo de declaración en Alice.

Figura 116. Declaración de una instrucción en Alice.



La regla de sintaxis que se ha usado para determinar el orden de las palabras que componen las sentencias en Alice es la siguiente:

```
<nombre_del_objeto> <nombre_de_método> <parámetros>
```

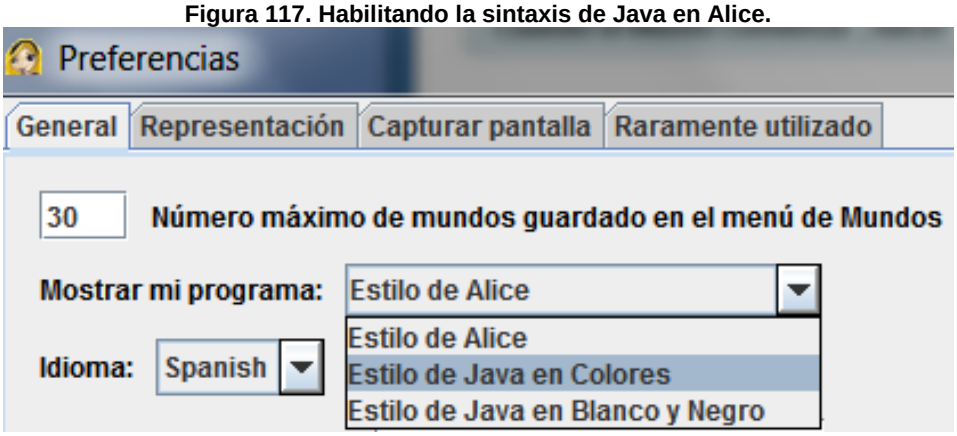
En este ejemplo: **<nombre_del_objeto>** se refiere a la **rana**; el **> <nombre_de_método>** es **mover**, y los parámetros son **hacia adelante**, y **1 metro**. Tenga en cuenta que el **más** que está al final de la declaración **sirve para que usted pueda seguir completando los otros parámetros opcionales que el método le ofrece**, que en más de una oportunidad hemos utilizado en el apunte. Estas opciones adicionales incluyen características como la **duración** o el **estilo** con que se desarrolle la animación. Están pensados para adoptar **valores por defecto**, que corresponden a datos que se asumen de antemano, como por ejemplo la duración de 1 segundo.

El método de arrastrar y soltar de Alice fue diseñado para trabajar con las reglas de la sintaxis de forma automática, de tal manera que el orden de las palabras genere **sentencias que sean fácilmente entendibles por usted**. En Ciencias de la Computación se dice que el editor es **sensible a la sintaxis**.

De manera predeterminada, el **editor deja de lado los signos de puntuación**, y hace **cambios de color** cuando se incluyen parámetros adicionales, o se agregan ciertas estructuras de programación como **if/else**, **bucles, while** o alguna **llamada a función**. Usted habrá notado que en algún instante, **no era posible arrastrar algún elemento hacia el editor**: esto se debe a que ciertos comandos **no se pueden ejecutar en cualquier parte**. El poder del editor de sintaxis permite que el programador principiante se concentre en cómo resolver los problemas y escribir los programas, sin que tenga que lidiar con los detalles sintácticos propiamente tales.

En la transición de Alice a otros lenguajes de programación, es posible que desee ver todos los signos de puntuación. Para esto, usted puede cambiar **el estilo de sintaxis** desde el **menú preferencias**, donde deberá marcar la opción relacionada con la **sintaxis de Java**, como se muestra en la Figura 117:

Figura 117. Habilitando la sintaxis de Java en Alice.



Decimos que esta característica de Alice actúa como un **interruptor** que permite **activar o desactivar** un determinado estilo sintáctico, que se ajuste mejor a sus preferencias. Para volver a la modalidad tradicional, basta con seleccionar el **estilo de Alice**

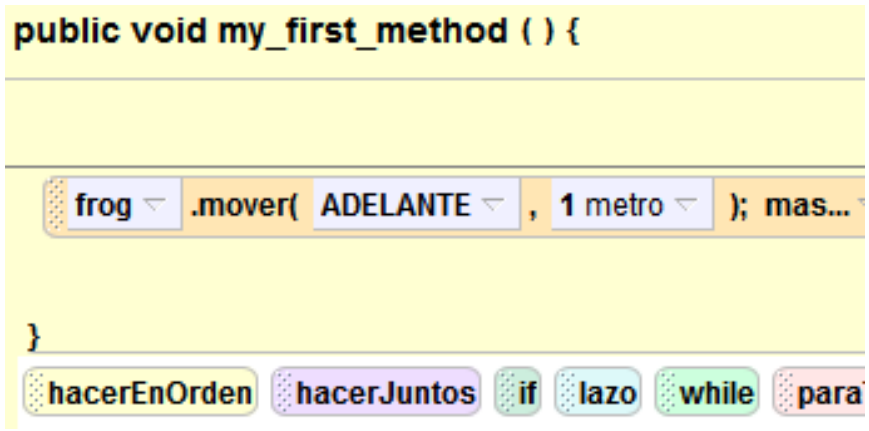
Una vez haya hecho esto, las declaraciones se vuelven mucho más detalladas, e incluyen los símbolos de puntuación estándar que aparecen en otros lenguajes orientados a objetos. Vea cómo queda la declaración de pedir que la rana avance hacia adelante 1 metro, que se exhibe en la Figura 118:

Figura 118. Cómo se ve en Java un código escrito en Alice.

Aunque cambie la sintaxis, **el significado de la sentencia sigue siendo el mismo**. Notemos por ejemplo que un **punto** se ha colocado entre el objeto rana, y el nombre del método que se está invocando. Con respecto a los **parámetros**, podemos ver que se escriben **entre paréntesis**, y que **cada uno de ellos se separa por una coma**. Además, **toda la instrucción finaliza con un punto y coma**. También, el **más** actúa como una **etiqueta** del editor que permite seleccionar parámetros, pero no es parte de ella, a no ser que usted seleccione algo de ahí. Hasta el momento, encontramos que en **Java**, una típica orden como esa se escribe bajo esta regla:

```
<nombre_del_objeto>.<Nombre_de_método>(<parámetro>,<parámetro>);
```

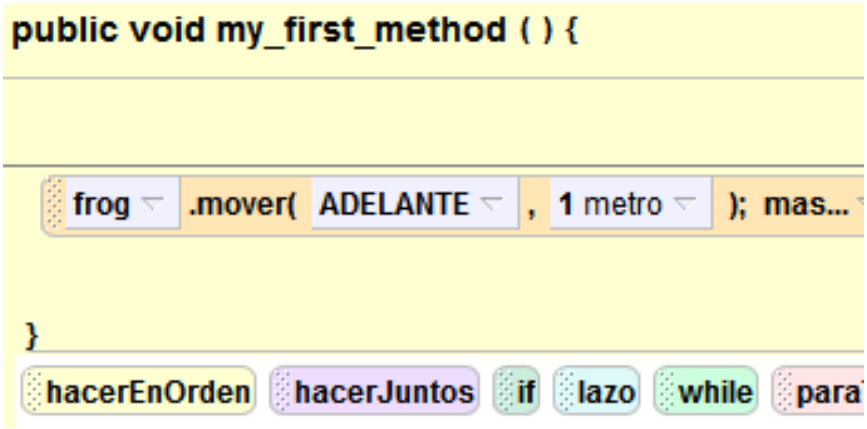
Si mira más de cerca la imagen, notará que algunos de los elementos de la interfaz también han cambiado. Por ejemplo, antes decía **Esperar**, ahora aparece **esperar(duración)**, o antes decía **Imprimir**, y ahora dice **imprimir(texto, objeto)**, lo que se debe a que cada una de estas operaciones requiere un parámetro. Otro cambio llamativo se da al principio del método:

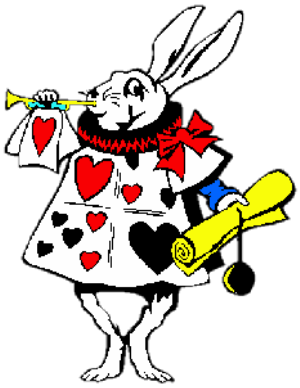


La sentencia **public void mi_primer_metodo() {** especifica información al sistema sobre cómo se debe ejecutar. En particular, indica el **tipo de dato** que se le tenga que pasar, así como lo que **tenga que devolver**. En muchos lenguajes orientados a objetos, el primer método que se ejecuta corresponde al **principal**, que en Alice se conoce como **mi primer método**, que tiene la característica de **ser público** y que **no tenga efecto**. Público significa que puede ser llamado desde cualquier lugar, mientras que la palabra **void** significa que **no hay devolución de datos, lo que no significa que el método no realice acciones** (¿recuerda la diferencia que hay entre un método y una función?). En Alice, **todos los métodos son públicos y son de tipo void**. Por último, note que dos llaves se han agregado para **encerrar las declaraciones del**

método, como se muestra en la Figura 119. Esto es coherente con los lenguajes orientados a objetos, ya que **un método es un bloque de código**, y éstos **siempre van delimitados por llaves**, como una forma de indicar **cuándo empieza y cuándo termina**.

Figura 119. Delimitación del principio y del final de un método.





Segunda Parte

Clases y Objetos

5.2 Clases

Objetos y Clases

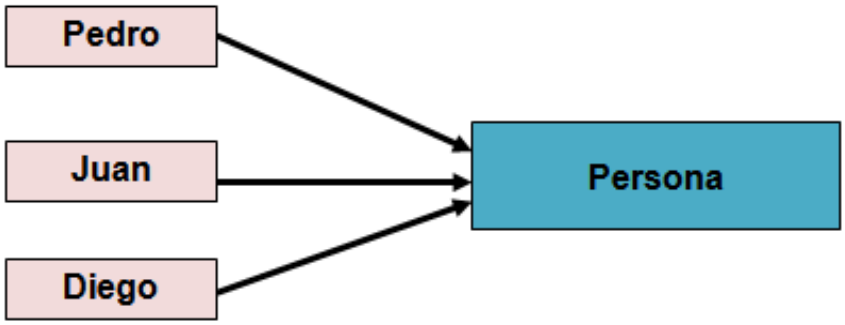
Los conceptos fundamentales que emergen en el ámbito de la programación orientada a objetos son los **objetos** y las **clases**. De forma intuitiva, un **objeto** es todo lo que **pueda ser identificado como único dentro de otras cosas**, y se identifican por tener: (1) **nombre**, (2) alguna cantidad de **propiedades**; y (3) la **capacidad de realizar ciertas acciones** o llevar a cabo tareas específicas. En Alice un objeto se representa como una imagen en 3D de un conejo, una persona, un auto, o alguna

otra selección que usted pueda hacer desde la galería. En J**ava, C++ y C#**, un objeto se representa por un nombre dentro del código del programa, y depende de la **abstracción mental del programador**.

Objetos

En Alice, como en los lenguajes orientados a objetos, **los objetos** (valga la redundancia) **son instancias de clases**. Como un ejemplo, en la Figura 120 tenemos a la clase **Persona**, y tanto Juan, Pedro y Diego son **instancias o ejemplares** de esa clase.

Figura 120. Ejemplo de tres objetos de una misma clase.



En la Figura 121, Hugo, Paco y Luis son **instancias** de la clase **Pato**. Tenga en cuenta que se les ha cambiado el nombre a los patos. Cuando en Alice hay más de una instancia para una clase, los objetos se renombran con un número correlativo a su derecha (pato2, pato3, pato4, etc.)

Figura 121. Tres instancias de una clase representadas en Alice.

También, **todos los objetos de la misma clase comparten algunas características comunes**. En el caso de una persona, los objetos que se deriven de ella tienen **atributos** como nombre, dos piernas, dos brazos, altura, o algún color de

ojos. Además, todos ellos pueden realizar ciertos **métodos o acciones**, como por ejemplo caminar, saltar, etc.

Mientras que cada objeto pertenece a una clase, es posible que esta instancia **redefina algunas propiedades** que **hereda** de la **clase madre**. Por ejemplo, el objeto **Luis** es de color verde, y **Paco** es más pequeño que los demás.

Las clases

En Alice, **las clases están predefinidas como modelos 3D** que usted puede encontrar en la galería. A su vez, estos modelos están agrupados en diferentes **categorías**, como animales, personas, edificios, fantasía, etc. La Figura 122 muestra algunas de las clases del modelo de la colección de animales.

Figura 122. Agrupación de clases en Alice.

Al igual que en las galerías de Alice, los demás lenguajes de programación también proporcionan clases predefinidas, que se guardan en **bibliotecas**, que se encargan de incluir un **repertorio de funciones de entrada y salida, matemáticas, para manipular cadenas de caracteres, hacer dibujos**, etc. Para utilizar algunas de las clases que provea una biblioteca, en el código de un programa se suele escribir una instrucción del tipo **import** o **#include**. Esta última es la que se emplea en **C++**: veamos un pequeño ejemplo:

```
#include <iostream.h>
#include "miclase.h"
```

Los corchetes angulares **<** y **>** le indican al **compilador de C++** que la biblioteca de las clases **iostream.h** y **miclase.h** deben ser incluídas. Las comillas se utilizan para llamar a una **clase que ha sido construida por el programador, o que se ha obtenido desde una fuente que no provee directamente el lenguaje**.

En **Java**, la sentencia `#include` tiene un equivalente con el comando **import**, que significa importar. Por ejemplo, la declaración de clase en **Java**

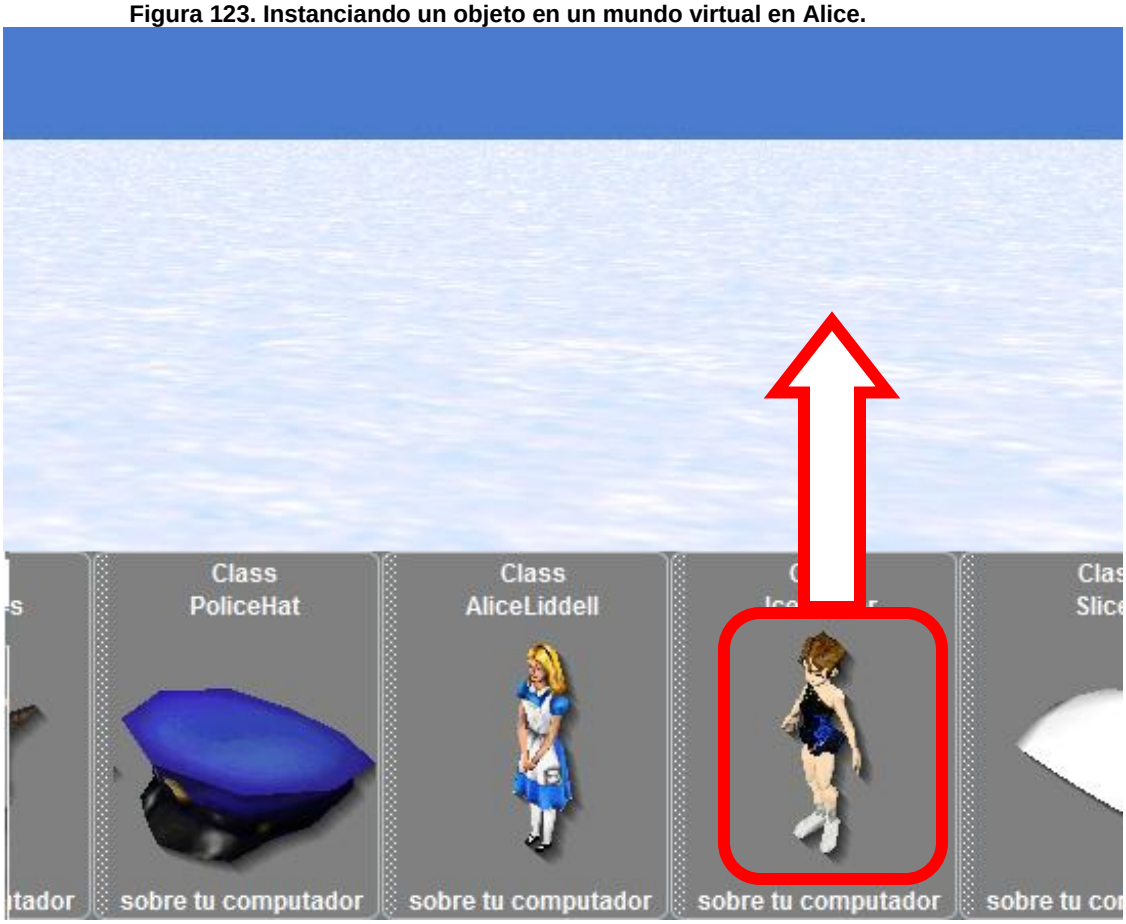
```
import java.applet.*
```

Sirve para llamar a la clase **Applet**, que se utiliza para la construcción de aplicaciones web, El uso del símbolo ***** es para indicar que **también vamos a considerar todas las subclases que se desprendan de applet**.

Instanciación de clases

En Alice, cuando usted selecciona una imagen desde la galería y la arrastra hacia el mundo, está **produciendo una instanciación de la clase**, que se materializa en un **objeto que va a poblar el mundo virtual que construye**. Por ejemplo, instanciar la clase **Muñeco de nieve** se instancia arrastrando el ejemplar hacia la escena principal, o bien cuando usted selecciona la opción **crear una instancia**, como se muestra en la Figura 123:

Figura 123. Instanciando un objeto en un mundo virtual en Alice.



Cuando se arrastra, Alice le otorga un nombre provisional que puede ser modificado directamente desde el árbol de objetos. En **C++ o Java**, este proceso de la creación del objeto se denomina a veces **nuevo objeto**. Una diferencia importante de Alice con respecto a **C++ y Java** es que todos **los objetos en Alice deben construirse desde el principio**, antes de que el programa se ponga en marcha. En otras palabras, los objetos no se pueden construir o destruir dinámicamente durante el tiempo de ejecución.

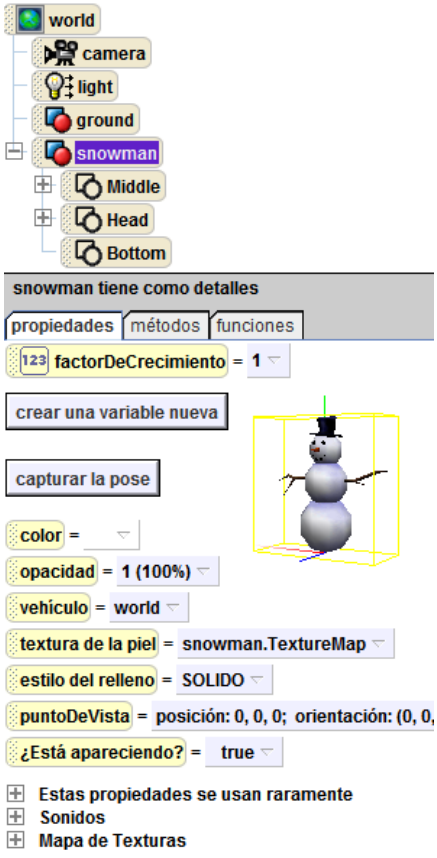
Entonces ¿cómo un objeto se crea en otro lenguaje de programación? Veamos algún código para definir una clase **Muñeco de nieve**, que se muestra en la Tabla 22. Esta clase **Muñeco de nieve** la vamos a construir con propiedades como la altura y el grosor. Si podemos hacerlo con dos, es fácil extender a más propiedades. En las líneas 4 a la 7 se muestra el **método muñecoDeNieve**, que es de tipo **constructor**. En la terminología de a programación orientada a objetos, un **método constructor** es aquel que se invoca **para crear un nuevo objeto de tipo Muñeco de nieve**. En este ejemplo, **muñeco1** es un tipo de muñeco de nieve, y que se ha creado gracias a una

llamada al método constructor de la clase, que podemos encontrar en la primera línea del método **main**. En Java, el método **main es el primer método que se ejecuta cuando se inicia el programa**, y equivale al método **mi primer método** que se ocupa en Alice. Por lo tanto, lo primero que sucede en tiempo de ejecución es la **construcción de un objeto muñeco**. A continuación, se presenta el código:

Tabla 22. Creación de un objeto en Java, junto a una llamada al método constructor

```
public class Snowman {
private double altura;
private double grosor;
public Snowman (double h, double w){
altura = h;
grosor = w;
}
public double getAltura() { return altura; }
public double getGrosor() { return grosor; }
// Crea el objeto muñeco de nieve, y muestra sus propiedades
public static void main(String [] args) {
Snowman snowMan1 = new Snowman(4.0, 2.0);
System.out.println("La altura del muñeco de nieve es "+
snowMan1.getAltura());
System.out.println(" , y su grosor vale "+ snowMan1.getGrosor() );
} // fin del método main
}// fin de la clase snowman
```

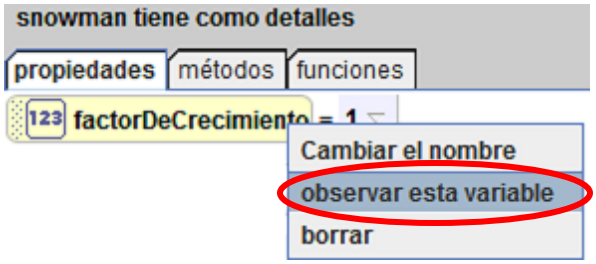
Lo que se ve es lo que se obtiene



Cuando un objeto muñeco de nieve se crea en Alice, podemos ver todas sus propiedades en la ficha del mismo nombre, como se ve en la imagen que está a la izquierda. En Java, cuando el objeto se crea, hay varias cosas que quedan invisibles al programador. En la memoria del computador, las **variables** que se crearon fueron la altura y el groso, y se guardaron en las propiedades del objeto muñeco de nieve, tal como se ve en la Tabla 22. En ese mismo código, se crearon los métodos **getAltura()** y **getGrosor()**, que pueden ser llamados para mostrar los valores en la pantalla. Como detalle técnico: la palabra **get** — que significa obtener en inglés — no es obligatoria, sino que por convención se utiliza; usted podría haber llamado al método como **obtenerAltura()**, o bien **obtenerGrosor()**: en ambos casos obtiene lo mismo. Cuando el código se ejecuta, se puede apreciar lo siguiente:

```
La altura del muñeco de nieve es igual a 4.0, y su grosor es
                        igual a 2.0
```

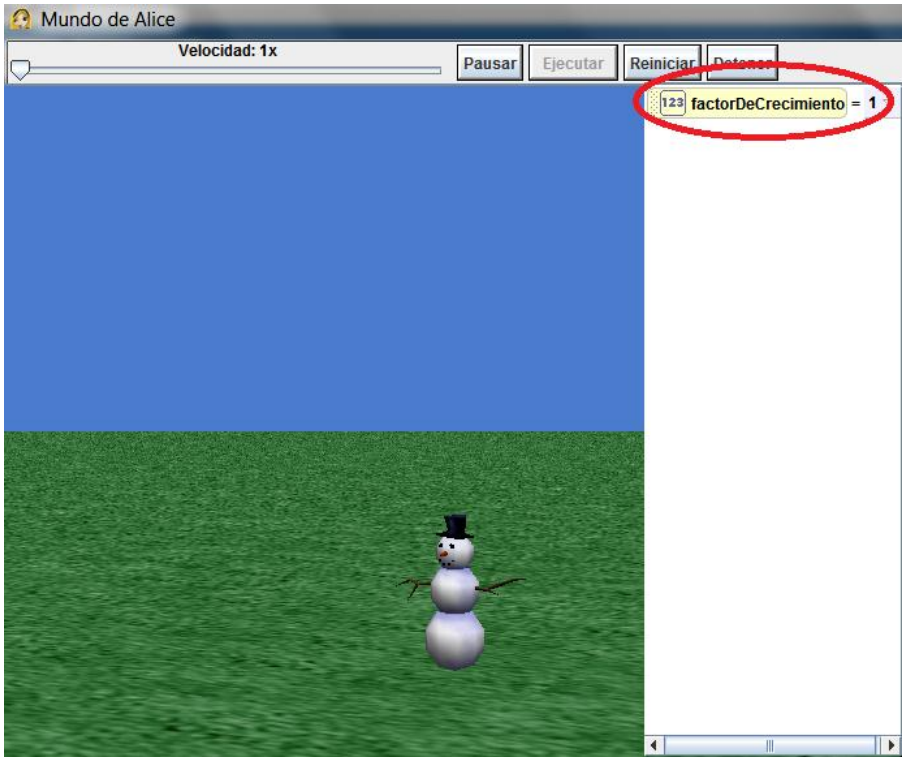
Claramente, la representación del objeto en el muñeco de nieve en Alice es muy visual, mientras que la representación del objeto muñeco en Java es textual, y por tanto más difícil de asimilar. Esto no quiere decir que ambas formas (gráfica y textual) no puedan darse en forma simultánea: es posible importar algunas **bibliotecas gráficas**, y escribir muchas líneas de código.



Otra característica similar de Alice con C++ o Java es la capacidad que ofrece para **observar las variables**. Por ejemplo, podemos crear una variable **factorDeCrecimiento** para el muñeco de nieve, y establecer una «*vigilancia*» sobre ella. Entenderemos que observar una variable es el proceso mediante el cual **se puede ver en tiempo de ejecución, la evolución de los valores que toma a medida que corre el programa**.

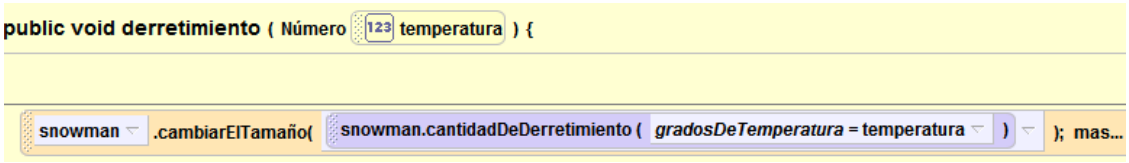
Cuando se ejecuta el programa, el cambio en el valor de la variable **factorDeCrecimiento** se muestra en la ventana que aparece a la derecha del mundo virtual, como aprecia en la Figura 124:

Figura 124. Valor que toma una variable en tiempo de ejecución.

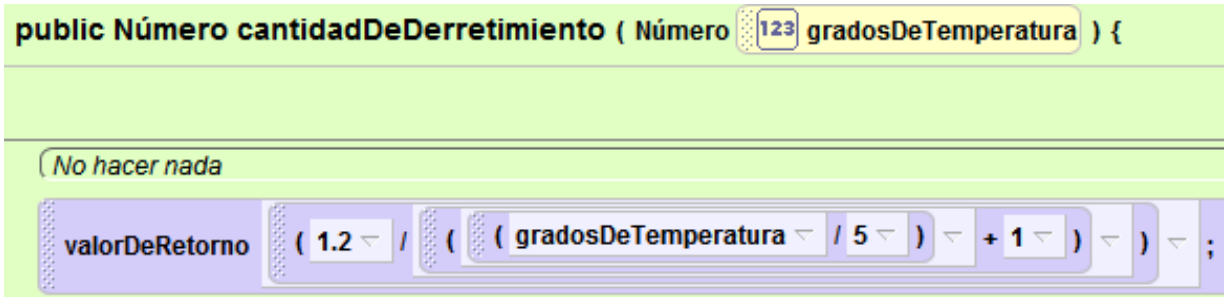


Crear nuevas clases

Supongamos que queremos crear un muñeco de nieve que se derrita como respuesta a un aumento de la temperatura externa. Para esto, se puede crear un método de objeto llamado **derretir**. El código que se muestra a continuación establece que el derretimiento se expresa como una disminución del tamaño del objeto, y está parametrizado en función del valor que adopte **temperatura**.



La incorporación del método **redimensionar** espera que el usuario le envíe un parámetro que le indique el factor de escalamiento en que el objeto se redimensionará. En este ejemplo, se ha creado también una función **cantidadDeDerretimiento** que determina cuánto es exactamente lo que se debiera escalar el porte del muñeco de nieve, en función de la temperatura, que se entrega como parámetro, y se llama **gradosDeTemperatura**. Esta pregunta se muestra en el código adjunto a este párrafo. Cuando la animación se ejecuta, el usuario podría ver un cambio mayor en la altura y en la anchura del muñeco de nieve, si el aumento de la temperatura fuera de 30º.



Una vez que los nuevos métodos se han escrito para el muñeco de nieve, podemos guardar el objeto con sus flamantes nuevos métodos como **una nueva clase**, de tal manera que después se pueda **reutilizar** gracias al mecanismo de **herencia** que provee Alice. En otros lenguajes, se pueden redefinir clases desde cero, pero en Alice no, porque el proceso de creación de un objeto 3D y de los métodos para mover, girar, rodar, etc., es demasiado tedioso, que va más allá de la intención educativa de esta herramienta. Por lo tanto, **la herencia en Alice se basa en trabajar con las clases que ya vienen integradas**.

¿Cómo se compara esto con la creación de una nueva clase de un lenguaje convencional? La Tabla 22 nos mostraba cómo crear una nueva clase del objeto muñeco de nieve. La definición de clase se escribe en un **editor** y se **compila**. También se puede crear una nueva clase en Java utilizando **herencia**. Vamos a crear una clase **MuñecoDeNieve2** que herede todas las propiedades y métodos de la clase **MuñecoDeNieve**, que le agregue el método de **derretimiento**, y la pregunta para calcular el monto en que finalmente se debiera escalar el tamaño del objeto. Observemos el código que se presenta en la Tabla 23.

Tabla 23. La clase MuñecoDeNieve2 hereda de la clase MuñecoDeNieve.

```
public class Snowman2 extends Snowman {
public void derretimiento (double degreeIncrease) {
altura = cantidadDeDerretimiento (altura, degreeIncrease);
grosor = cantidadDeDerretimiento (grosor, degreeIncrease);
}
private double cantidadDeDerretimiento(double dimension, double
gradosDeTemperatura) {
return dimension * ( 5 / gradosDeTemperatura );
}
// Se crea el objeto muñeco de nieve, y se establece el derretimiento con
10 grados centígrados

public static void main(String [] args) {
Snowman2 snowperson = new Snowman(4.0, 2.0);
snowperson.derretimiento(10.0);
System.out.println("La altura del muñeco de nieve es "+
snowperson.getAltura ());
System.out.println(" , y su ancho es igual a "+ snowperson.getGrosor() );
} // fin del método main
}// fin de la definición de la clase snowman class
```

Tenga en cuenta que el método **factorDeDerretimiento** es de tipo **privado**, lo que significa que no se puede invocar desde el método **main**. Cuando el método se llama, tanto la altura como la anchura del muñeco de nieve se encogen. Por supuesto, ningún objeto muñeco de nieve se puede ver en la pantalla, por lo que es imposible ver el cambio de tamaño. A lo más, el programa imprime un mensaje que da cuenta de los cambios que experimentaron la altura y el grosor.

Encapsulamiento

En Alice así como Java, cada una de ejemplos del programa muñeco de nieve hace uso de una clase muñeco de nieve. Se crean objetos como instancias de la clase por medio del **método constructor**. Aunque no se muestra en estos ejemplos particulares, varios ejemplares de muñecos de nieve se pueden crear y utilizar en nuestros proyectos. Cada objeto muñeco de nieve tiene sus propiedades particulares, y puede llevar a cabo ciertas acciones o métodos. Las propiedades de un objeto muñeco de nieve se dice que son **privadas**, porque sólo «*le importan*» a él, y a nadie más, de tal manera que la única forma de poder acceder a ellas es a través de los métodos **getAltura()** y **getGrosor()** que vimos en el ejemplo de Java. En Alice empleamos funciones para poder acceder a la información de la privacidad del estado del objeto. Una manera de pensar en un objeto es compararlo con una **cápsula**, ya que lo podemos ver como si fuera una unidad autosuficiente. Esta característica recibe el nombre de **encapsulamiento**, ya que el objeto puede encubrir sus propiedades y métodos.

Tipos de datos y estructuras

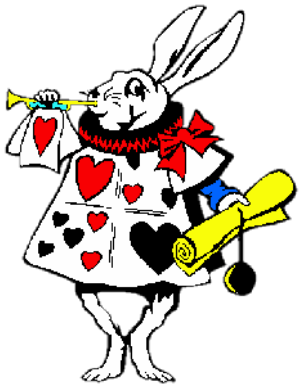
Las propiedades de los objetos en Alice pueden ser de diferente naturaleza: **número, booleano, string, color, textura o ubicación**. Salvo en el caso de textura y ubicación, estos **tipos de datos** son comunes en todos los lenguajes orientados a objetos. La **textura** corresponde al **mapa de una imagen**, que puede representar la superficie de un objeto, mientras que la **ubicación** corresponde a las coordenadas que un objeto tiene dentro del mundo.

Alice es compatible con las listas y los arreglos, al igual que la mayoría de los lenguajes orientados a objetos. Las estructuras de datos son colecciones de elementos de datos, que se organizan en un cierto orden.

Métodos y funciones (preguntas)

En la programación orientada a objetos, las acciones que se pueden realizar con los objetos se materializan a través de los **métodos**, pero también se pueden aplicar otros nombres para estos grupos de acciones, que son bastante comunes entre los desarrolladores de programas computacionales: **modificador, descriptores y funciones**. Los **modificadores** son métodos que realizan una acción que cambia el objeto de alguna forma. En el programa anterior que revisamos, el método **derretir** es de tipo **modificador**, porque cambia las dimensiones del objeto. Los **descriptores** son métodos que **recuperan información** de algunas propiedades de un objeto. En el ejemplo anterior del programa Java, los métodos **getAltura()** y **getGrosor()** son descriptores, porque sirven para obtener la altura y la anchura del objeto. Esta clase de métodos no es necesaria en Alice, porque las acciones de modificar propiedades, o de imprimir algún mensaje son hechas por el programador.

Las **funciones** — que en Alice reciben el nombre de **preguntas** — en el fondo **también son métodos** que permiten realizar algunas operaciones aritméticas o lógicas, y que a cambio devuelven un valor de un cierto tipo que responda al resultado esperado de la operación. En el ejemplo anterior de Alice y Java, el método **cantidadDeDerretimiento** se considera una función, ya que retorna un valor. Es importante señalar que la terminología que se ha discutido, no se aplica sistemáticamente: de hecho, **en C++ todas las acciones se consideran funciones**, independiente si devuelven o no algún valor, etiquetándose como **void** a aquellas que no lo hagan.



Tercera Parte

Construcciones de Programas

5.3 Constructores de programas

Las construcciones de programas se usan para **controlar el flujo de ejecución de las instrucciones** que lo componen, **siguiendo el orden** en que fueron definidas. En las formas tradicionales de programación, este flujo puede ser **secuencial**, en el sentido que cada comando se va ejecutando de a una línea a la vez, de principio a fin.

También, el flujo se puede **ramificar** en función de las **estructuras condicionales** y de los **bucles finitos o infinitos** que puedan aparecer en el camino. Los lenguajes de programación modernos están incorporando otras formas de control, donde el orden de ejecución depende de la interacción con el usuario, que tienen la particularidad de ser dirigidos por **eventos** que lo afectan de manera significativa. La **programación orientada a eventos** a menudo utiliza muchas aplicaciones de software que hacen uso de ventanas, botones, casillas de verificación, y de otros **widgets** destinados para la interacción con el usuario. En Java, las clases swing y AWT proporcionan eventos orientados a la capacidad para crear **interfaces gráficas de usuario**.

Las animaciones que se crean en Alice son equivalentes a un **programa no orientado a eventos** cuando las acciones se desarrollan como si fueran una película o un dibujo animado. Sin embargo, las animaciones en Alice también pueden ser interactivas, al permitir que el flujo de ejecución pueda estar en las manos del usuario. Una animación interactiva en Alice consiste en métodos que están vinculados a eventos, usando en este caso el teclado y el ratón como dispositivos de comunicación humana.

En esta parte nos vamos a centrar en estos bloques de construcción de programas, para que usted tenga una introducción más completa hacia **Java** o **C++**.

Hacer en orden y hacer juntos

Los primeros constructores que usamos en este apunte son **hacer en orden** y **hacer juntos**, donde el primero corresponde al orden secuencial de acciones en Alice. La diferencia con Java y C++ es que en Alice se puede especificar de manera explícita que un determinado grupo de acciones se haga en secuencia, mientras que en Java y C++ se **asume que es así**. Entonces:

¿Para qué se necesita hacer una distinción entre un tipo de bloque y otro?

Las instrucciones que se ejecutan simultáneamente — o que al menos así lo parecen — se dice que son **concurrentes**. Es importante tener claro que **esta característica no está presente en todos los lenguajes de programación**.

La concurrencia en Java existe gracias a la creación de estructuras llamadas **hebras** o **hilos**, que proporcionan una manera de dividir un programa en varias tareas independientes que se hagan por separado. El **hacer juntos no corresponde exactamente a hilos de Java**, porque éstos pueden comunicarse entre sí, y en Alice eso no es posible.

Toma de decisiones

El constructor fundamental que se emplea en Alice para tomar decisiones es if/else. Un ejemplo de esto es lo que se muestra en la Figura 125, que tiene la sintaxis de Java activada, mientras que el código se puede ver en la Figura 125. Ejemplo de if/else con la sintaxis de Java activada.

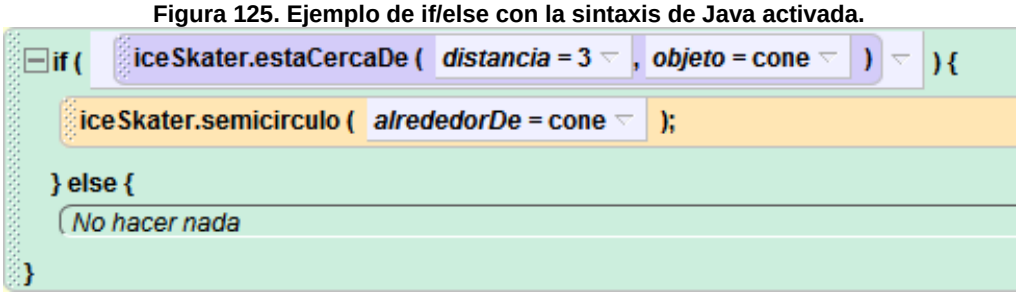


Tabla 24. El ejemplo supone que la patinadora es un objeto de una clase que define un método para dibujar un círculo, y una función booleana para determinar la cercanía con otro objeto:

Figura 125. Ejemplo de if/else con la sintaxis de Java activada.

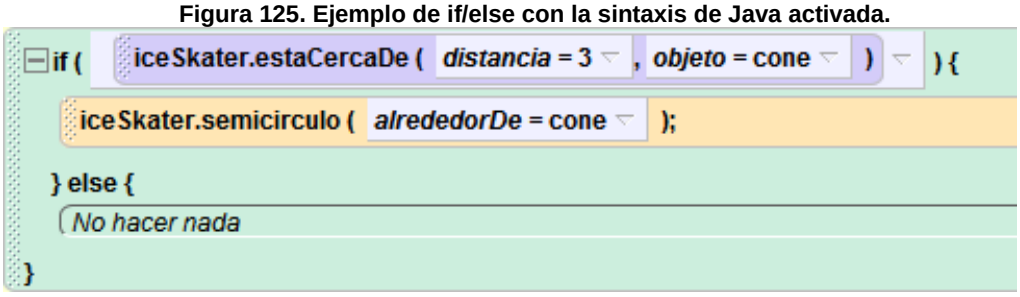


Tabla 24. Un ejemplo de construcción if/else en Java.

```
if (IceSkater.estaCercaDe(3,cone)) {
```

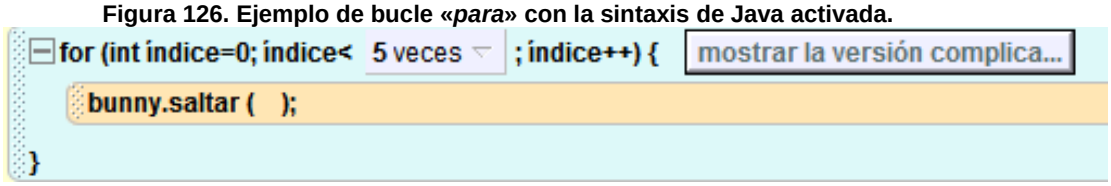
```
IceSkater.semicirculo(cone);
}
```

Una diferencia en cómo Alice hace un **if/else** con respecto a Java o C++, es que Alice siempre incluye la opción de **else**, pero en estos otros lenguajes se puede omitir si acaso el programador lo encuentra necesario. Que esté presente en Alice no altera la consistencia de los códigos, porque siempre considera las posibilidades de acción cuando la condición se evalúe como verdadero o como falso. En Java y en C++ existe una instrucción llamada **switch**, que es una especialización del **if/else** que no le agrega mayor poder computacional a las aplicaciones.

Repetición: bucle (para)

En Alice vimos cómo se pueden recrear bloques de programación que permitan repetir un conjunto de instrucciones de una manera coherente. En Java y en C++ eso también se da, con la salvedad de algunas excepciones de corte sintácticas. El **bucle para** se muestra en la Figura 126, y tiene activada la sintaxis para Java:

Figura 126. Ejemplo de bucle «*para*» con la sintaxis de Java activada.



En cambio, si lo apagamos y volvemos al estilo tradicional de Alice, llegamos a una expresión equivalente que se aprecia a continuación:

En la Tabla 25 se muestra un código que ha sido escrito como si estuviera dado en un editor de texto, en el cual hemos supuesto que **bunny** es una instancia de una clase **Conejo** que tiene implementado el método **saltar**:

Tabla 25. Un código en Java que contiene un bucle «*para*».

```
for (int i = 0; i < 5; i++) {
bunny.saltar();
}
```

Repetición: mientras

Por último, el **bucle while** se ilustra en la Figura 127 con la sintaxis de Java activada, y lo mismo se expone en la Tabla 26, donde vuelve a aparecer el objeto **Conejillo**, al que se ha agregado una pregunta de **distancia**:

Figura 127. Ejemplo de bucle «*while*» con la sintaxis de Java activada.

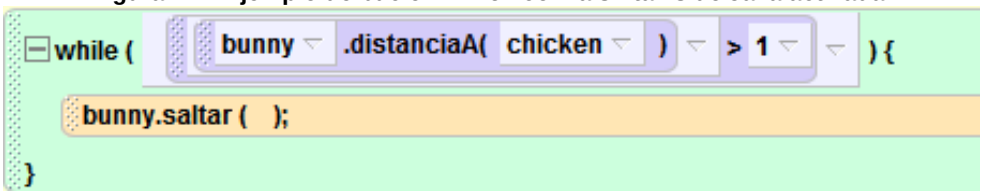


Tabla 26. Ejemplo de un bucle «*while*» escrito en Java.

```
while (bunny.distanciaA(chicken) > 1 ) {
bunny.saltar();
}
```

Resumen

La transición de Alice a Java, C++ o C# es un proceso que consiste en hacer conexiones entre lo que hemos aprendido a lo largo de estas cinco unidades, y las particularidades de los otros lenguajes. La buena noticia es que Alice le ha proporcionado a usted todos los conceptos fundamentales para que usted pueda emprender con éxito el aprendizaje de herramientas de construcción de programas mucho más potentes. El énfasis se ha puesto en disminuir los niveles de frustración que originan los errores sintácticos, y que este proceso haya resultado de gran entretención para usted.

Para hacer que la transición a otros lenguajes de programación sea más fácil, en esta unidad hemos incorporado el **interruptor de sintaxis de Alice**, que le permite ver cómo las animaciones que estaban hechas inicialmente lucen cuando las transportamos a otros entornos de programación, donde partíamos observando los **tipos de datos** que **retornaban los métodos**, la inclusión de **llaves, paréntesis, comas y punto y comas**. Dijimos que en Alice los métodos son de tipo **void** porque **no retornan valores**, y las **funciones** sí tienen asociado un **tipo de valor de retorno**, aunque en algunos lenguajes no se respete esta diferencia

Las clases de objetos en Alice se definen por los modelos proporcionados en las galerías 3D de Alice. En otros lenguajes, podrá definir sus propias clases utilizando un editor de texto, pero en Alice eso no se puede hacer, ya que el proceso de creación de objetos 3D junto a las operaciones clásicas es demasiado tedioso

para un curso introductorio como este. En Alice — así como en otros lenguajes de programación —, un **objeto es una instancia de una clase**, y tiene **propiedades y métodos que encapsula**. En Alice, usted puede ver con facilidad un objeto y sus propiedades, pero en otros lenguajes la única manera es verlos a través de llamas a **métodos descriptores** que permiten imprimir los valores en la pantalla.

Los **bloques de construcción** se usan para **regular el flujo de las instrucciones a lo largo de la ejecución de un programa**. En Alice, se ha ocupado el **if/else** para decidir si algo se ejecuta o no, en función de una expresión condicional de tipo booleana. También, hemos revisado otros que permiten hacer repeticiones, como los **bucles para (for) y while**, que están presentes en todos los lenguajes de programación orientada a objetos.

Conceptos importantes en este capítulo

- La **sintaxis** es un término técnico que define las normas que especifican cómo se deben escribir las instrucciones y signos de puntuación para construir **declaraciones o sentencias** en un programa.
- El **encabezado** de un método o de una **función** es donde se especifica el nombre, los parámetros que debiera recibir (junto con el tipo), y el tipo de dato que se va a devolver, si lo hay.
- **En Alice, un método no devuelve un valor**, y su encabezado tiene la etiqueta **void**.
- **En Alice, una pregunta o función tiene que retornar algún tipo de valor**. En otros lenguajes, los conceptos de método y función se confunden, y se toman como si fueran lo mismo.
- El flujo de programas puede ser **secuencial** o **ramificado**. También existe la posibilidad de dirigir el curso de las acciones por medio de **eventos**, que corresponden a acciones provocadas por el usuario o por otro agente del programa.
- Los métodos y preguntas se componen de bloques de construcción de código que contienen una o más instrucciones.

Orientaciones Pedagógicas

Para conseguir los aprendizajes esperados, es necesario que los estudiantes tengan claridad con respecto a los siguientes puntos:

- El proceso de programación en Java es muy similar al que se emplea en Alice: leer el escenario, diseñar una solución, implementar el programa, y probar lo que se ha hecho.
- La resolución de problemas a menudo obliga a hacer una descomposición en subproblemas.
- Una clase describe todos los objetos que sean de un cierto tipo.
- Los métodos se pueden usar para lograr la comunicación entre objetos.
- Algunos métodos requieren parámetros para que funcionen, y que deben corresponder con el tipo apropiado para funcione de acuerdo a lo esperado.
- El código fuente es el texto donde se pueden definir los detalles de una clase.
- El comportamiento de un objeto se define a través de los métodos que se hayan definido.
- BlueJ:
  - Provee una interfaz que permite crear objetos, inspeccionar sus propiedades, y llamar a los métodos que se les hayan creado.
  - El proceso para llamar a un método en BlueJ es similar al que se ocupa en Alice, en el sentido que la selección se hace desde aquellos que efectivamente estén habilitados para tales efectos.

Para que estos entendimientos se puedan llevar a cabo de manera exitosa, se recomienda que los estudiantes desarrollen en las sesiones de práctica no sólo los problemas propuestos al final de la unidad, sino también que reflexionen en torno a estas preguntas, que se pueden distribuir en el tiempo que el profesor estime conveniente:

- ¿En qué sentido el ambiente de trabajo que provee BlueJ se parece al de Alice?
- ¿En qué se diferencia BlueJ de Alice?
- ¿Cuáles son los pasos que se requieren para poder ejecutar alguna aplicación en BlueJ?
- ¿Qué significa compilar una clase, y por qué es necesario compilar los programas hechos en Java?
- En el contexto de BlueJ:
  - ¿Qué significa el concepto de «*estado de un objeto*»?
  - ¿Cómo se produce la transferencia mutua de información entre métodos?
  - ¿Qué información se requiere para poder escribir alguna clase en Java?