

**UNIVERSIDAD DEL BÍO-BÍO**  
FACULTAD DE CIENCIAS EMPRESARIALES  
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN Y  
TECNOLOGÍA DE LA INFORMACIÓN



**DISEÑO E IMPLEMENTACIÓN DE UN  
VIDEOJUEGO DE ACCIÓN-AVENTURA 2D PARA  
DISPOSITIVOS ANDROID UTILIZANDO  
EL MOTOR UNITY**

**MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN INFORMÁTICA**

**AUTOR : ARAVENA VERGARA CRISTIAN ENRIQUE**

**Profesor Guía : Gajardo Díaz Luis Daniel**

**CHILLÁN, 2016**

## **Agradecimientos**

Quisiera agradecer principalmente a mi familia, por el apoyo incondicional que me han brindado durante todos estos años. Además, a Camilo Pino de Jerbosis Art, que aportó muchísimo y fue un apoyo en el arte gráfico del proyecto; a Kevin MacLeod, que a pesar de apenas conocerlo aceptó ayudarme con el arte musical y en especial al profesor Luis Gajardo, quien guío todo el desarrollo de este trabajo.

También quiero agradecer a mis compañeros del taller de videojuegos Botacura Games, con los cuales aprendí bastante en cada competencia que participábamos y fueron la principal motivación de este proyecto.

Finalmente, agradecer a todos los amigos que logré conocer en esta universidad, por su incondicional amistad y a todos los profesores que durante toda esta carrera me dieron las herramientas necesarias para afrontar la vida laboral.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

## ***Dedicatoria***

*A mi Madre, por su incansable lucha y nunca bajar los brazos*  
*A mi Padre, por enseñarme a nunca rendirme y dar lo mejor de mí*  
*A mis hermanas, por siempre apoyarme*  
*A mis sobrinos que son mi vida.*

***Cristian Enrique Aravena Vergara***

## RESUMEN

En la Actualidad, los videojuegos tienen un rol muy importante en el rubro del entretenimiento a nivel mundial, de hecho, es un fenómeno social que traspasa todas las edades, sin distinción de género.

Es por esto, que el siguiente proyecto enfatiza el desarrollo de un videojuego, cuya motivación nace del deseo de crear un juego orientado a la Universidad y sus estudiantes, que permita al jugador divertirse con éste y demostrar la versatilidad que tiene un egresado de la carrera Ingeniería civil en informática al momento de aplicar todo lo aprendido durante el proceso de aprendizaje en la casa de estudios.

En el desarrollo de este proyecto se utilizó la metodología de proceso de software Iterativa Incremental, con el fin de producir continuamente incrementos completamente funcionales. Además se debió ajustar la etapa de diseño, producto de lo distinto que resulta el desarrollo de un videojuego en comparación a un software tradicional.

Una vez finalizado este proyecto, se obtiene un videojuego de Plataformas 2D, con funcionalidades y estética de nivel profesional, que entregan al usuario final una experiencia de entretenimiento única.

# ÍNDICE GENERAL

CAPÍTULO I .....	1
ANTECEDENTES GENERALES.....	1
1.1 MOTIVACIÓN DEL PROYECTO .....	2
1.2 OBJETIVO DEL PROYECTO.....	2
1.2.1 OBJETIVO GENERAL .....	2
1.2.2 OBJETIVOS ESPECÍFICOS .....	2
1.3 METODOLOGÍA A UTILIZAR EN EL PROYECTO .....	3
CAPÍTULO II .....	4
ESTUDIO DE FACTIBILIDAD.....	4
2.1 FACTIBILIDAD TÉCNICA. ....	4
2.2 FACTIBILIDAD OPERATIVA. ....	5
2.3 FACTIBILIDAD ECONÓMICA .....	5
2.4 INVERSIÓN DEL PROYECTO .....	6
2.4.1 INVERSIÓN PARA EL DESARROLLO.....	6
2.4.2 INVERSIÓN PUESTA EN MARCHA DEL PROYECTO.....	7
2.5 INGRESOS DEL PROYECTO .....	8
2.6 COSTOS DEL PROYECTO.....	8
2.7 FLUJO NETO DE CAJA Y VAN DEL PROYECTO .....	9
2.8 CONCLUSIÓN DE LA FACTIBILIDAD .....	10
CAPÍTULO III .....	12
MARCO TEÓRICO.....	12
3.1 ORIENTACIÓN AL MUNDO 2D .....	12
3.1.1 ORIENTACIÓN Y SISTEMA DE COORDENADAS .....	13
3.1.2 TRANSFORMACIONES GEOMÉTRICAS .....	14
3.1.2.1 TRASLACIÓN .....	15
3.1.2.2 ROTACIÓN.....	16
3.1.2.3 ESCALA (REDIMENSIONADO).....	16
3.1.3 LA CÁMARA .....	18
3.1.4 LOS OBJETOS RENDERIZABLES.....	19
3.2 LOS MOTORES DE VIDEOJUEGOS .....	20
3.2.1 DEFINICIÓN.....	21
3.2.2 HISTORIA .....	22
3.3 UNITY 2D.....	24
3.3.1 ASPECTOS GENERALES.....	24
3.3.2 COMPONENTES BÁSICOS DE UNITY.....	24
3.3.2.1 INTERFAZ DE USUARIO DE UNITY.....	26
3.3.2.2 MENÚ DE APLICACIONES .....	27
3.3.2.3 BOTONES CONTROL.....	27
3.3.2.4 BOTONES DE REPRODUCCIÓN .....	28
3.3.3 COMPONENTES TÉCNICOS UNITY 2D .....	29
3.3.3.1 SPRITE RENDERER .....	29
3.3.3.2 ANIMATION / ANIMATOR .....	30
3.3.3.3 CANVAS .....	31
3.4 TIPOS DE VIDEOJUEGOS MÓVILES .....	32

CAPÍTULO IV.....	34
ARQUITECTURA Y DISEÑO.....	34
4.1 DIAGRAMA DE CLASES .....	35
4.2 CASOS DE USO .....	36
4.3 PATRONES DE DISEÑO .....	40
CAPÍTULO V.....	43
DESARROLLO DEL VIDEOJUEGO .....	43
5.1 CONTEXTUALIZACIÓN DEL VIDEOJUEGO.....	44
5.1.1 MOTIVACIÓN .....	44
5.1.2 TRAMA DEL VIDEOJUEGO .....	45
5.1.3 CARACTERIZACIÓN DE PERSONAJES.....	45
5.1.4 STORYBOARD .....	47
5.2 ELEMENTOS GRÁFICOS.....	48
5.2.1 PALETA DE COLORES.....	49
5.2.2 TIPOGRAFÍA.....	50
5.2.3 MENÚS .....	50
5.3 CONSTRUCCIÓN DEL ESCENARIO .....	52
5.3.1 CREACIÓN DEL ESCENARIO .....	52
5.3.2 NIVELES DEL JUEGO .....	54
5.3.3 PARALLAX .....	57
5.3.4 GESTIÓN DE LA CÁMARA .....	57
5.4 CONTROL DE LA INTERACCIÓN [Lenguaje de Scripting] .....	58
5.4.1 CONTROL DE PERSONAJE PRINCIPAL.....	59
5.4.2 INTELIGENCIA ARTIFICIAL .....	62
5.4.2.1 INTELIGENCIA ARTIFICIAL BASADA EN AGENTE DE REFLEJO SIMPLE .....	62
5.4.2.2 INTELIGENCIA ARTIFICIAL BASADA EN ÁRBOL DE COMPORTAMIENTO.....	64
5.4.3 PERSISTENCIA.....	65
5.4.3.1 MODELO DE DATOS .....	66
5.4.3.2 GUARDAR Y RECUPERAR PARTIDA.....	67
5.4.3.3 REGISTRO DEL RANKING.....	67
5.4.3.4 GESTIÓN DE LOS NIVELES.....	68
CAPÍTULO VI.....	69
PRUEBAS.....	69
6.1 PRUEBAS DE USABILIDAD .....	69
6.1.1 EVALUACIÓN DEL PROTOTIPO .....	70
6.1.2 PROCEDIMIENTO .....	70
6.1.1.2 RESULTADOS .....	71
6.2 PRUEBA DE RENDIMIENTO .....	72
6.2.1 PROCEDIMIENTO .....	72
6.2.2 RESULTADOS .....	73
6.3 PRUEBAS UNITARIAS .....	74
6.3.1 PROCEDIMIENTO .....	74
6.3.2 RESULTADOS .....	75
CAPÍTULO VII .....	76
PUESTA EN MARCHA.....	76
7.1 PLATAFORMAS DE EJECUCIÓN .....	77
7.1.1 PC .....	77
7.1.2 ANDROID.....	77

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

7.2 PUBLICACIÓN DE LA APLICACIÓN .....	78
7.2.1 GOOGLE PLAY .....	78
7.2.2 SERVIDOR WEB .....	79
CONCLUSIÓN .....	82
RESULTADOS OBTENIDOS .....	83
PROYECCIÓN A FUTURO .....	83
<b>BIBLIOGRAFÍA</b> .....	84
ANEXOS. ....	86
ANEXO 1: STORYBOARD .....	86
ANEXO 2: PALETA DE COLORES .....	92
ANEXO 3: SCRIPTING .....	93
ANEXO 4: PRUEBAS .....	101

# ÍNDICE DE FIGURAS

FIGURA 1, SISTEMA DE COORDENADAS.....	13
FIGURA 2, DESCRIPCIÓN DE WORLD SPACE Y LOCAL SPACE.....	14
FIGURA 3, OPERACIONES PARA REALIZAR UNA TRASLACIÓN EN UN SISTEMA TRIDIMENSIONAL.....	15
FIGURA 4, EJEMPLO DE ROTACIÓN EN UN EJE DE 2 DIMENSIONES. ....	16
FIGURA 5, FÓRMULA PARA ROTACIÓN. ....	16
FIGURA 6, EJEMPLO DE ESCALADA BIDIMENSIONAL.....	17
FIGURA 7, FÓRMULA PARA ESCALADO. ....	17
FIGURA 8, OPERACIONES DE TRANSFORMACIÓN SUCESTIVAS SOBRE UN OBJETO.....	17
FIGURA 9, ÁNGULO DE VISIÓN (FOV) DE CÁMARA DE PERSPECTIVA.....	18
FIGURA 10, TAMAÑO Y FORMA DE CÁMARA ORTOGRÁFICA. ....	19
FIGURA 11, COMPARATIVA DE VISIÓN EN 2D TERCERA PERSONA Y 3D PRIMERA PERSONA. ....	19
FIGURA 12, ANIMACIÓN DE MEGAMAN DESPLAZÁNDOSE.....	20
FIGURA 13, ESQUEMA DE ARQUITECTURA DE UN MOTOR GRÁFICO. ....	21
FIGURA 14, ENUMERACIÓN DE ÁREAS EN UNITY.....	26
FIGURA 15, MENÚ DE APLICACIONES DE UNITY.....	27
FIGURA 16, BOTONES DE CONTROL.....	27
FIGURA 17, BOTONES DE REPRODUCCIÓN EN UNITY.....	28
FIGURA 18, COMPONENTE SPRITE RENDERER.....	29
FIGURA 19, HERRAMIENTA ANIMATION Y SU INCORPORACIÓN JUNTO A MECANIM.....	30
FIGURA 20, HERRAMIENTA ANIMATOR JUNTO A UN DIAGRAMA DE ESTADOS DE ANIMACIÓN.....	31
FIGURA 21, CANVAS Y SUS PROPIEDADES.....	32
FIGURA 22 DIAGRAMA DE CLASES DE GEEKNATOR.....	35
FIGURA 23 CASOS DE USO DE GEEKNATOR.....	36
FIGURA 24, SCRIPT DE INICIALIZACION DE PATRÓN SINGLETON, RUTINA DE RELOJ Y ACTUALIZACIÓN DE PUNTAJE.....	42
FIGURA 25, PALETA DE COLORES NIVEL 1 DE GEEKNATOR.....	49
FIGURA 26, MENU PRINCIPAL DE GEEKNATOR.....	51
FIGURA 27, MENÚ DE CONFIGURACIÓN DE SONIDO EN GEEKNATOR.....	51
FIGURA 28, COMPARACIÓN DE ESCENARIO GENERADO.....	53
FIGURA 29, COMPARACIÓN DE ESCENARIO Y SU ARCHIVO DE TEXTO.....	53
FIGURA 30, TROZO DE ESCENARIO 2.....	55
FIGURA 31, RESULTADO DE BOXCOLLIDER APLICADO EN TILE PISO.....	55
FIGURA 32, COMPONENTES AGREGADOS AL TILE DE PISO.....	56



Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

FIGURA 33, SCRIPT DE SALUD DEL PERSONAJE PRINCIPAL .....	61
FIGURA 34, PUNTO DE VISIÓN Y ESTIMULACIÓN DE ENEMIGO. ....	62
FIGURA 35, PSEUDOCÓDIGO AGENTE DE REFLEJO SIMPLE .....	63
FIGURA 36, AUTÓMATA AGENTE DE REFLEJO SIMPLE .....	63
FIGURA 37, AUTÓMATA GENERADO A TRAVÉS DE ÁRBOL DE COMPORTAMIENTO.....	64
FIGURA 38, FÓRMULA PARA DETERMINAR LA DISTANCIA DE MANHATTAN. ....	65
FIGURA 39, MODELO DE DATOS GEEKNATOR. ....	66
FIGURA 40, PERFIL DE GEEKNATOR EN GOOGLE PLAY STORE. ....	79
FIGURA 41, CARGA DE PLUG-IN EN PÁGINA WEB. ....	80
FIGURA 42, JUEGO CARGADO EN WEBGL .....	81

# ÍNDICE DE TABLAS

TABLA 1 - HARDWARE Y SOFTWARE NECESARIOS PARA LA IMPLEMENTACIÓN DE LA SOLUCIÓN .....	4
TABLA 2 - CUADRO VALOR TOTAL COSTO MANO DE OBRA. ....	6
TABLA 3 - CUADRO VALOR TOTAL DE ANÁLISIS, DISEÑO Y CONSTRUCCIÓN .....	6
TABLA 4 - INVERSIÓN PUESTA EN MARCHA .....	7
TABLA 5 - CUADRO INVERSIÓN TOTAL DEL PROYECTO .....	8
TABLA 6 - VALOR ESTIMACIÓN DE DESCARGAS POR DISPOSITIVO.....	8
TABLA 7 - COSTOS DEL PROYECTO .....	8
TABLA 8 - CUADRO FLUJO NETO DE CAJA.....	10
TABLA 9 - CASO DE USO CU-01 MOVER PERSONAJE.....	37
TABLA 10 - CASO DE USO CU-02 ATACAR CON PERSONAJE.....	37
TABLA 11 - CASO DE USO CU-03 INICIAR PARTIDA NUEVA. ....	38
TABLA 12 - CASO DE USO CU-04 SELECCIONAR ESCENARIO .....	38
TABLA 13 - CASO DE USO CU-05 CARGAR PARTIDA GUARDADA .....	38
TABLA 14 - CASO DE USO CU-06 ABANDONAR NIVEL .....	39
TABLA 15 - CASO DE USO CU-07 PAUSAR NIVEL.....	39
TABLA 16 - CASO DE USO CU-08 REINICIAR NIVEL .....	39
TABLA 17 - CASO DE USO CU-09 VISUALIZAR MEJORES PUNTAJES.....	40
TABLA 18 - CARACTERÍSTICA DE LOS PERSONAJES EN GEEKNATOR. ....	47
TABLA 19 - TIPO DE LETRAS UTILIZADAS EN GEEKNATOR. ....	50
TABLA 20, TIPOS DE CÁMARAS USADOS EN GEEKNATOR.....	58
TABLA 21, MÉTRICA UTILIZADA PARA EVALUAR EL DESEMPEÑO DEL VIDEOJUEGO. ....	71
TABLA 22, ESPECIFICACIONES DE CARACTERÍSTICAS EN PRUEBA. ....	73

## INTRODUCCIÓN

Hoy en día la industria de los videojuegos en Chile, aún no posee la madurez y envergadura de otros países del mundo, debido al gran riesgo que supone desarrollar un videojuego y al poco conocimiento por parte de las empresas e instituciones de herramientas que son comúnmente utilizadas para su creación, esto es, los motores de videojuego.

Actualmente existen empresas Chilenas de desarrollo de videojuegos, como ACE Team, Atakama Labs, Behaviour Games, AmnesiaGames, Baytex, Mazorca Studios que orientan su producto a un público objetivo, ya sea consolas de sobremesa o portátiles; pero muy pocas empresas se dedican a desarrollar videojuegos para dispositivos Android que sean para un público general, ya que éstas no tienen un impacto en el grupo de usuarios, a estos tipos de juegos se les conoce como Indie-Games.

El presente proyecto aborda el tema del desarrollo de videojuegos utilizando tecnologías y herramientas especializadas, que permiten reducir tanto tiempo, así como también la complejidad que supone la producción de un videojuego. Todo esto, con el fin de mostrar la accesibilidad y el gran potencial que poseen herramientas como Unity.

Se espera lograr un videojuego atractivo, con una estética que introduzca al usuario en la aventura, con una jugabilidad profesional y agradable que comparta la sensación de control de la situación en todo momento al jugador y lo motive a jugar de forma periódica.

El siguiente informe, detalla el desarrollo del Videojuego en cuestión y este documento se organiza de la siguiente manera:

En el primer capítulo se introduce sobre los objetivos del proyecto, dejando en claro cuáles serán las características de éste, en el segundo capítulo se hace el estudio de factibilidad, que determina si es conveniente realizar el proyecto de software. En el tercer capítulo se hablará de toda la teoría que hay detrás de un proyecto de esta envergadura, detallando como se subdivide un motor gráfico. En el cuarto capítulo se habla sobre la arquitectura a aplicar para el desarrollo del videojuego, en el quinto capítulo se documenta el desarrollo del videojuego y se subdivide las características desarrolladas. En el sexto capítulo se mencionan los tipos de pruebas aplicadas para evaluar y gestionar el videojuego. En el séptimo capítulo se habla sobre la puesta en marcha y como se publica el videojuego en las distintas plataformas.

# Capítulo I

## Antecedentes Generales

---

El siguiente capítulo tiene como fin introducir y comunicar al lector los objetivos del proyecto a desarrollar, así como también conocer sus principales características y funcionalidades.

En primer lugar, se comenzará con una introducción general, la cual presenta a grandes rasgos, las principales características del proyecto. Luego, se describe la problemática a abordar, en donde se menciona la situación actual y la alternativa de solución.

Además, se presentan los objetivos del proyecto, con el fin de comprender mejor lo que se quiere desarrollar. Por último, se describe la metodología de desarrollo de software seleccionada para construir el proyecto.

## **1.1 Motivación del Proyecto**

El grupo de desarrollo de videojuegos "Botacura Games" de la universidad del Bio-Bio, donde actualmente el autor de este proyecto participa activamente, ha participado constantemente en actividades de difusión. Un problema frecuente al momento de exponer es que el público solicita un videojuego orientado a dispositivos móviles, con el cual se pueda ver el trabajo realizado por el grupo en otro momento con más dedicación y tiempo.

Es por eso que surge la motivación de desarrollar un videojuego para dispositivos móviles, el cual pueda servir para interiorizar a los estudiantes de enseñanza media a la carrera en cuestión.

El género del videojuego será de tipo plataformas 2D el cual relatará una historia ficticia relacionada con el campus Fernando May, sus docentes, estudiantes y edificios.

## **1.2 Objetivo del Proyecto**

### **1.2.1 Objetivo General**

Desarrollar un videojuego para dispositivos Android que logre ser de apoyo en charlas de difusión de carrera o universidad abierta; utilizando el motor gráfico Unity.

### **1.2.2 Objetivos Específicos**

A continuación se presentan los objetivos específicos del proyecto a desarrollar, los cuales muestran más en detalle las características principales de éste:

- Desarrollar 3 escenarios con distinta ambientación y misiones.
- Desarrollar un personaje principal con sus respectivos movimientos, scripts y sprites (imágenes y animaciones del personaje).
- Diseñar una Inteligencia Artificial (agente de reflejo simple) para al menos 4 tipos de enemigos de nivel distintos.
- Diseñar una Inteligencia Artificial (Árbol de comportamiento) para al menos dos enemigos de zona.

### **1.3 Metodología a Utilizar en el Proyecto**

El modelo de proceso de desarrollo de software a utilizar en la realización de este proyecto es el iterativo incremental, el cual permitirá generar incrementos de software operacionales, los cuales entregarán una idea del avance y cumplimiento de los objetivos del proyecto.

Cabe mencionar que esta metodología solo se tomará como una referencia, debido a que el proceso de desarrollo de un videojuego es muy distinto al desarrollo tradicional de software, en donde el proceso de desarrollo es más flexible y un poco más informal. Sin embargo, igualmente se utilizarán aspectos de la metodología, con el objetivo de promover la ingeniería del software.

El desarrollo constará de 3 incrementos, iniciando por el desarrollo del personaje principal, el escenario y la implementación de la cámara; para luego implementar la jugabilidad y el parallax y terminar con la incorporación de enemigos, el diseño de los menús e interfaz de usuario y finalmente llegar a los records y logros del juego.

# Capítulo II

## Estudio de Factibilidad

---

Este estudio tiene como meta identificar la disponibilidad de recursos para llevar a cabo los objetivos del proyecto a través de 5 aspectos: Factibilidad técnica, operacional, económica, política, y de fechas.

Gracias a estos estudios se puede concluir el éxito de este proyecto, el cual está determinado por el grado de rentabilidad que se pueda llegar a lograr.

### 2.1 Factibilidad técnica.

Este estudio consiste en determinar si técnicamente es factible desarrollar el proyecto, si en el mercado existe el hardware y software necesarios para su implementación.

En la Tabla 1 se presenta el hardware y software necesarios para la puesta en marcha del proyecto.

Requisitos para el desarrollo de proyecto	
Hardware	Software
Notebook Acer Intel Pentium 4GB RAM-500GB DD 13,3"	Unity ver. 4.6 free
Monitor LED 16"	Photoshop CC home edition (free)
Mouse Óptico NetScroll USB	frootLoop Studio trial versión
Tableta Digitalizadora	SDK Android + API
Servidor Web	Windows 10 Pro.

Tabla 1 – Hardware y software necesarios para la implementación de la solución

De los recursos hardware anteriormente mencionados y que se requieren para la implementación del videojuego, El desarrollador cuenta con el Notebook, el monitor y el mouse óptico; cabe destacar que el resto de los dispositivos con los que no cuenta el desarrollador como la tableta digitalizadora y el servidor web (el cual cuenta con el personal capacitado para administrar de manera correcta y eficiente los servidores) se

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

encuentran disponibles en la carrera de Ingeniería civil en Informática de la universidad. De los recursos de software necesarios, todos los nombrados cuentan con licencia gratuita o son de libre distribución.

## **2.2 Factibilidad operativa.**

Para los efectos de evaluación del proyecto, la factibilidad operativa corresponde al estudio del grado de aceptación y apoyo por parte de los usuarios y directivos a la implementación de un videojuego y las nuevas directrices que ello conllevará.

La inquietud de la necesidad de un nuevo videojuego para esta casa de estudio partió debido a su creciente volumen de estudiantes, que junto al gran interés por las ciencias de la computación, deseaban visualizar resultados tangibles y distintos de su futuro aprendizaje (programación y Metodologías).

Es por esto que el desarrollo del proyecto se realiza con miras a potenciar el interés de los estudiantes por la carrera de ingeniería civil en informática como asimismo difundir la universidad entre los estudiantes de enseñanza media que visiten el campus en actividades de difusión.

Finalmente, y según lo anteriormente descrito el proyecto se considera operativamente factible ya que no se presentan barreras a la hora de la implantación.

## **2.3 Factibilidad económica**

Este estudio espera determinar en el ámbito económico los costos y beneficios de desarrollar el proyecto, entre los cuales se encuentran el costo de hardware y software, costo mano de obra, costo de instalación, costo de operación, costo de mantención y beneficios intangibles.

A continuación se detalla el valor del hardware y software necesarios para el desarrollo del proyecto, y así determinar cuál será el VAN (Valor Actual Neto), el cual nos permite calcular el valor presente de un determinado número de flujos de caja futuros, originados por la inversión.



Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

## 2.4 Inversión del proyecto

### 2.4.1 Inversión para el desarrollo

#### Costo de mano de obra

Se requiere de un Ingeniero Civil en Informática para llevar a cabo las labores de análisis y programación del proyecto. Además, se requiere un Diseñador Gráfico, el cual se encargará de transformar los bocetos del juego en interfaces aplicables al proyecto.

Los detalles de costos según horas de trabajos necesarias se pueden observar en la tabla 2.

Cantidad	Item	Total Mensual	Total Proyecto
20 horas a la semana	Valor de \$6400 por hora de un Ingeniero Civil en Informática	\$512.000 CLP	\$2.048.000 CLP
20 horas totales	Valor de \$4.500 por hora de un Diseñador Gráfico	\$90.000 CLP	\$90.000 CLP
<b>Valor total por Proyecto</b>			<b>\$2.138.000 CLP</b>

Tabla 2 - Cuadro valor total costo mano de obra.

De la Tabla 2 se puede inferir que durante los 4 meses de desarrollo del proyecto, el valor total del Ingeniero es de \$2.048.000 y del diseñador es de \$90.000.

<b>Inversión de Análisis, Diseño y Construcción</b>	
Item	Valor Total
Notebook Acer Intel Pentium 4GB RAM-500GB DD 13,3"	\$299.990 CLP
Smartphone Lg g3 beat, Equipo aportado por el ingeniero.	\$109.990 CLP
Servidor Web (anual)	\$29.990 CLP
Herramientas de programación (Unity, GenyMotion Emulator, Android Developers Tools y SDK Android + APIS: Herramientas de uso libre.	\$ 0CLP
<b>Valor Total de Análisis, Diseño y Construcción</b>	<b>\$409.980 CLP</b>

Tabla 3 - Cuadro valor total de Análisis, Diseño y Construcción

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

En la Tabla 2 se obtiene un total de \$2.138.000 CLP para el desarrollo del proyecto, cabe destacar que el sueldo del desarrollador que es \$2.048.000 CLP se descuenta, ya que el Ingeniero Civil en Informática es un estudiante que se encuentra realizando su proyecto de título. Además, el costo del diseñador gráfico (\$90.000 CLP) es asumido por el estudiante que desarrolla el proyecto.

De la tabla 3, en la que se obtiene un total de \$409.000 CLP, se descuenta el Smartphone y los computadores utilizados para el desarrollo de este sistema la suma resultante es de \$29.990 CLP, que es el monto por el pago del servidor web, el cual también es descontado, ya que éste es aportado por la carrera.

#### 2.4.2 Inversión puesta en marcha del proyecto

En este apartado se calculará el valor total que deberá invertirse para poner en marcha el proyecto.

En la Tabla 4 se presentan los valores correspondientes a recursos materiales, hardware, servicios y capacitación necesarios para la implantación del sistema.

<b>Inversión puesta en marcha del proyecto</b>	
<b>Item</b>	<b>Valor</b>
<b>Servicios</b>	
Google Play Developer console	\$18.000 CLP
<b>Total inversión puesta en marcha</b>	\$18.000 CLP

Tabla 4 – Inversión Puesta en Marcha

En la Tabla 5 se presenta la inversión total del proyecto, que corresponde al total de los puntos 2.4.1 y 2.4.2.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

<b>Inversión Total del Proyecto</b>	
<b>Item</b>	<b>Valor</b>
Total Inversión de Desarrollo	\$ 90.000 CLP
Total Inversión Puesta en Marcha	\$18.000 CLP
Inversión Total del Proyecto	\$108.000 CLP

Tabla 5 - Cuadro Inversión Total del Proyecto

## 2.5 Ingresos del proyecto

El beneficio obtenido que se puede mencionar al publicar este videojuego en la plataforma Google Play y gracias al sistema de pago por publicidad AdMOV, que genera ganancias por cada descarga y cada clic en publicidad que se ingresa al juego o app publicado.

En la Tabla 7 presenta el monto ahorrado que constituye el beneficio del proyecto.

<b>Cantidad</b>	<b>Ítem</b>	<b>Total</b>
1 clic	Descargas desde google play	\$ 3 CLP
<b>Valor total estimación 5000 descargas (1 mes aproximadamente)</b>		\$15.000 CLP
<b>Valor total anual (estimación 35000 descargas)</b>		\$ 105.000 CLP

Tabla 6 – Valor estimación de descargas por dispositivo

## 2.6 Costos del proyecto

En la Tabla 7 se presentan los costos del proyecto. Éstos se estiman por lo que necesita el cliente mantener el videojuego en google play listo para descargar.

<b>Descripción</b>	<b>Valor</b>
Google play Services (servicio de contrato con un solo pago)	\$0 CLP
Costo Total anual del proyecto	\$0 CLP

Tabla 7 – Costos del proyecto

## **2.7 Flujo Neto de Caja y VAN del proyecto**

En la Tabla 8, se procede a calcular el Flujo Neto de Caja, tomando como en base el costo total del proyecto y el ahorro total anual en cuanto a las horas de trabajo.

Para este cálculo, se toman en cuenta los siguientes factores:

- Tiempo de vida útil del proyecto se estima en 3 años.
- Se evalúa este proyecto con una tasa de descuento del 10%.
- La tasa de impuesto a la renta es de 22,5%.

Descripción	Año 0 (\$)	Año 1 (\$)	Año 2 (\$)	Año 3 (\$)
(+)Ingresos	0	105.000	105.000	105.000
(-)Costos	0	(0)	(0)	(0)
Total	0	105.000	105.000	105.000
(-)Impuesto (22,5%)	0	23.625	23.625	23.625
(-)Inversión	(108.000)	0	0	0
<b>Total</b>	(108.000)	81.375	81.375	81.375

Tabla 8 - Cuadro Flujo Neto de Caja

A continuación, se procede a calcular el VAN (Valor Actual Neto), procedimiento que permite obtener el valor presente de un determinado número de flujos de caja futuros, en base a una inversión (Jansen Molina, 1999). Como ya se dijo, se estima una tasa de descuento de 10%.

El cálculo del VAN se realiza de acuerdo a la siguiente fórmula:

$$VAN = -I + \sum_{n=1}^N \frac{Qn}{(1+i)^n}$$

Figura 1 - Fórmula de cálculo del VAN

En donde:

$I$  = Inversión del proyecto

$n$  = Número de períodos

$Qn$  = Flujos de caja estimados

$i$  = interés

A continuación se muestra el resultado que se obtiene:

$$VAN = -108.000 + (81.375/(1 + 0,10)^1) + (81.375/(1 + 0,10)^2) + (81.375/(1 + 0,10)^3)$$

$$VAN = -108.000 + 73.977,27 + 67.252,06 + 61.138,24$$

$$VAN(10\%) = 94.367,57$$

## 2.8 Conclusión de la factibilidad

Desde el punto de vista de la factibilidad técnica, aunque la mayoría del hardware lo posee el desarrollador, este es fácil de adquirir, por lo cual técnicamente es factible cualquier desarrollo de videojuegos. Además, desde el ámbito económico la solución posee un VAN positivo, esto significa que la solución genera ingresos para la

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

compañía desde el año 2. Por otro lado, operacionalmente la aplicación genera aceptación por los usuarios, lo cual aumenta la valoración y expansión, ayudado de una forma a popularizar el videojuego y todo lo que esto contenga (publicidad a la universidad y alrededores).

Tomando en cuenta también los beneficios intangibles del proyecto, se adiciona la utilidad que la Universidad Del Bío-Bio puede sacar del videojuego, utilizándolo en Difusión, Universidad Abierta y promocionar su nombre en plataformas móviles (como apoyo a la aplicación "yo soy UBB").

Dado los aspectos anteriormente descritos, se puede concluir que la solución es factible de llevar a cabo en su totalidad.

# Capítulo III

## Marco Teórico

---

### 3.1 Orientación al mundo 2D

El mundo como lo conocemos, el cual es visible en un sistema de coordenadas tridimensional, difiere mucho en la perspectiva que tiene un videojuego al desplegarse en una pantalla de Computador o dispositivo móvil. Es por esto que el objetivo del diseñador del videojuego es desplegar el entorno logrando que el jugador pueda sentir la sensación de la dimensión faltante, para esto se utilizan diversas técnicas de estética y jugabilidad.

Es por eso que este capítulo se enfocará en las técnicas utilizadas para crear y simular la dimensión faltante dentro del videojuego, y lograr un resultado de un mundo parecido al nuestro.

### 3.1.1 Orientación y Sistema de Coordenadas

Un concepto primordial al momento de describir un entorno, es conocer acerca de sus dimensiones físicas, para las cuales existen 3 y están denominadas como ejes X, Y, Z en el plano cartesiano, igualmente denominadas alto, ancho y profundidad. Es por esto que es muy importante conocer cuál es el sistema de coordenadas con que trabaja un entorno 2D/3D en particular.

Generalmente cada dimensión en un entorno o sistema de coordenadas está definida por un eje Cartesiano, por lo que comúnmente se conocerá al eje X como ancho, el eje Y como alto y finalmente la profundidad por el eje Z.

En la Figura 1 se ilustra cómo sería el orden de los ejes desde un punto de vista frontal como tendría una persona en un entorno tridimensional.

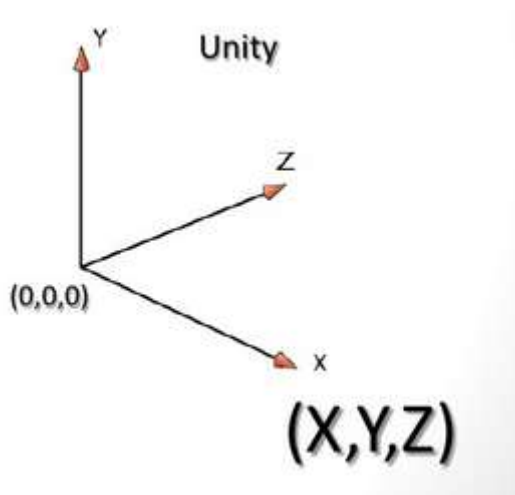


Figura 1, Sistema de coordenadas

Para nuestra aplicación, el sistema de coordenadas será visto de una forma un poco distinta a como estamos acostumbrados en la vida diaria, ya que un enfoque bidimensional solo es así al momento de apreciarlo. En simples palabras, el sistema se verá modificado en el eje z, ya que la profundidad no se verá afectada como en un sistema común, trabajando de forma ortográfica, es decir, no importará la distancia del objeto en el eje Z con respecto del centro (0,0,0) su tamaño no disminuirá como si ocurriría en un entorno 3D.



Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

Cabe mencionar que el Motor Gráfico Unity utiliza el sistema ortográfico al momento de crear videojuegos 2D y así mantener su sistema de referencias intacto en ambos tipos de vistas.

Una de las ventajas que ofrece el sistema de coordenadas en el desarrollo de videojuegos es que existen dos tipos de ubicaciones en el entorno, una denominada "Space World" que es la que logra ubicar a todos los objetos dentro del espacio a trabajar, donde comparten el punto de origen y las dimensiones pertinentes. El otro tipo de ubicación es el denominado "Local Space" en el cual tiene un punto dentro del "Space World" pero que tiene un punto de origen propio que normalmente se encuentra en el centro del objeto y que puede ser utilizado a conveniencia por el desarrollador.

En la figura 2 se describe la definición de ambos tipos de ubicaciones en el espacio. [1]

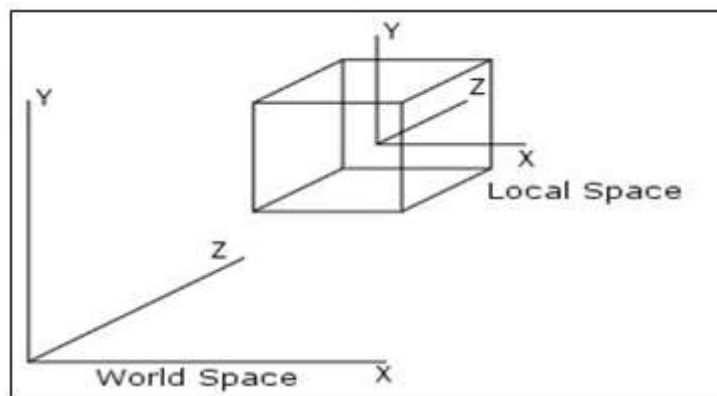


Figura 2, descripción de world space y local space

### 3.1.2 Transformaciones Geométricas

Una pregunta común al momento de ver y disfrutar un videojuego es ¿Cómo se hace para que el videojuego cobre vida? Bueno, esto ocurre gracias a las transformaciones geométricas que son las encargadas de lograr el movimiento de la interfaz dentro del videojuego, como ya sabemos cada objeto tiene una ubicación

dentro de un sistema de coordenadas, análogamente, también podemos deducir que estos objetos pueden tener así mismo transformaciones geométricas.

Pero ¿Qué son las transformaciones geométricas?

*“Las transformaciones geométricas son la o las operaciones geométricas que permiten crear una nueva figura a partir de una previamente dada. A esta nueva figura se le llama la homóloga de la original” [2].*

Podemos clasificar éstas transformaciones en 3 distintos tipos: Traslación, Rotación y Escala y pueden ser descritas como ecuaciones matemáticas.

A continuación se detallarán cada una de estas operaciones:

### 3.1.2.1 Traslación

La traslación es una de las transformaciones más básicas que existe, ya que consiste en mover el objeto dentro del sistema de coordenadas. Esta operación se lleva a cabo mediante la adición de un vector (Vector 3D para una traslación en 3 dimensiones y 2D para una traslación en 2 dimensiones) el cual mueve el objeto hacia el destino. Por ejemplo, se tiene el punto A (1,4) el cual corresponde a la posición actual del objeto a trasladar, además de P (3,4) que sería el vector de traslación.

Por lo tanto, el punto de destino de nuestro ejemplo sería la adición de las unidades de cada eje del punto P a nuestro punto A, dando como resultado un punto A' (4,8).

La figura 3 describe como sería la traslación de un objeto en cada eje ordenado del sistema de coordenadas.

$$\begin{aligned}x' &= x + d_x \\y' &= y + d_y \\z' &= z + d_z\end{aligned}$$

Figura 3, operaciones para realizar una traslación en un sistema tridimensional.

### 3.1.2.2 Rotación

La rotación es una de las transformaciones geométricas más difíciles de realizar, ya que éstas dependen de la posición inicial del objeto. Consiste en girar el cuerpo u objeto con respecto a un punto de eje coordenado en un ángulo dado, que es determinado a través de identidades trigonométricas.

La figura 4 describe la rotación de un objeto, y la ubicación de sus vértices antes y después de su rotación.

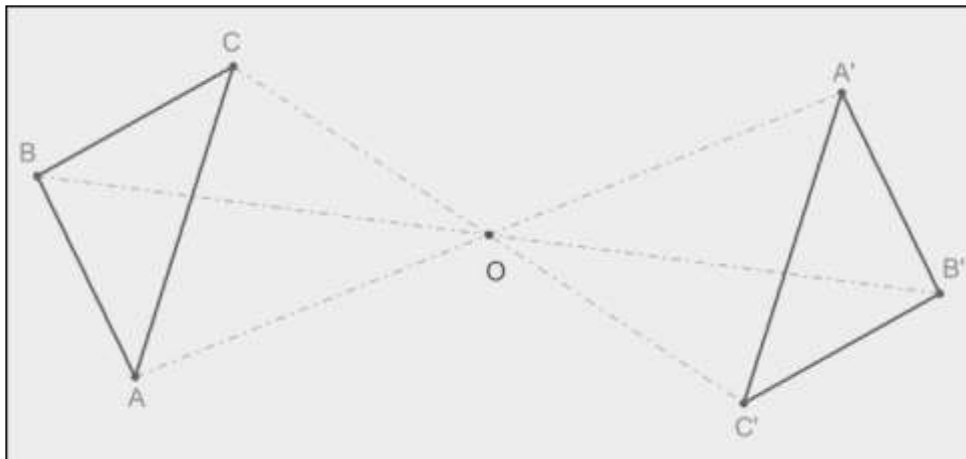


Figura 4, ejemplo de rotación en un eje de 2 dimensiones.

En la figura 5 se describe la fórmula para Rotación en todos los ejes cartesianos.

En el eje X:  $X' = X$ ;  $Y' = Y * \cos \theta - Z * \sen \theta$ ;  $Z' = Y * \sen \theta + Z * \cos \theta$   
 En el eje Y:  $X' = X * \cos \theta + Z * \sen \theta$ ;  $Y' = Y$ ;  $Z' = -X * \sen \theta + Z * \cos \theta$   
 En el eje Z:  $X' = X * \cos \theta - Y * \sen \theta$ ;  $Y' = X * \sen \theta + Y * \cos \theta$ ;  $Z' = Z$

Figura 5, Fórmula para rotación.

### 3.1.2.3 Escala (Redimensionado)

El redimensionado de un objeto es una de las transformaciones geométricas más notorias al momento de ser aplicada, corresponde a una deformación del objeto en sí y consiste en escalar una coordenada por un valor constante, si este valor es mayor a 1, el objeto aumentará su tamaño, pero a su vez si este es menor a 1 y mayor a cero el objeto disminuirá su tamaño.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

Existe también en este aspecto un escalado no uniforme, otorgándole distintos valores a los componentes del eje de coordenadas, deformando el objeto a escalar. En la figura 6 se visualiza el escalado de un objeto en un sistema bidimensional.

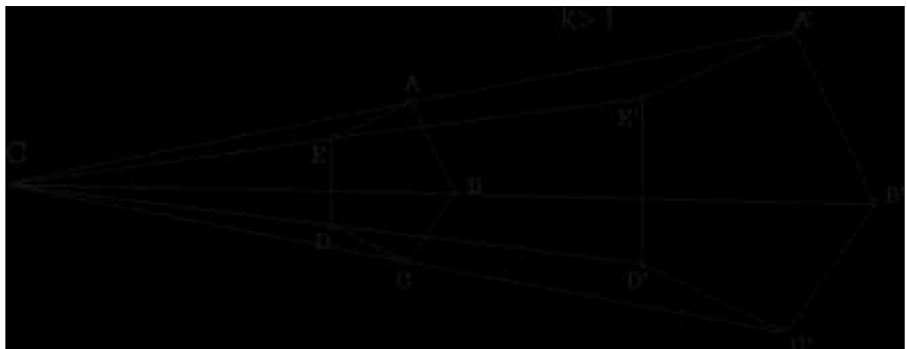


Figura 6, ejemplo de escalada bidimensional.

En la figura 7 se ilustra la ecuación para el escalado en los 3 ejes cartesianos.

$$X' = X * S_x \quad Y' = Y * S_y \quad Z' = Z * S_z$$

Figura 7, Fórmula para Escalado.

Finalmente, para ejemplificar lo explicado anteriormente, la Figura 8 muestra una sucesión de operaciones de transformación sobre un objeto dentro de un espacio 3D [3]:

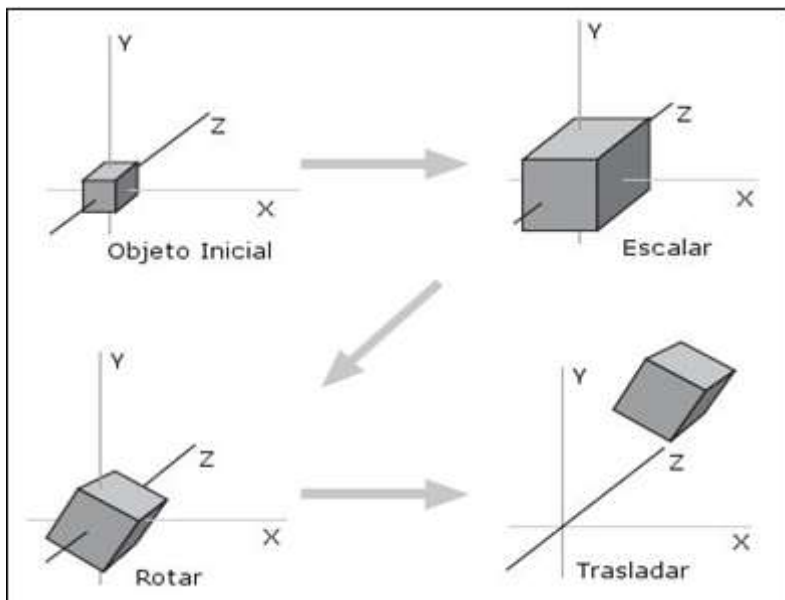


Figura 8, Operaciones de transformación sucesivas sobre un objeto

### 3.1.3 La Cámara

La cámara es uno de los componentes que le dan vida a un videojuego, ya que se puede comportar de diversas formas, puede ser la vista de un personaje en el videojuego, puede ser la visión del narrador de la historia, con un perfil en 3ra persona o cualquier vista de algún actor.

La cámara puede ser posicionada en cualquier punto del espacio cartesiano y configurada para visualizar el contenido de dos modos distintos:

- **Perspectiva:** este tipo de cámara es la utilizada normalmente para un videojuego 3D, el cual requiere que la cámara tenga un campo de visión amplio y limitado tal como sería la visión humana. El ángulo de visión (FOV) puede ser controlado para modificar el tipo de vista, por lo que ampliando este ángulo la cámara pueda simular un zoom dentro del entorno [3].

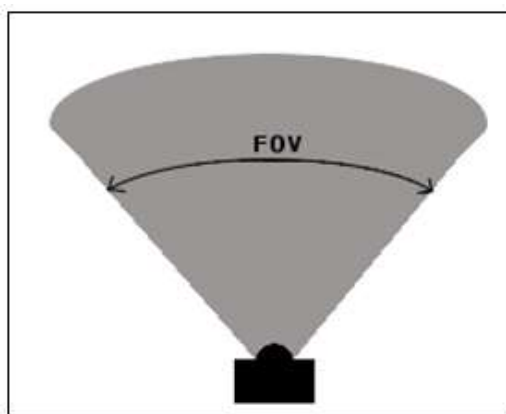


Figura 9, ángulo de visión (FOV) de cámara de Perspectiva

- **Ortográfica:** Este tipo de cámara ha sido incorporada en las últimas versiones de Unity para compensar el modelado de videojuegos 2D, consiste en eliminar un eje de perspectiva para renderizar sin profundidad. en esta ocasión, la cámara posee una profundidad (Size) y bordes definidos, que son rectos y paralelos entre si [3].

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

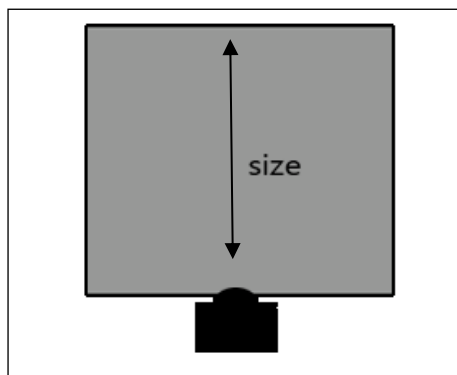


Figura 10, tamaño y forma de cámara ortográfica.

La figura 11 muestra la diferencia entre los dos tipos de cámara.



Figura 11, comparativa de visión en 2D tercera persona y 3D primera persona.

### 3.1.4 Los objetos Renderizables

En los videojuegos, todo lo que vemos, es conocido como interfaz gráfica y una forma coloquial de nombrarles es como "Sprites Renderer", que es la capacidad de un objeto de visualizarse como una imagen o animación en un entorno bidimensional. Los objetos renderizables en simples palabras, son más que nada una imagen cargada para ser visualizada, ya sea el personaje principal, suelo, las nubes de fondo o cualquier forma visible [4].

En la figura 12, se muestra el sprite de un personaje 2D, y su renderizado de animación.



Figura 12, animación de megaman desplazándose.

### 3.2 Los Motores de Videojuegos

Desde tiempos inmemorables, el ser humano ha sido dotado de la habilidad de relatar distintas historias que se van conociendo de generación en generación, desde cuentos hasta mitología, también con el fin de entretener, impresionar o crear mundos imaginarios que abren el mundo a la ficción y utopía. Esto sucede gracias a que el hombre tiene la capacidad de fomentar su imaginación, gracias a los sentidos.

En un videojuego ocurre algo de forma muy similar, el crear un mundo de fantasía donde podemos sumergir nuestros sentidos, embriagando nuestra curiosidad de la historia relatada y el posible desenlace de la problemática o aventura a resolver es la que nos envuelve completamente y es justamente eso lo que desean lograr los desarrolladores de videojuegos.

El principal problema consiste en la diferencia de un videojuego de un mito o leyenda, ya que el videojuego está completamente adentrado en el uso de gráficos, música y dificultad propia para utilizar todos los sentidos y éste es un desafío para el usuario, es por eso que un juego es muy difícil de llevar a cabo, entonces ¿Qué ayuda tengo para lograr desarrollar un videojuego?

Las Empresas de desarrollo de videojuegos tenían la misma inquietud, Es por esta razón, que la industria de los videojuegos ha buscado soluciones a este problema, desarrollando software especializado, que facilite la construcción de un videojuego. ¿La solución?, Los motores de videojuegos.

El siguiente capítulo a presentar, tiene por objetivo describir que es un motor de videojuegos y cuáles son los más conocidos en desarrollo 2D.

### 3.2.1 Definición

Un motor gráfico, rigurosamente hablando, se refiere a un conjunto de librerías específicamente desarrolladas para el renderizado de imágenes, incorporación de sonidos, interfaces y herramientas de desarrollo y eventos en la construcción de un videojuego.

El objetivo de crear un motor gráfico es la ventaja de tener herramientas reutilizables de rápido acceso sin tener que desarrollarlas cada vez que se requiera su uso, ésta técnica ayuda a programadores, diseñadores y músicos a ahorrar tiempo en el trabajo que deben realizar y enfocarse solamente en la lógica del juego a realizar, el modelado y diseño de personajes, escenarios y otros objetos incorporados en el juego, pudiendo profundizar los aspectos técnicos que hacen que el videojuego sea único.

En la figura 13, se especifica la arquitectura de un motor gráfico de forma general, que logra procesar componentes en tiempo real [5].



Figura 13, esquema de arquitectura de un motor gráfico.



### 3.2.2 Historia

En los inicios de la interfaz gráficas, tanto para los videojuegos como para la computación, un videojuego era desarrollado por un número muy reducido de personas, incluso con un solo programador, puesto que la dificultad de éstos no era muy destacada y el tiempo de desarrollo del proyecto no era muy amplio.

Sin embargo, con el pasar de los años, los equipos de trabajo fueron aumentando, y con esto la calidad de los juegos fue aumentando significativamente, tanto en estética como en complejidad; los equipos de trabajo comenzaron a incorporar a diseñadores gráficos, sonidistas, guionistas, entre otros.

A pesar de que los equipos de trabajo iban creciendo, al momento de crear los juegos, construían todo desde cero, derrochando recursos y tiempo en cada nueva producción o entrega.

Tiempo después, con la invención de los motores gráficos, que llegaron para resolver el problema de pérdida de tiempo y recursos, adquiriendo la capacidad de reutilizar componentes de proyectos anteriores o componentes en común que comparten la gran mayoría de los videojuegos, como lo es la composición de la física e interacción.

Ahora la reutilización de componentes de software es un hecho, se ahorra tiempo y costos en la realización de tareas que son comunes y habituales en el desarrollo de un videojuego. Es más, muchas empresas dejaron de producir videojuegos con el fin de especializarse en la construcción de motores de juego, para luego venderlo a otras empresas del rubro.

Hoy en día, para desarrollar un juego 2D destacan 4 motores gráficos por su extensa comunidad activa y por el futuro prometedor que se avecina para ellos, para que cualquier persona interesada en crear un videojuego, no tenga barreras en su desarrollo [6]:

- **Shiva 3D:** Shiva 3D es uno de los motores gráficos con un futuro más que prometedor, en el cual se han creado más de 8000 juegos. A pesar de ser un motor gráfico 3D, éste responde de buena forma en el desarrollo de videojuegos 2D.

Una de las características fuertes que posee Shiva 3D es que es uno de los motores gráficos con mayor compatibilidad, logrando llegar a una gran lista de dispositivos exclusivos, como Nintendo WiiU y Nintendo 3DS [7].

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

- **GameMaker Studio:** Este motor gráfico es uno de los más populares dentro del desarrollo de juegos para IOS, se caracteriza por ser una herramienta muy flexible e intuitiva para los desarrolladores, facilitando la entrada de las personas que no son tan familiarizadas con la herramienta o el lenguaje de programación.

Este motor está desarrollado totalmente para creaciones de tipo 2D, por lo que todas herramientas entregadas por el motor pueden ser aplicadas en el desarrollo o entrega de un juego.

Como último, GameMaker entrega una relación directa en la creación del juego y su monetización, haciendo mucho más fácil la forma en que este logre recibir ingresos a través de publicidad [8].

- **Unity:** Unity es uno de los motores más populares en el día de hoy, ya que es el que mantiene la comunidad de desarrolladores más activa, logrando así que los programadores y desarrolladores traspasen el umbral de novatos a profesionales en menos tiempo que con cualquier otro motor gráfico.

Hoy en día, Unity cuenta con dos tipos de motores incorporados en su Software, un motor 3D y otro 2D. El motor 2D fue incorporado gracias a la petición de un gran número de Desarrolladores de la comunidad oficial, logrando su objetivo en la versión 4.3 oficial [9].

- **Torque 3D, Torque 2D y iTorque 2D:** Este motor gráfico es uno de los menos usado de los nombrados anteriormente, y eso se debe a que está orientado a desarrollo un poco más serio, pero logrando juegos meramente casuales, por lo que termina siendo utilizado para hacer juegos one-hand o indie games.

Una de las ventajas que posee Torque en todas sus versiones, es la versatilidad de su entorno gráfico, acomodando totalmente al programador del proyecto [10].

En definitiva, existe una amplia gama de posibilidades, para todos los gustos y tipos, pero que al fin y al cabo todos con un objetivo en común, facilitar la compleja tarea de desarrollar un videojuego.

### **3.3 Unity 2D**

El camino hacia un juego exitoso consiste en aplicar todo lo aprendido hasta el momento, es por eso que en este capítulo especificaremos algunas características de Unity en su versión 2D, cabe destacar, que ésta herramienta posee varios conceptos, que es recomendable conocer y manejar antes de comenzar a desarrollar un proyecto, ya que éstos definen la forma de trabajo de Unity2D.

#### **3.3.1 Aspectos Generales**

Desde su creación, Unity ha tenido un sinnúmero de cambios, con el objetivo de entregar la mejor calidad al usuario final, es por esto que con el correr de los años se han incorporado distintas librerías, herramientas y capacidades a este motor gráfico; en las siguientes líneas Explicaremos los componentes básicos y algunos componentes técnicos pero esenciales para desarrollar un proyecto en 2 dimensiones.

Para lograr esto, dividiremos nuestra explicación de la siguiente forma.

- Componentes Básicos:
  - Interfaz de Usuario de Unity
  - Menú de Aplicaciones
  - Botones Control
  - Botones de Reproducción
  
- Componentes Técnicos para 2D:
  - Sprite Renderer
  - Animation/Animator
  - Canvas

#### **3.3.2 Componentes Básicos de Unity**

Se conoce como componentes básicos de Unity a toda herramienta otorgada por el fabricante, que facilite el uso de un videojuego tanto para 3D como para 2D, en el cual se tiene una versión similar de una herramienta 3D, pero adaptada para una visualización 2D. Éste tipo de componentes difieren solo en una característica de los componentes técnicos y es el objetivo por el cual han sido creados; Principalmente

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

Unity había sido creado para desarrollar videojuegos en 3D, entonces se le denomina componentes Básicos a las herramientas entregadas para desarrollar este tipo de juegos, y componentes Técnicos son los que solamente sirven para un tipo de desarrollo ya sea 2D o 3D (creación de terrenos en 3D por ejemplo) pero que no tendría sentido utilizar en el ámbito 2D (y viceversa).

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

### 3.3.2.1 Interfaz de Usuario de Unity

En este segmento veremos cómo se compone la interfaz de usuario de Unity. Existen actualmente 4 áreas de la interfaz de usuario, enumeradas en la Figura 14.

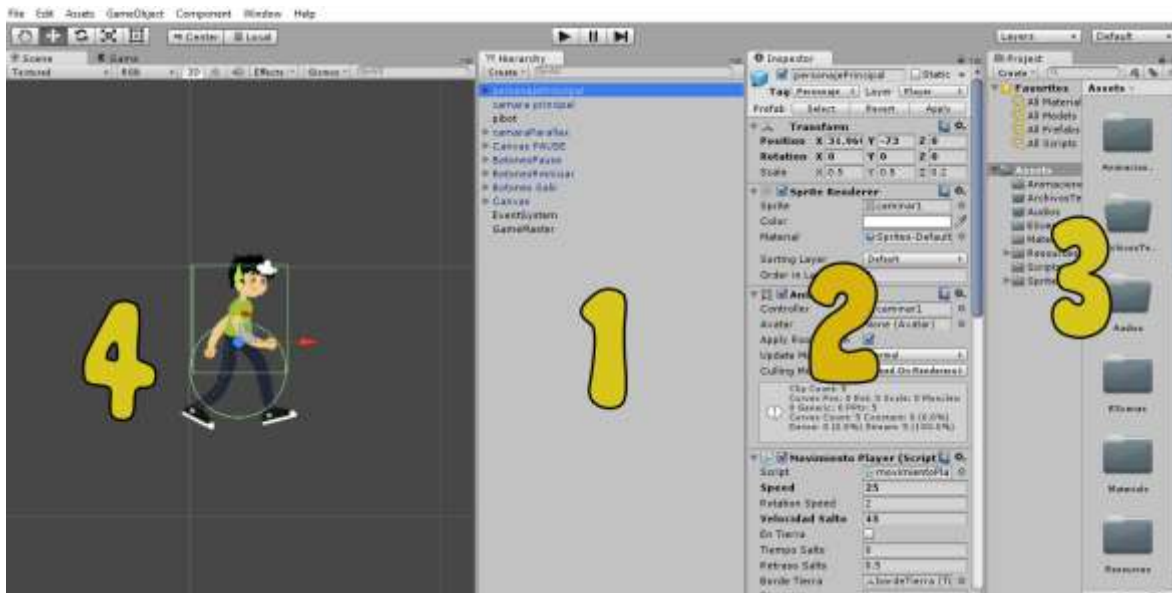


Figura 14, Enumeración de áreas en Unity.

1. Vista de Jerarquía: En este campo se visualizan todos los componentes que han sido creados para componer la escena actual.
2. Vista de Inspección (Inspector): En este campo se visualiza todo lo que compone un objeto de juego, siempre y cuando se encuentre uno seleccionado, ya que en caso contrario no visualizará nada. En simples palabras, Inspector visualizará las propiedades de un objeto seleccionado.
3. Vista del Proyecto: En esta vista se visualiza la arquitectura de carpetas, objetos y componentes que conforman el proyecto creado.
4. Vista de Escena: En esta vista es donde se crea la arquitectura del juego, donde se puede interactuar directamente con los componentes creados anteriormente, y donde se desarrolla visualmente cada escena, que luego formará parte del videojuego. La vista de escena es también donde se editan los componentes gráficos en un videojuego 2D y donde se crean los terrenos en un videojuego 3D.

### 3.3.2.2 Menú de aplicaciones

A continuación se muestra el menú de opciones de Unity que se encuentra en la parte superior izquierda. A lo largo de este capítulo se irán mostrando las utilidades de cada una de las secciones de este menú.

En la figura 15 se ilustra el menú de aplicaciones de Unity.

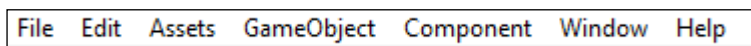


Figura 15, menú de aplicaciones de Unity

### 3.3.2.3 Botones Control

Bajo el menú de Aplicaciones de Unity, se puede visualizar los botones de control, con los cuales se pueden hacer distintas interacciones dentro de la vista de Escena.

En la Figura 16 se puede visualizar la barra con botones de control, que serán explicados en las siguientes líneas.

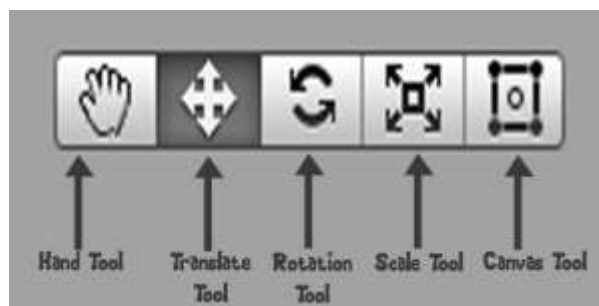


Figura 16, Botones de control

- Hand Tool: Esta herramienta nos permite movernos dentro de la vista de escena sin mover ningún objeto. Puede acceder a ella de forma rápida presionando la tecla Q.
  - ALT nos permitirá rotar.
  - CTRL nos permitirá hacer zoom
  - SHIFT incrementa la velocidad de movimiento mientras usas la herramienta.
- Translate Tool: Esta Herramienta nos permite mover cualquier objeto al ser seleccionado dentro de la escena. Puede acceder a ella de forma rápida presionando la tecla W.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

- Rotate/Rotation Tool: Esta Herramienta es similar a la herramienta Translate, permitiéndole al Desarrollador rotar el objeto dentro de los ejes X, Y, Z sin alterar su tamaño.
- Scale Tool: Esta Herramienta permite escalar un objeto dentro de la escena, esto también puede realizarse desde el campo de Inspector, en el cual aparecerá la información de tamaño, posición y rotación del objeto seleccionado.
- Canvas Tool: Esta herramienta ha sido incorporada desde la versión 4.6 y viene a reparar un error en el desarrollo de interfaces, logrando crear botones de menú de forma mucho más sencilla, anteponiéndose siempre a la cámara principal, sin importar su ubicación.

### 3.3.2.4 Botones de Reproducción

En Unity se puede ejecutar el juego sin salir del editor, lo que es una estrategia muy atrevida para los diseñadores que están construyendo niveles y los desarrolladores que están añadiendo nuevas mecánicas de juego.

En la Figura 17 se ilustran los 3 Botones a utilizar al momento de ejecutar un proyecto.



Figura 17, Botones de Reproducción en Unity

El botón de la izquierda es el botón de play, el cual ejecuta el proyecto antes de ser compilado, logrando encontrar errores de cualquier tipo antes de tener que ejecutarlo en la plataforma objetivo.

El botón central es el botón de Pause, con el cual se puede pausar la escena actual, ya sea para cambiar variables o ante cualquier eventualidad.

También se puede lograr el mismo efecto dentro del videojuego, alterando el campo TimeScale (que es el encargado de alterar la velocidad de reproducción del juego) igualándolo a cero.

El botón de la Derecha es botón de frames, encargado de avanzar el juego frame a frame, para revisar detalles mucho más minuciosamente, especialmente cuando se habla de colisiones exactas y rangos de energía. Este botón es de completa

utilidad cuando se desarrolla un videojuego de peleas, ya que las colisiones deben ser muy exactas para minimizar el rango de errores.

Una característica muy útil de estos botones en general, es la capacidad de cambiar variables en el campo Inspector mientras ejecuta el juego y todo valor que ha sido cambiado, al momento de detener la reproducción, vuelve a su estado original. De esta forma se puede probar valores sin alterar su lógica principal, optimizando la jugabilidad en el live-testing.

### 3.3.3 Componentes Técnicos Unity 2D

En esta sección se visualizarán los componentes que diferencian a Unity 2D de Unity3D, y que logran comprometer la jugabilidad de un videojuego, optimizándola al máximo.

#### 3.3.3.1 Sprite Renderer

El renderizado de imágenes vs el renderizado de texturas es la diferencia principal entre un videojuego 2D y otro 3D, es por eso que Unity incorpora Sprite Renderer en la versión 4.3 de Unity por primera vez, la misma versión que incorpora todas las herramientas de apoyo a desarrollo de juegos 2D.

El renderizado de imágenes opta por un ahorro de memoria significativo, ya que omite un eje cartesiano de gráficos y con ello también se omite la creación de materiales y texturas que se le agregaban a componentes creados por el motor gráfico (esferas, cubos y planos).

La Figura N°16 ilustra el componente Sprite Renderer, y todos los campos a editar para hacer del renderizado un poco más flexible y no se limite solamente a la carga de imágenes.

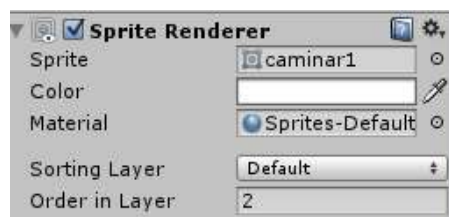


Figura 18, Componente Sprite Renderer



Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

También se incorpora una jerarquía de capas (Layers) que viene a suplir la ausencia del eje faltante, asumiendo el rol de simulación de profundidad, logrando que un objeto no dependa de su ubicación con respecto a la cámara o con respecto a su world space.

### 3.3.3.2 Animation / Animator

Otra herramienta muy útil al momento de desarrollar Juegos 2D son las herramientas Animation y Animator, con las cuales se logra animar un personaje 2D con distintas técnicas.

**Animation:** Esta herramienta es la encargada de ordenar un conjunto de órdenes dentro de un objeto, éstas órdenes pueden ser los 3 principios básicos de la animación (traslación, rotación y escalado) para así lograr el efecto de movimiento en el objeto.

Esta herramienta se caracteriza por ser muy similar a una línea de tiempo.

En la Figura 19 se muestra la herramienta Animation y su incorporación con **Mecanim** (nombre que recibe el componente que logra grabar las órdenes y transformarlas en animación).



Figura 19, herramienta animation y su incorporación junto a **Mecanim**.

**Animator:** esta otra herramienta de Unity va de la mano junto a Animation, ya que es la que logra unificar los componentes generados por animation y darles objetivos específicos (a través de un script) para simular el realismo en los personajes.

La estrategia de uso de Animator es la creación de autómatas finitos, que representados a través de diagramas de estados, logran unificar todas las animaciones

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

que han sido creadas a través de animation y enviárselas al objeto a animar dependiendo de los estados del autómata.

Éste autómata generado, es el que logra controlar el personaje en tiempo real, el cual recibe parámetros del script de movimiento que interactuarán en los estados, todas las instrucciones están en la esquina inferior derecha, que luego serán los parámetros a relacionar entre los estados del personaje, logrando así más realismo al movimiento.

En la Figura 20 se ilustra la herramienta Animator y su interfaz de uso.

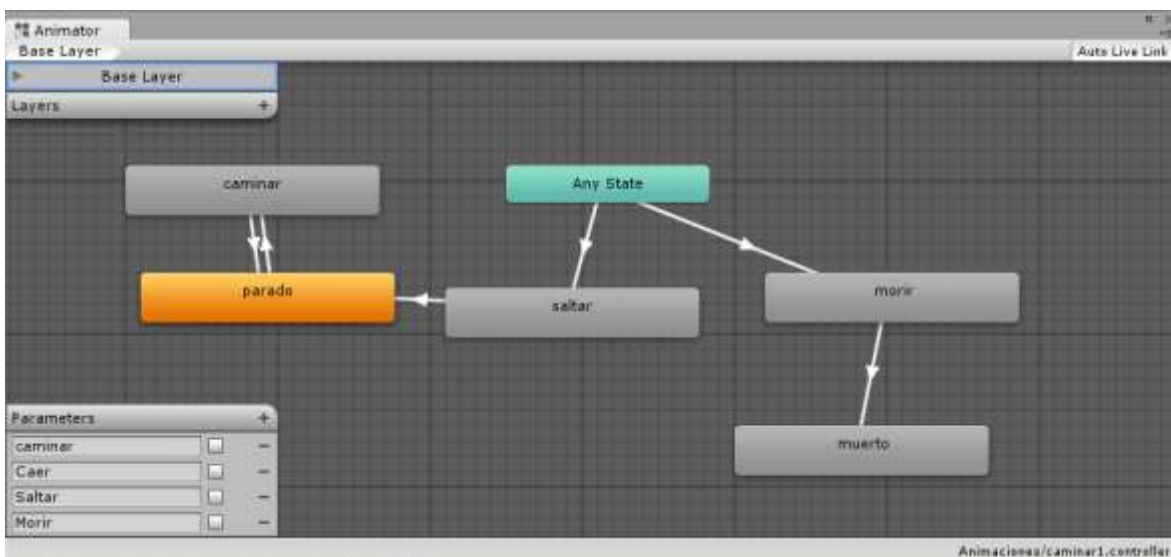


Figura 20, Herramienta Animator junto a un Diagrama de estados de animación.

### 3.3.3.3 Canvas

La última Herramienta a comentar es Canvas, la cual es incorporada en la versión 4.6 de Unity y que llega a llenar un vacío en la creación de menús, Interfaces de Usuario (UI) y su característica principal, es el auto ajuste ante distintas condiciones. Éste puede ser desplegado sobre las cámaras con un tamaño fijo y estático y que a la vez se renderice por sobre todos los objetos creados, o también puede ser con un tamaño fijo y con todos los objetos por sobre el canvas; de la misma forma puede hacerse un canvas con tamaño variable y que dependa del tamaño de la ventana para auto ajustarse, logrando un acabado más técnico y adaptativo.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

En la Figura 21 se muestra el componente Canvas y sus propiedades.

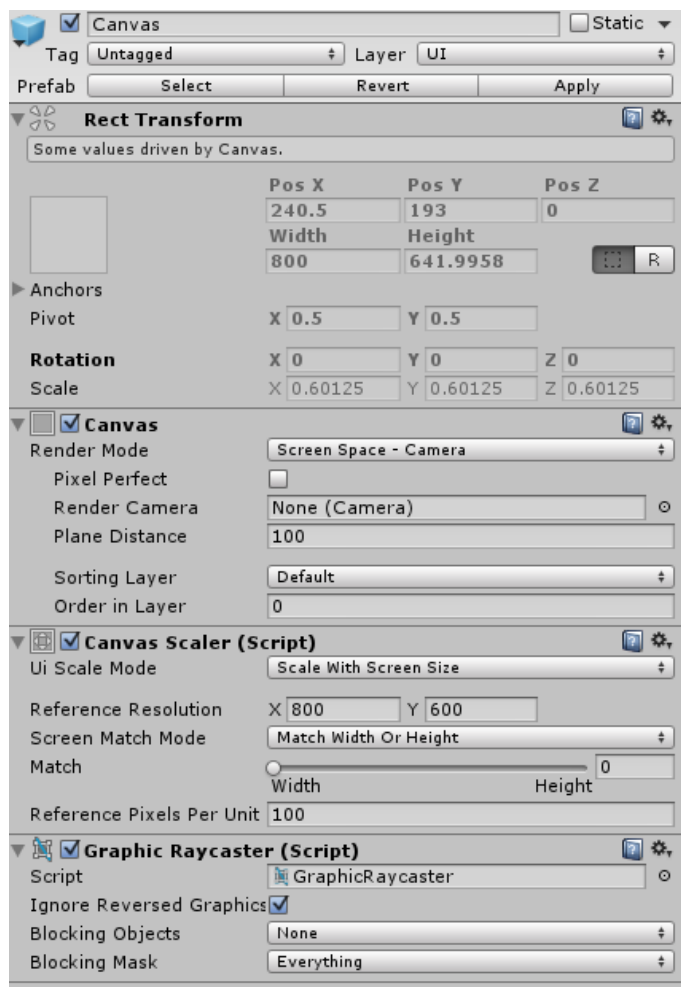


Figura 21, Canvas y sus propiedades.

### 3.4 Tipos de Videojuegos Móviles

En esta Sección se Listarán y explicarán los tipos de Videojuegos móviles con mayor realce en los últimos años:

- **Acción:** Los videojuegos de este género consisten básicamente en eliminar enemigos, sin muchos más añadidos. El contenido tiende a ser violento.
- **Aventura:** La aventura es un popular género donde el protagonista del juego debe atravesar grandes niveles, luchar contra enemigos y recoger objetos de valor. Normalmente son juegos de larga duración con un argumento extenso y complejo.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

- **Espionaje táctico, infiltración:** Juegos de acción donde lo importante no es eliminar a los enemigos, sino pasar inadvertido y sin ser descubierto, utilizando para ello todos los elementos de que dispongamos.
- **Cartas:** Los juegos de cartas hablan por sí solos, el objetivo es simular los juegos casuales de cartas reales en un dispositivo móvil. Como curiosidad cabe señalar que los juegos de cartas para una sola persona (los solitarios) han logrado un gran auge dentro de los videojuegos.
- **Carreras:** videojuegos que tiene como objetivo conducir un vehículo y lograr la mejor posición ante adversarios controlados por otras personas o por la IA.
- **One-Hand Games:** Juegos que tiene como característica su jugabilidad, la cual puede ejecutarse solamente con una mano, especial para jugarse en el transporte público como el metro o microbús. Un ejemplo de este tipo de juegos es Candy Crush.
- **Indie Games:** Juegos que tienen como característica el desarrollo independiente, donde el equipo de trabajo no supera las 4 personas, con reducido costo de producción y en el que el tiempo de desarrollo no se extiende a más de 1 año.

# Capítulo IV

## Arquitectura y Diseño

---

Algo primordial al momento de desarrollar un software, es aclarar la arquitectura que poseerá éste y el diseño de toda la lógica que utilizará.

Cabe mencionar que desarrollar un videojuego no es muy distinto de la ingeniería de cualquier otro software, ya sea un sistema web u otro tipo de tecnología.

Es por eso que en esta unidad mencionaremos que tipo de arquitectura es la utilizada para llevar a cabo el desarrollo de este proyecto, mencionando también los casos de uso y el diagrama de clases que este requiere antes de ser desarrollado.

### 4.1 Diagrama de Clases

En la industria de los videojuegos, para lograr un juego dinámico, se requiere procesar cómo será la interacción de nuestro personaje principal con su entorno, para esto se ha diseñado un diagrama de clases, el cual visualizará un boceto de la arquitectura del juego.

A continuación se ilustra el modelo conceptual del diagrama de clases, el cual se encuentra en la Figura 22.

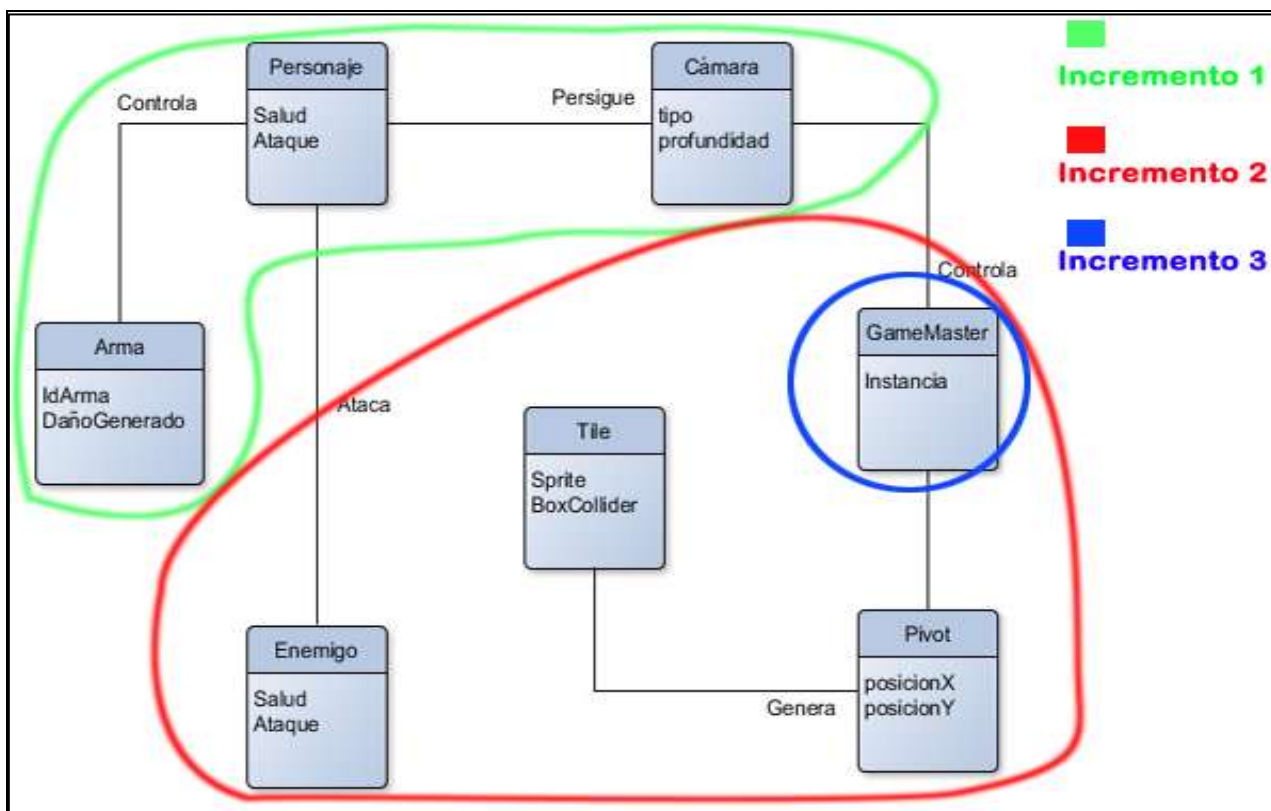


Figura 22 Diagrama de Clases de GeekNator

Posteriormente al diseño del diagrama de clases, se estima que tipos de funcionalidades son las que quieren abarcarse en el juego en si, por lo que se toman requerimientos, que posteriormente se convertirán en casos de uso, que son los que lograrán la funcionalidad de los campos de menú y del sistema de puntaje.

## 4.2 Casos De Uso

Para que nuestro videojuego ofrezca una jugabilidad dinámica y que lo caracterice, es necesario tomar los requerimientos que posteriormente lograrán la usabilidad deseada. Es para esto que se utilizan los casos de uso en un proyecto.

A continuación, en la figura 23 se muestran los casos de uso que requiere nuestro videojuego.

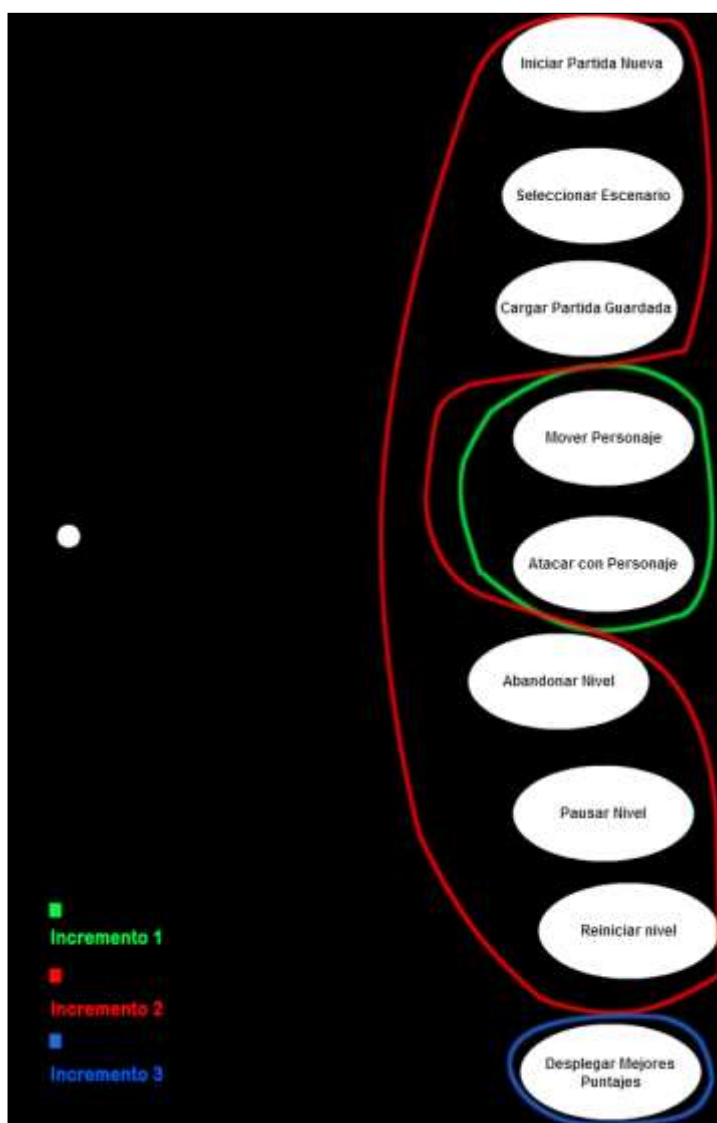


Figura 23 casos de uso de GeekNator

### 4.2.1 Especificación de Casos de Uso

A continuación se presentan las especificaciones de casos de uso, las cuales están agrupadas según el incremento, donde las tablas 9 y 10 representan el primer incremento, desde la tabla 11 hasta la 16 representan el segundo incremento y finalmente la tabla 17 representa el tercer incremento.

#### Primer Incremento:

<b>Nombre De Caso de Uso</b>	CU-01 Mover Personaje
<b>Descripción</b>	El Jugador es capaz de mover El personaje, controlando los movimientos y el salto.
<b>Precondición</b>	El Personaje debe encontrarse en un escenario cargado y en Ejecución.
<b>Secuencia Principal</b>	1.- El personaje se encuentra esperando un evento que logre moverlo en la dirección que se desee. 2.- El Jugador presiona un botón de movimiento 3.- El personaje se mueve según el botón presionado anteriormente.
<b>Errores/Alternativas</b>	-
<b>Post Condición</b>	El personaje queda a la espera de más eventos.
<b>Notas</b>	NO

Tabla 9 - Caso de uso CU-01 Mover personaje

<b>Nombre De Caso de Uso</b>	CU-02 Atacar con Personaje
<b>Descripción</b>	El personaje debe ser capaz de atacar lanzando o golpeando a oponentes (dependiendo del arma que se encuentre cargada en uno de los dos botones de ataque)
<b>Precondición</b>	El Personaje debe tener un arma cargada en uno de los botones de ataque como mínimo.
<b>Secuencia Principal</b>	1.- El Personaje está a la espera de la instrucción. 2.- El Jugador presiona uno de los dos botones de ataque. 3.- El Personaje realiza el movimiento de ataque (lanzando o abanicando un arma). 4.- El personaje queda a la espera de alguna otra instrucción
<b>Errores/Alternativas</b>	-
<b>Post Condición</b>	El personaje queda a la espera de más eventos.
<b>Notas</b>	NO

Tabla 10 - Caso de uso CU-02 Atacar con personaje



Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

## Segundo Incremento:

<b>Nombre De Caso de Uso</b>	CU-03 Iniciar Partida nueva
<b>Descripción</b>	El usuario es capaz de iniciar una nueva partida.
<b>Precondición</b>	El usuario no debe tener una partida anteriormente guardada.
<b>Secuencia Principal</b>	1.- El Jugador debe seleccionar nueva partida 2.- El juego debe cargar el primer escenario (el único a escoger) 3.- El Juego Muestra el Escenario Principal y el jugador debe ser capaz de mover el personaje.
<b>Errores/Alternativas</b>	Errores: el Jugador debe asegurarse de no iniciar una nueva partida si se ha guardado una partida anterior. En ese caso: Alternativas: Se despliega un menú dándole a escoger al Jugador si desea continuar en la partida anterior o iniciar una nueva borrando la existente.
<b>Post Condición</b>	Partida Iniciada
<b>Notas</b>	NO

Tabla 11 - Caso de uso CU-03 Iniciar Partida nueva.

<b>Nombre De Caso de Uso</b>	CU-04 Seleccionar Escenario
<b>Descripción</b>	El Jugador debe seleccionar un escenario de un mapa que muestra los niveles disponibles a jugar
<b>Precondición</b>	El jugador debe estar en el menú de selección de escenarios
<b>Secuencia Principal</b>	1.- El jugador debe seleccionar un escenario desbloqueado. 2.- El juego debe cargar el escenario seleccionado, mientras despliega un mensaje de carga. 3.- El juego debe mostrar el escenario cargado y jugable.
<b>Errores/Alternativas</b>	-
<b>Post Condición</b>	Escenario seleccionado listo para jugar.
<b>Notas</b>	NO

Tabla 12 - Caso de uso CU-04 Seleccionar Escenario

<b>Nombre De Caso de Uso</b>	CU-05 Cargar Partida guardada
<b>Descripción</b>	El usuario es capaz de continuar una partida guardada anteriormente.
<b>Precondición</b>	Tener una partida guardada en memoria
<b>Secuencia Principal</b>	1.- El Jugador debe seleccionar cargar partida guardada 2.- El juego despliega un menú de los posibles estados de guardado. 3.- El Jugador selecciona un estado de guardado. 4.- El juego carga el escenario y la instancia de juego. 5.- El juego muestra el escenario tal como había quedado antes de guardar la partida
<b>Errores/Alternativas</b>	Errores: el Juego no es capaz de cargar la partida. Alternativa: Envía un mensaje de disculpa y vuelve al menú principal.
<b>Post Condición</b>	-
<b>Notas</b>	NO

Tabla 13 - Caso de uso CU-05 Cargar partida guardada

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

<b>Nombre De Caso de Uso</b>	CU-06 Abandonar Nivel
<b>Descripción</b>	El Jugador abandona el nivel para iniciar nuevamente el mismo escenario u otro distinto al actual.
<b>Precondición</b>	El personaje debe encontrarse dentro de una partida y dentro de un nivel.
<b>Secuencia Principal</b>	1.- El Jugador debe pausar el juego 2.- El Jugador debe presionar el botón de abandonar nivel 3.- El Juego muestra un mensaje de confirmación de abandono del nivel. 4.-El Jugador confirma el abandono de nivel. 5.- El nivel se cierra. 6.- El juego regresa a la pantalla de selección de niveles.
<b>Errores/Alternativas</b>	Alternativa: 4.1: El jugador cancela el abandono de nivel. 5.1: El juego vuelve a mostrar el menú de pausa.
<b>Post Condición</b>	-
<b>Notas</b>	NO

Tabla 14 - Caso de uso CU-06 Abandonar nivel

<b>Nombre De Caso de Uso</b>	CU-07 Pausar Nivel
<b>Descripción</b>	El Jugador puede ser capaz de pausar el nivel actual que esté en desarrollo.
<b>Precondición</b>	El Jugador debe estar jugando un nivel.
<b>Secuencia Principal</b>	1.- El Jugador debe presionar el botón de pausa en la pantalla. 2.- El juego debe pausar el escenario, ya sean enemigos e instancias, además de bloquear los botones de ataque y salto. 3.- El juego debe desplegar el menú de pausa.
<b>Errores/Alternativas</b>	-
<b>Post Condición</b>	Partida Pausada.
<b>Notas</b>	NO

Tabla 15 - Caso de uso CU-07 Pausar nivel

<b>Nombre De Caso de Uso</b>	CU-08 Reiniciar Nivel
<b>Descripción</b>	El Jugador es capaz de reiniciar el nivel actual en juego.
<b>Precondición</b>	El Jugador debe estar jugando un nivel.
<b>Secuencia Principal</b>	1.- El Jugador debe presionar el botón de pausa en la pantalla. 2.- El juego debe cargar el menú de pausa. 3.- Jugador debe identificar y presionar el botón Reiniciar nivel del menú de pausa. 4.-El juego debe reiniciar el nivel actual, reiniciando todo tipo de contador de energía, puntaje, etc.
<b>Errores/Alternativas</b>	-
<b>Post Condición</b>	Partida reiniciada.
<b>Notas</b>	NO

Tabla 16 - Caso de uso CU-08 Reiniciar nivel

### Tercer Incremento:

<b>Nombre De Caso de Uso</b>	CU-09 Visualizar Mejores Puntajes
<b>Descripción</b>	El Jugador podrá visualizar los mejores puntajes obtenidos
<b>Precondición</b>	El Jugador debe estar en el menú principal
<b>Secuencia Principal</b>	1.- El Jugador debe presionar el botón de mejores puntajes en la pantalla principal 2.- El juego debe cargar la pantalla antes mencionada
<b>Errores/Alternativas</b>	-
<b>Post Condición</b>	Regresar al menú principal.
<b>Notas</b>	NO

Tabla 17 - Caso de uso CU-09 Visualizar Mejores puntajes

### 4.3 Patrones de Diseño

Cuando hablamos de Ingeniería de software, no podemos dejar de hablar de paradigmas de programación o de patrones de diseño, es por eso que al momento de desarrollar un videojuego, tampoco es indiferente utilizar patrones que faciliten el orden en la programación y que garanticen el acceso a la información de forma ordenada.

En el desarrollo de este Videojuego, se han utilizado cuatro patrones de diseño, los cuales garantizan la funcionalidad del juego.

Se procederá a explicar cada patrón de diseño y como se aplica en el videojuego a crear:

- **Singleton:** es un patrón utilizado normalmente para creación, se caracteriza por restringir la creación de un objeto perteneciente a una clase para así evitar la multi-instancia.

Es utilizado en GeekNator para crear la instancia que controlará los botones de movimiento y acción del personaje principal, ya que requiere ser accesada desde cualquier script.

El patrón a la vez es el que controla el tiempo transcurrido desde que se creó la escena de juego y a la vez cuenta el puntaje obtenido por el jugador, que finalmente es enviado al servidor, para ser actualizado en la lista de mejores puntajes, todo esto desarrollado en un objeto que tiene la capacidad de no destruirse, llamado GameMaster.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

- **Composite:** este patrón se caracteriza por ser estructural y sirve para crear objetos complejos a través de otros objetos más simples y similares entre sí.

Es utilizado en GeekNator para interactuar y controlar las capas gráficas y la interfaz de usuario, así como también requiere de controlar los eventos capturados en la interfaz de juego.

Dentro de cada Escena de juego, el componente que controla el patrón es el **GameMaster** (el cual es un objeto vacío, que gestiona todos los objetos que se pueden instanciar dentro de la escena y que controla gran parte de otros scripts), que interactúa directamente con la barra de vida, obligando a este a responder ante cualquier evento (regeneración de vida o muerte).

- **Strategy:** es un patrón que permite la elección de distintos algoritmos a través de decisiones por parte del cliente (o a través de estímulos) que pueden ir intercambiando según las necesidades.

En GeekNator se utiliza bastante para la aplicación de la IA, tanto como en un agente de reflejo simple, como así también en un árbol de comportamiento.

- **Object Pool:** este patrón de diseño es utilizado para desarrollo de software de alto requerimiento de objetos, como su nombre lo indica, es una piscina de objetos que al iniciar el sistema, crea todos los objetos a usarse, para que posteriormente solo sean llamados cuando se necesitan y devueltos cuando se han dejado de usar.

En GeekNator este patrón es el más utilizado, referenciado en la técnica de tiling, la cual consiste en almacenar todos los prefabs y luego bajo demanda solamente hacer una instancia de éstos y ubicarlos en la escena.

Object Pool también es usado en el ataque del personaje principal, el cual reutiliza el mismo prefab de forma iterativa.

A continuación, en la figura 24 se ilustrará parte de código referente a GameMaster, el cual contiene aplicación de los patrones object pool y singleton.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

```
void Start () {
    GameManager.instancia = this;
    this.puntuacion = 0;
    fondoPausa.enabled = false;
    BotonesPause.enabled = false;
    VerificadorReinicio.enabled = false;
    VerificadorSalir.enabled = false;

    this.startPosition = GameObject.Find("personajePrincipal new").transform.position;
    UpdatePuntText ();
    Time.timeScale = 1;

    this.clockRunning = true;
    this.time = 0;

    StartCoroutine (Clock());
}
```

Figura 24, Script de Inicialización de patrón singleton, rutina de reloj y actualización de puntaje.

# Capítulo V

## DESARROLLO DEL VIDEOJUEGO

---

En este capítulo se abordará el tema crucial de este proyecto, el cual corresponde a la implementación del videojuego, denominado “**GeekNator**”, el cual se enmarca en una historia de fantasía, en donde el personaje principal deberá cumplir con la misión de derrotar a una horda de Alienígenas.

Por ende, el objetivo de este capítulo es mostrar las técnicas y procedimientos que se utilizaron en el desarrollo del proyecto, siguiendo la metodología iterativa incremental. Dentro de las siguientes líneas se describirá con mayor detalle los incrementos a realizar y que abarcará cada una de ellas:

**Primer Incremento:** Incremento encargado de la funcionalidad del personaje principal y el uso de los componentes visuales que se desplegarán frente a la cámara. Consta de 3 aspectos.

- La implementación del control del personaje principal.
- La implementación de la cámara.
- El diseño del mundo 2D.

**Segundo Incremento:** Incremento encargado de la jugabilidad y funcionalidad del juego.

- El diseño de la interfaz de usuario.
- La incorporación de enemigos.
- Implementación de Parallax.

**Tercer Incremento:** Incremento encargado de la funcionalidad de Ranking y Logros.

- Conexión con Google Play Services.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

Al terminar los incrementos se realizarán las pruebas de rendimiento, con el fin de determinar las especificaciones mínimas para poder ejecutar el videojuego.

## **5.1 Contextualización del videojuego**

Para contextualizar el desarrollo de este videojuego es necesario conocer el estilo que representará la implementación de éste, es decir qué tipo de videojuego se llegará a desarrollar, cuáles serán las limitaciones que este conlleva y como se juega este tipo de videojuegos normalmente.

Para la aceptación del público ante un videojuego se toman en cuenta 3 factores que son totalmente relevantes a la hora de evaluarlo: jugabilidad, entretención e historia; es por eso que por decisión del desarrollador se realiza un videojuego de estilo plataformas, el cual tiene como objetivo llegar a una meta en determinado tiempo, adquiriendo puntaje, bonificaciones y evaluación al terminal cada escenario; que posteriormente se reflejará en un ranking, comparando las mejores puntuaciones a nivel global.

En las siguientes páginas se dará a conocer como fue desarrollado cada aspecto relevante del videojuego, complementando lo anteriormente descrito.

### **5.1.1 Motivación**

El grupo de desarrollo de videojuegos "Botacura Games" de la universidad del Bio-Bio, fundado el año 2012, ha participado constantemente en actividades de difusión.

Un problema frecuente al momento de realizar las presentaciones es que el público solicita un videojuego orientado a dispositivos móviles, con el cual se pueda apreciar el trabajo realizado por el grupo.

Es por eso que surge la motivación de desarrollar un videojuego para dispositivos móviles, el cual pueda servir para difundir la carrera de Ingeniería Civil en Informática de nuestra Universidad a los estudiantes de enseñanza media.

Otro motivo relevante para desarrollar un videojuego es dar a conocer el amplio rubro que puede lograr un ingeniero civil en informática, que aplicando lo aprendido en la universidad puede generar tecnologías de distinto ámbito y ser un profesional integral.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

El género del videojuego será de tipo plataformas 2D el cual relatará una historia ficticia relacionada con el campus Fernando May, sus docentes, estudiantes y edificios.

### **5.1.2 Trama Del Videojuego**

La trama que el videojuego relata, es una historia ficticia, que describe como un estudiante de la universidad logra rescatar a los profesores que han sido capturados por extraterrestres, que se alimentan de la inteligencia de estos, para hacerse más poderosos.

Ante la valentía del personaje principal, los extraterrestres deciden utilizar clones perversos de los profesores raptados, para evitar que éste los rescate y diluir sus planes.

Para lograr su objetivo, el personaje recorrerá la universidad y sus distintos edificios, enfrentándose a múltiples y variados enemigos para buscar a sus profesores y así poder rescatarlos.

El personaje principal dispondrá de movimientos por defecto, que serán determinados previamente por el desarrollador, estableciendo un salto, un ataque y un movimiento horizontal a través de los escenarios.

### **5.1.3 Caracterización de Personajes**






Para lograr que el videojuego tenga una similitud entre la realidad y la ficción, es que se aplica una leve similitud entre los personajes que interactuarán en el videojuego y su doble de la vida real; para esto es que se tomaron a los profesores más conocidos de la carrera y se incorporaron con aptitudes y movimientos característicos de cada uno, también modificando levemente su aspecto y nombres, para que sean identificados solamente con su alias (que será una asignatura que los docentes impartan con mayor regularidad) y sus ataques.

No solamente los docentes son los caracterizados en el videojuego, se representa al personal de la universidad, ya sea guardias o cocineras, que son parte primordial del progreso del juego en sí.

En la tabla 18 se describe brevemente cada personaje, incorporando su gráfica, alias y característica propia.



Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

Gráfica	Alias	Característica
	Don Pollo	Es el personaje principal del juego, es capaz de lanzar discos DVD a través de una pistola lanza discos y saltar más de lo normal para un ser humano.
	Cocinera Pirata	Enemigo del escenario cocina, es capaz de lanzar cucharones y golpear con el cuerpo y garfio; camina bastante lento y no puede correr.
	Guardia Mutante	Es uno de los enemigos del escenario entrada a la universidad y Mart Collin, es capaz de correr y perseguir al personaje principal, ataca con patadas y puños cuando el enemigo se encuentra cerca.
	La Red	Es una de los enemigos jefes de nivel 3, es capaz de moverse a súper velocidad y atacar lanzando piezas de código, certámenes reprobados y conexiones de red.
	El Fundamento	Es uno de los enemigos jefes del nivel 1, es capaz de saltar bastante, lanzar pantallazos azules y certámenes reprobados, lanza código en SableCC y bloquea los disparos.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

	<p>El Educando</p>	<p>Es uno de los enemigos jefes del nivel 2, lanza libros de inteligencia artificial, usa sus poderes para bloquear ataques y lanza certámenes con parciales rojos.</p>
	<p>Robótica men</p>	<p>Es un profesor de apoyo, que orienta al alumno por donde debe ir en los escenarios y como debe vencer a los enemigos.</p>
	<p>Estatua Voladora</p>	<p>Representación gráfica de la estatua perdida de Marta Colvin que adquiere alas y recorre regularmente el campus. Al entrar en contacto con el personaje principal reduce inmediatamente la salud de éste.</p>
	<p>Bandejas Sucias</p>	<p>Bandejero que ocasionalmente patrulla por el casino de la universidad, chocando a la gente que se cruza en su camino, al entrar en contacto con el personaje principal reduce la salud de éste.</p>

Tabla 18 - Característica de los personajes en GeekNator.

### 5.1.4 StoryBoard

Comúnmente en el desarrollo de un videojuego o corto animado se utilizan las storyboards para orientar a los desarrolladores y guionistas a entender cuál será la transición entre escenarios, cómo se desarrollará la historia durante el uso del juego y cómo se relatará todo de forma lineal y adecuada.

Un StoryBoard es un conjunto de imágenes o ilustraciones que al ser mostradas en secuencia logran relatar una historia, que luego guía la estructura del relato, película o videojuego.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

Este proceso fue desarrollado en los estudios Disney y fue tal su éxito que posteriormente muchos estudios de producción de películas comenzaron a utilizarlo para ordenar las grabaciones y la transición de la película.

Actualmente, en la creación de un proyecto audiovisual con un cierto grado de fidelidad al guion o historia antes creada, un StoryBoard proporciona el orden y secuencia de acontecimientos en su forma lineal, sirviendo de guía para todo el equipo del proyecto.

En el desarrollo de este videojuego se utiliza la misma estrategia, para lograr unir todo de forma idónea y porque es una buena práctica que debe seguirse para lograr un éxito más cuantificable.

En el anexo 1 se visualiza el StoryBoard del videojuego.

## **5.2 Elementos Gráficos**

Otra de las características a mencionar al momento de desarrollar un videojuego, son los elementos gráficos, encargados de darle vida a la programación del videojuego y lograr ser del gusto del usuario, se caracterizan por estar en segundo plano en la aplicación con un objetivo meramente estético (aunque en ocasiones interactúan con el personaje principal del juego) y que aportan con dinamismo a la hora de jugar.

GeekNator posee muchos objetos gráficos, tanto de uso estático como de uso dinámico, haciendo del videojuego un componente visual estéticamente agradable y logrando que el escenario sea casi un protagonista más de la aventura.

Para lograr dinamismo entre el personaje y el escenario, algunos objetos que se encuentran en el juego se han desarrollado con el objetivo de transformarse en enemigos de escenario, dificultando el avance del protagonista; el detalle del funcionamiento de los enemigos de escenario se verá con más detalle en el capítulo 5.4.2.1.

### 5.2.1 Paleta de colores

La paleta de Colores es una herramienta que lista los colores a usar, controlándose por una regla de uso, para crear combinaciones de colores y evitar pixelado, degradado inesperado o solamente un mal contraste de colores. (Agregando también que los usuarios pueden ser daltónicos y la mala mezcla de colores llegue a confundir al mismo)

Encontrar una paleta de colores idónea para la implantación del videojuego es en ocasiones el trabajo más tedioso, ya que se requiere conocer el tipo de degradado que necesita un estilo gráfico o cómo lograr un rango de color que no sea desagradable al usuario y que termine estropeando el juego, aunque este tenga una jugabilidad muy entretenida.

Es para esto que el diseñador gráfico encargado del estilo gráfico, colores y estética del juego crea una paleta de colores.

En la figura 25 se ilustra la paleta de colores usada para recrear el nivel 1 de GeekNator, el cual estará más detallado en el Anexo N°2.



Figura 25, Paleta de colores nivel 1 de GeekNator.

### 5.2.2 Tipografía

Es muy importante elegir sabiamente el tipo de tipografía a usar en la visualización, ya que una mala elección de este confundirá al usuario, haciéndole creer que el videojuego no está del todo acabado o en una fase de testing.

Para ser más concretos, en GeekNator se utilizaron 3 tipos de tipografía, una orientada al menú de opciones, otra para el menú de pausa, y una para la interfaz gráfica del videojuego (puntaje en pantalla y botones de interacción) logrando una visual más clara y fácil de entender.

En la tabla N°19 se ilustrarán los tipos de letra y su uso en el videojuego.

Tipo de Letra	Uso en el videojuego
<b>ChubbyCheeks</b>	Tipo de letra utilizado en los menús de transición y visualización de puntajes acumulados.
Chery Liney	Tipo de letra usado en el menú de pause del videojuego, además de incorporarlo en algunos párrafos de algunos personajes.
Gwibble	Tipo de letra utilizado en interfaz principal del videojuego (pantalla de juego).

Tabla 19 - Tipo de letras utilizadas en GeekNator.

### 5.2.3 Menús

Otra de las características importantes en un juego es la incorporación de menús que logren una transición y un timing (tiempo de reposo) durante el cambio de escenas, como también un cierto control de parte del usuario en configuraciones del juego, ya sea un control sobre la música, configuración de botones, control de la velocidad del personaje, etc, es por esto que se incorporó un menú de configuraciones en GeekNator, para mantener un orden y flexibilidad en el uso del videojuego de parte del usuario, esto quiere decir que jugar no sea limitante entre otras interacciones que

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

pueda tener el usuario con su teléfono móvil, como por ejemplo desactivar el sonido para poder escuchar música y jugar a la vez.

En las Figuras 26 y 27 se ilustra el menú de opciones de GeekNator.



Figura 26, Menu Principal de GeekNator.

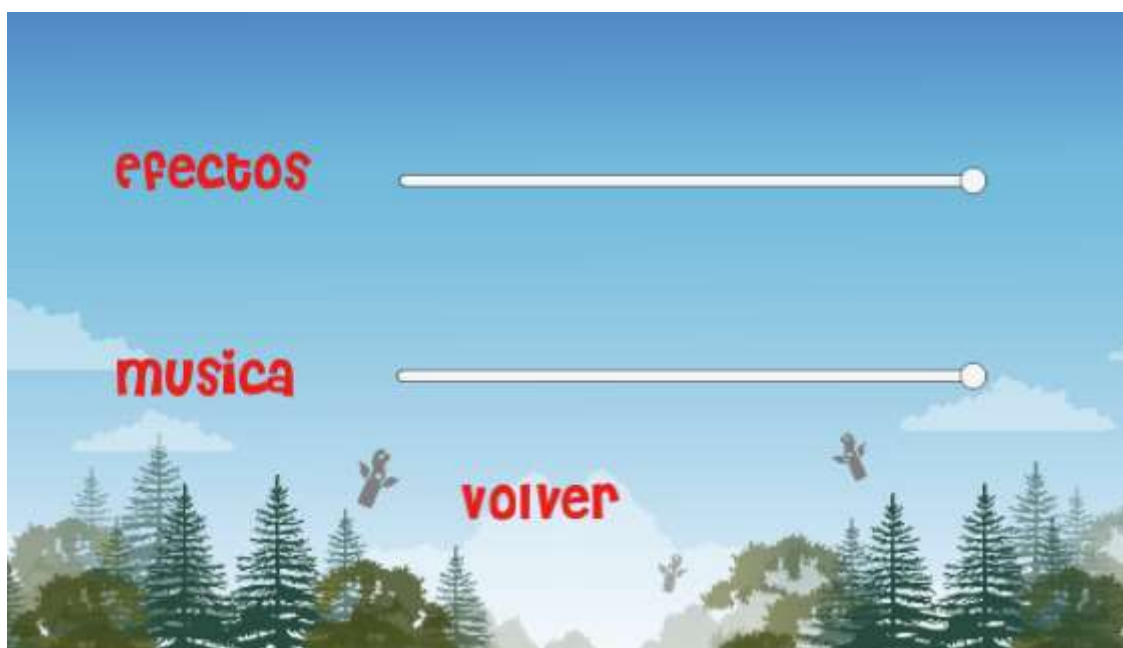


Figura 27, menú de configuración de sonido en GeekNator.

## 5.3 Construcción del Escenario

Una de las dificultades al inicio del desarrollo del videojuego, fue lograr la creación de los niveles de una forma fácil, flexible y de bajo costo en memoria, lo que conlleva un gran problema, porque se requería de un creador de escenario de bajo costo y aunque existen en el mercado assets que hacen un trabajo similar, no se deseaba invertir en algo que no se adaptaba exactamente a lo necesitado, por lo que por convicción se decidió crear un script que permitiera crear los escenarios.

### 5.3.1 Creación del Escenario

Para evitar la carga del escenario completo de una sola vez en la memoria principal de un smartphone se utilizó una técnica antigua en el desarrollo de videojuegos llamada **tiling**, la cual consiste en redibujar el escenario a través de instancias de pequeños fragmentos (haciendo alusión al patrón de diseño Composite).

Entonces, para la creación de escenarios en este proyecto se ha utilizado la estrategia antes mencionada en los videojuegos antiguos, que aseguraba una calidad en el escenario y un bajo consumo de memoria principal, donde el escenario se va generando a través de un archivo de texto que contiene el nivel dibujado con caracteres, que luego un Script intérprete es capaz de traducir y generar en la escena de juego, así la creación de niveles es más rápida, ya que requiere solamente generar el archivo de texto ASCII que posteriormente será traducido.

Así mismo, para convertir estos caracteres ASCII en elementos gráficos de Unity se requieren muchos Prefab (objetos prefabricados que se pueden instanciar sin riesgo de modificarse) que contienen una imagen de renderizado del mismo tamaño; así el generador lo que hace es dibujar el Prefab, y luego posicionar un punto donde irá generado el siguiente Prefab, haciendo esto de forma iterativa hasta leer todo el escenario.

En las figuras 28 y 29 se muestra la comparativa de la lectura de archivo del escenario del archivo de texto generador de nivel, para demostrar la eficiencia que se buscaba.





### 5.3.2 Niveles del Juego

El script para generar el escenario, explicado en la sección 4.3.1, permitirá definir los niveles con mayor comodidad; es por esto que ahora se define la cantidad de niveles a crear y que representa cada uno.

GeekNator consta de 3 niveles en los cuales se recorre la universidad, desde la entrada hasta el edificio de laboratorios centrales de computación. El escenario 1 comienza en el paradero cercano a la universidad, para posteriormente pasar por el camino frontal a la universidad hasta llegar al campus Marta Colvin; terminando el escenario al entrar al museo del Mismo Nombre.

El escenario 2 comienza en la entrada del casino del campus Fernando May, lo recorre completamente para terminar con una batalla en el mismo lugar.

El escenario 3 comienza en la entrada del edificio de laboratorios centrales de computación, recorre el exterior de los laboratorios y aulas, para posteriormente terminar con dos batallas en el final del escenario.

Ya explicado esto, a continuación se necesita crear los prefabs que posteriormente darán vida al escenario; para que esto suceda se debe tener conocimiento de 3 aspectos:

- **Física de los Personajes:** El escenario debe ser capaz de gestionar las físicas instanciadas a los personajes que interactúen dentro del escenario, por lo que la creación de prefabs con colisionadores es primordial al momento de instanciarlos posteriormente.
- **Capas de Renderizado:** El renderizado de los sprites renderer debe ser gestionado por capas, para lograr un mejor acabado y la estética deseada, es por eso que se utilizan las 10 primeras capas de renderizado para darle vida al nivel, ya que desde la capa 11 en adelante se utilizaran para la interacción de los personajes (ya sean enemigos o no).

En el caso de los escenarios 2 y 3, que contienen objetos sobre el área de juego, las capas usables deben ser superior a 50, para lograr el detalle estético esperado.

- **Dimensiones de los elementos gráficos:** Todos los prefabs deben tener al menos la misma dimensión horizontal, para lograr un acabado sin bugs o fallas estéticas, que terminen perjudicando las físicas o el renderizado de éste.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

Al lograr estos 3 aspectos a cabalidad la creación de los escenarios se normalizará y se visualizará de forma homogénea.

La Figura 30 ilustra un trozo del escenario N°2.



Figura 30, trozo de escenario 2.

Finalmente, en la creación de escenarios, para que el suelo pueda interactuar con el personaje principal y con los enemigos, es necesario agregarle un componente de tipo box collider (colisionador en forma de cuadrado) a todos los tiles que representan el suelo y aplicarle el parámetro de capa "Piso", para que el personaje reconozca a este como tal y finalmente orientarse al momento de saltar.

Las figuras 31 y 32 ilustrarán los componentes agregados al tile piso, y como se verá este con las aplicaciones.

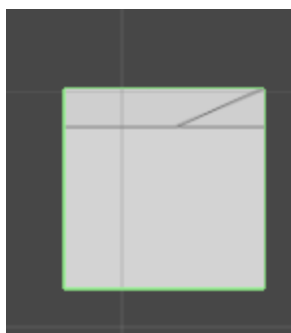


Figura 31, resultado de BoxCollider aplicado en tile piso.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

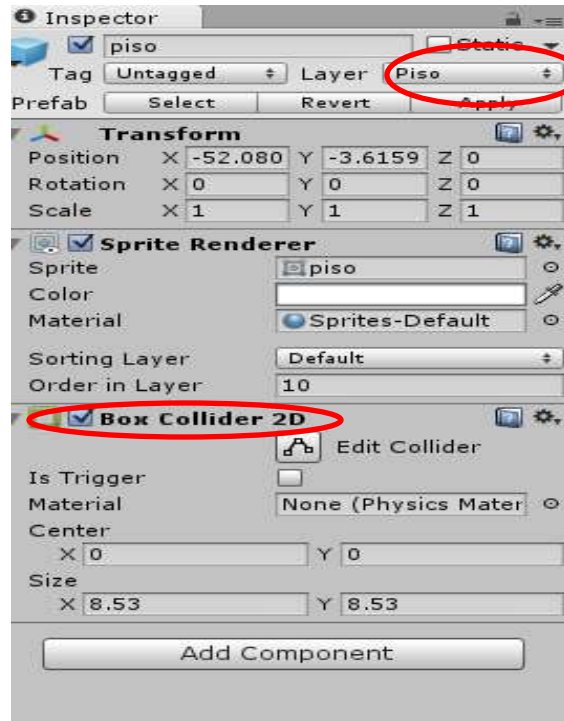


Figura 32, componentes agregados al tile de piso

### **5.3.3 Parallax**

Una de las técnicas más usadas últimamente en los videojuegos y páginas web responsives es el uso de parallax o parallaxing.

El Parallax es una técnica de desarrollo de 2D que consigue darle un efecto de profundidad y proyección al escenario del videojuego a través del movimiento de los componentes más lejanos y más cercanos a la cámara, otorgando realismo y un acabado consistente, que alcanza una visualización más agradable del escenario.

La premisa del parallax es el movimiento desigual de los componentes, debido a una visión subjetiva del escenario, esto quiere decir que mientras más lejos esté el objeto, su movimiento relativo será menor, a diferencia de los objetos más cercanos a la cámara o al observador, que se moverán con mayor velocidad ante el perceptor, desapareciendo del punto de visión mucho más rápido que los objetos lejanos.

En GeekNator, el escenario 1 y 3 poseen parallax para distintos objetivos, pero usando el mismo scripting, dando a conocer que el parallax tiene distintos usos, aportándole versatilidad a este mismo.

En el Anexo 3 se ilustra el script de Parallax utilizado en el videojuego.

### **5.3.4 Gestión de la cámara**

En Unity y en muchos otros motores gráficos gratuitos, el uso de cámaras puede ser casi ilimitado (teniendo los componentes necesarios, bastante memoria ram y video) pero esto no hace que un videojuego con muchas cámaras sea de mejor calidad o de mayor entretención, ya que el ahorro de memoria en un videojuego móvil es la prioridad ante todo. Teniendo en cuenta esto, es necesario gestionar la cantidad de cámaras y el uso que se le dará a cada una, ya sea para un uso similar o no, es totalmente ineficiente usar dos cámaras con el mismo objetivo.

El uso de múltiple cámaras está justificado solamente si cada una tiene un uso único e irreplicable, cabe destacar que éste es un componente que usa muchos recursos, y abusar de su uso terminará consumiendo muchos recursos.

Es por eso que en GeekNator el uso de las cámaras también será justificado, para ahorrar recursos y lograr una fluidez más grata a la vista.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

En el desarrollo del videojuego, se establecieron 3 tipos de cámaras distintas, las cuales serán descritas en la tabla N°20:

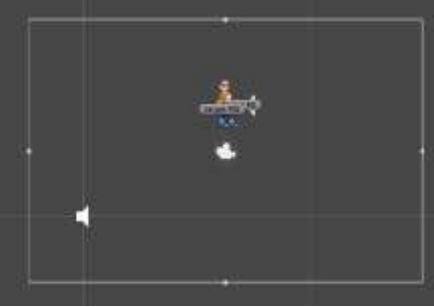

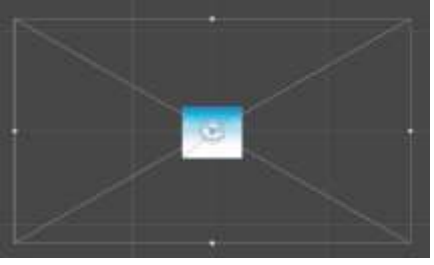
Nombre de Cámara	Ilustración	Uso de Cámara
Cámara Principal		<p>Cámara usada para visualizar el escenario, la cual contiene scripts de seguimiento del personaje principal, además de contener el objeto AudioListener, encargado de dirigir todo el sonido del escenario.</p>
Camara Game Over		<p>Cámara utilizada para visualizar el término de cada nivel y la muerte del personaje, usando los mismos componentes pero distintas reacciones ante la profundidad del videojuego.</p>
Camara Parallax		<p>Cámara utilizada para controlar el parallax de los distintos escenarios, con un uso distinto al de las otras dos cámaras, ya que está configurada en perspectiva y no en ortográfica (conceptos mencionados en el capítulo 3.1.3)</p>

Tabla 20 - tipos de cámaras usados en GeekNator.

## 5.4 Control de la interacción [Lenguaje de Scripting]

Ciertamente, el componente con mayor utilidad y con mayor cantidad de horas de trabajo, es el control de las interacciones en el videojuego, lograr el movimiento de los personajes, la secuencia de eventos y funcionamiento lógico del juego en sí; es por esto que el uso del lenguaje de programación es un pilar fundamental en la utilización y creación del videojuego, teniendo como objetivo disponer de los componentes del

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

motor gráfico a través de los scripts, dando como resultado una reacción visible (en la mayoría de las ocasiones) entre la gráfica y la lógica.

Unity posee 3 lenguajes de programación utilizables a la hora de desarrollar, C# que está basado en la plataforma .NET, JavaScript que viene por defecto en el motor y finalmente Boo, que es una aplicación y breve modificación de Python.

Para el desarrollo de este proyecto se utiliza el lenguaje de Scripting C#, que contiene una mayor cantidad de documentación y aplicación entre los desarrolladores en Unity, por lo que hay mucha más documentación.

#### **5.4.1 Control de personaje principal**

En los videojuegos de plataformas en la mayoría de las ocasiones, el uso de un personaje principal es casi evidente, es por eso que al momento de desarrollar el control de personajes, el personaje principal es el primero a desarrollarse, ya que controlando el movimiento de este, es posible testear el acabado del nivel y su lógica propuesta.

Para lograr este objetivo, se crearon scripts para distintas funcionalidades del personaje, como lo era la interacción, movimiento y salud; para posteriormente comunicarse con la gráfica y retroalimentar al usuario de sus acciones.

Con la interacción del personaje, se hace referencia al trabajo que este requiere para colisionar con objetos del escenario, activar y desactivar funciones y atacar a los enemigos, por lo que principalmente se crean scripts de arma y municiones.

Con el movimiento del personaje se referencia al desplazamiento vertical y horizontal de este, estableciendo una velocidad de movimiento, un tiempo de salto y una fuerza con la que el personaje se desvincula del suelo; Para informar al Script cuando el personaje salta y separa los pies del suelo, se utiliza una técnica llamada Raycasting, encargada de crear una línea imaginaria entre el centro del personaje y un punto un poco inferior a los pies, cuando esta línea atraviesa un objeto tipo suelo, informa al script para que este lance la animación correspondiente y viceversa.

Con la salud del personaje, se desarrollan más interacciones con el entorno visual que con los otros dos aspectos, ya que es totalmente necesario informar al usuario de cuando el jugador está perdiendo salud o cuando está a punto de morir.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

Es por esto que el script de salud interactúa directamente con el Componente Canvas (encargado de la interfaz de usuario) y la cámara que informa de los componentes visuales (ya sea barra de salud como el contorno de muerte) para retroalimentar al usuario de modo que cuide su accionar durante la partida.

En la Figura 33 se ilustra el script de salud del personaje principal.

```

public class SaludPersonaje : MonoBehaviour {
    public float ps = 100;
    public Slider healthbar;
    static float tiempoEspera = 3.0f;
    private float delay=0.25f;
    private float delaySangre=1.0f;
    private bool check=false;
    private bool checkSangre=false;
    public Image sangre;
    Animator anim;
    public GameObject pers;
    private movimientoPlayer movim;
    private EnemigoMov eM;
    public Image fondoMenu;

    void Start () {
        anim = GetComponent<Animator>();
        pers = GameObject.Find ("personajePrincipal new");
    }
    public void reducirVida(float damage){
        this.ps -= damage;
        colorear ();
        if (this.healthbar) {
            this.healthbar.value =this.ps;
        }
        if (this.ps < 20) {
            sangre.enabled=true;
        }
        if(this.ps <= 0){
            StartCoroutine(Wait(3));
            CamaraGameOver.enabled =true;
        }
    }
    IEnumerator Wait (float tiempo) {
        yield return new WaitForSeconds(tiempo);
        Time.timeScale=0f;
        fondoMenu.enabled=true;
        GameObject.Find("personajePrincipal
new").GetComponent<Animator>().animation.Stop ();
    }
}

```

Figura 33, Script de salud del personaje principal.



## 5.4.2 Inteligencia Artificial

A diferencia del personaje principal, los enemigos no son controlados por una persona natural detrás de un control o un dispositivo móvil, sino que se trata de algo autónomo, que debe funcionar como si alguien la controlase, es por esto que al momento de crear un videojuego siempre se debe hablar de enemigos con inteligencia artificial, un concepto que ha ido cambiando con el tiempo puesto que lo que años atrás se consideraba inteligencia artificial hoy en día no se considera como tal, por lo que desarrollar una en estos días no garantizará que en un tiempo determinado siga atribuyéndose como inteligencia Artificial.

### 5.4.2.1 Inteligencia Artificial basada en Agente de Reflejo Simple

Dentro de la inteligencia artificial a implementar durante el proceso de desarrollo, está la de los enemigos de escenario, los cuales tienen un rango de visión del entorno un poco limitada y un comportamiento reactivo (como lo ilustra la Figura 34); podría compararse a un agente de reflejo simple, que junto con un poco de lógica difusa logra crear un personaje similar a un tercero controlando el enemigo. Este proyecto está orientado a lograr hasta 4 personajes que llegan a ser enemigos de escenario, algunos con mayor nivel de dificultad que otros, cambiando su velocidad y reacción en el momento de interactuar con el enemigo.

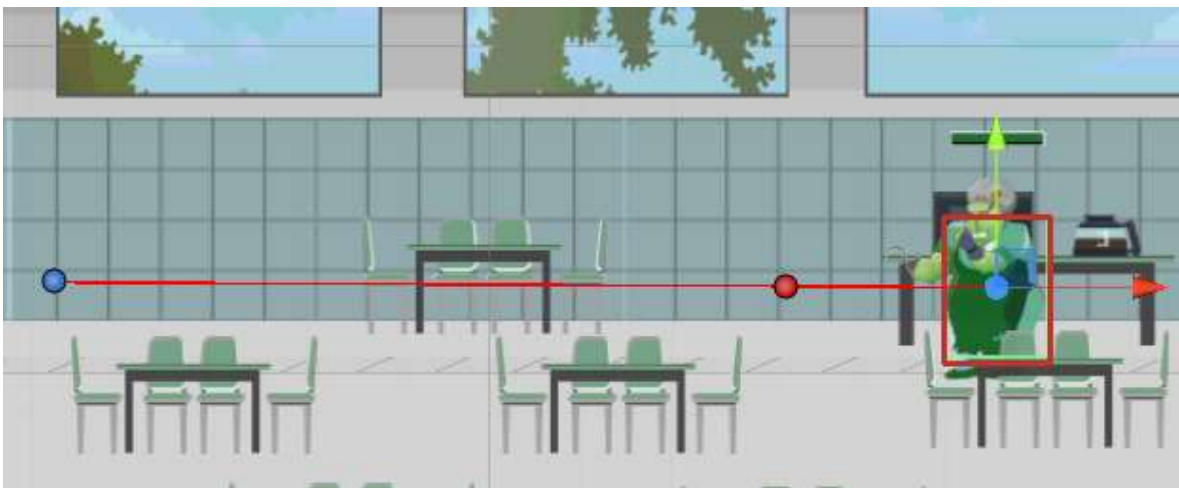


Figura 34, Punto de visión y estimulación de enemigo.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

Para describir satisfactoriamente los agentes de reflejo simple utilizados en el proyecto, se procederá a explicar parte del pseudocódigo del comportamiento de uno de los enemigos del juego.

En la Figura 35 se ilustra el pseudocódigo de la IA de Cocinera Pirata y posteriormente en la Figura 36 se ilustra el autómata de estado finito del mismo personaje.

```

Void update () {
    Parado;
    If (distancia < 100) {
        Acercarse;
        If (lineaEsAtravesada) {
            LanzarCucharon ();
        }
        If (!lineaEsAtravesada &&
            distancia < 15) {
            AtaqueGarfio ();
        }
    }
}
    
```

Figura 35, Pseudocódigo agente de reflejo simple

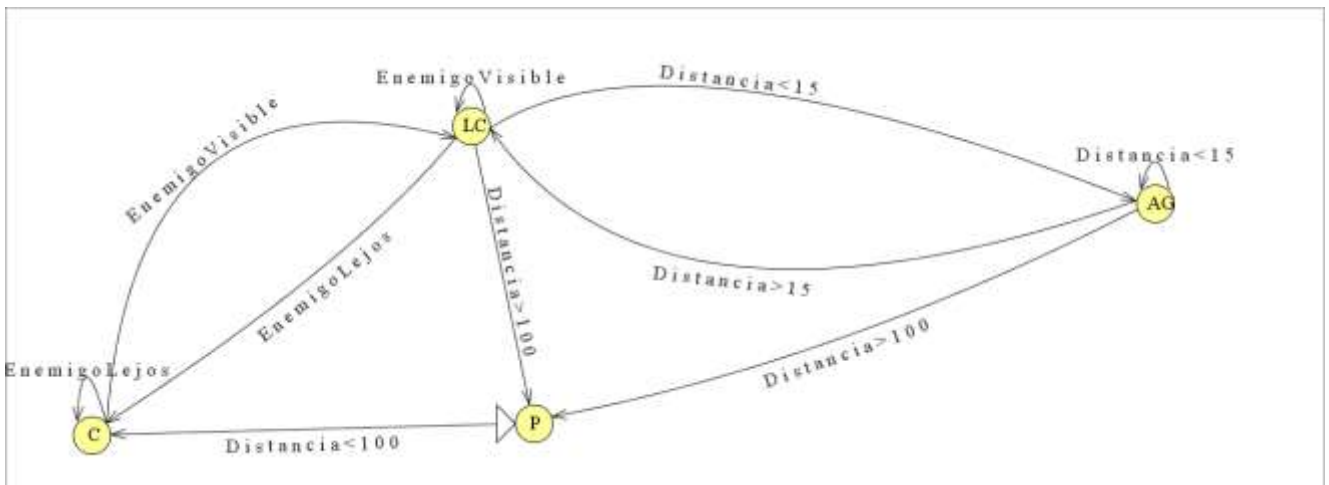


Figura 36, Autómata Agente de reflejo simple

### 5.4.2.2 Inteligencia Artificial Basada en Árbol de Comportamiento

Los jefes de nivel, a diferencia de los enemigos de escenario, son personajes con más capacidades de interacción que los agentes de reflejo simple, esto se debe a que están basados en árboles de comportamiento; un sistema utilizado en juegos a fines de los años 90 y que aún se utiliza en un sinnúmero de juegos.

Un árbol de comportamiento, es una estructura que simplifica la forma de escribir los movimientos de un personaje, unificando el árbol binario y los autómatas de estado finito, dejando como resultado un árbol limitado en estados pero eficiente en movimientos de búsqueda desde y hacia la raíz de este mismo.

Al sistema más básico de árbol de comportamiento se le llama isla o pulpo (en la jerga vulgar del grupo de desarrolladores), ya que el autómata generado se asemeja bastante a estos, instanciando los ataques y movimientos dependiendo de un estado base.

En GeekNator existen 3 enemigos con inteligencia artificial basada en árbol de comportamiento (dos de la cual también poseen Pathfinding) y uno que tiene un sistema más básico, asemejando al pulpo anteriormente mencionado; el cual queda ilustrado en la figura 37.

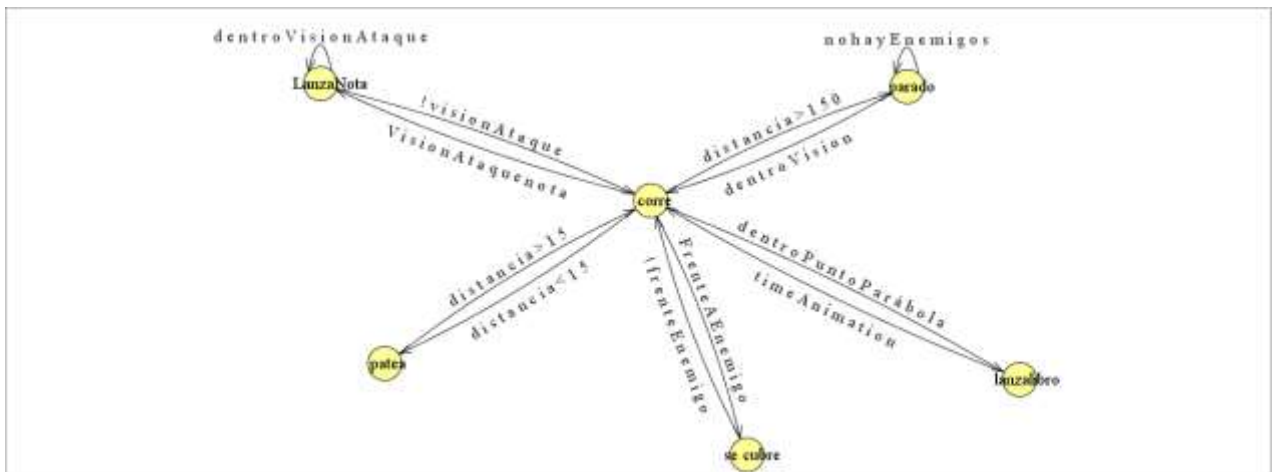


Figura 37, Autómata generado a través de árbol de comportamiento, conocido vulgarmente como pulpo.

El sistema de búsqueda informada que utilizan estos enemigos es algoritmo A\*, el cual requiere de 3 características para su eficiente funcionamiento:

- Un nodo objetivo (al cual debe llegar).

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

- Un nodo o punto inicial
- métodos para determinar los costes directos, indirectos y cuales nodos son traspasables (en este caso el espacio por donde puede transitar).

Los personajes a utilizar este algoritmo, usan las posiciones en los ejes cartesianos como nodos en el cálculo de la heurística, para posteriormente utilizar la distancia de manhattan y determinar cuál es el movimiento más efectivo al atacar.

La fórmula para determinar la distancia de manhattan está descrita en la Figura 38.

$$H = \text{Math.Abs}(\text{nodoActual.X} - \text{nodoFinal.X}) + \text{Math.Abs}(\text{nodoActual.Y} - \text{nodoFinal.Y})$$

Figura 38, Fórmula para determinar la distancia de Manhattan.

Luego de determinar la distancia más corta, se procede a contemplar si ese trayecto es el más eficiente, el cual será descrito en el Script A\* en el anexo N° 3.

Finalmente, cuando el script es ejecutado y el autómata comienza a funcionar, los personajes aplican hilos separados, que logran computar cada uno de los movimientos anteriormente generados a través de animaciones o scripting.

### 5.4.3 Persistencia

Una de las características significativas de los videojuegos actualmente es la capacidad de almacenar el progreso que el jugador obtenga durante su sesión, para posteriormente recuperar su partida y seguir jugando donde lo había dejado anteriormente y así no perder su progreso. Para hacer esto se explicará una técnica anteriormente utilizada por un par de empresas del rubro y muy utilizada en la orientación a objetos; la cual es el uso del patrón de diseño Singleton, el cual se caracteriza por asegurar a lo más una instancia del programa ejecutado, para evitar inconsistencias y problemas más severos.

### 5.4.3.1 Modelo de datos

En esta sección se mencionará cuál es el modelo actual del videojuego y cómo se utilizará este mismo para posteriormente guardar información.

El modelo de datos es importante para ordenar la información a salvar en la instancia al momento de guardar una partida, para que al momento de ser recuperada esta no afecte en nada en el estado o jugabilidad del videojuego. Para lograr esto se modela una base de datos, la cual se ilustra en la Figura 39, para posteriormente enseñar cómo será el proceso de guardado, lo cual se menciona en el siguiente subcapítulo.

Cabe mencionar que en comparación a un proyecto informático Genérico, el modelo de un videojuego suele ser muy simple y pequeño.

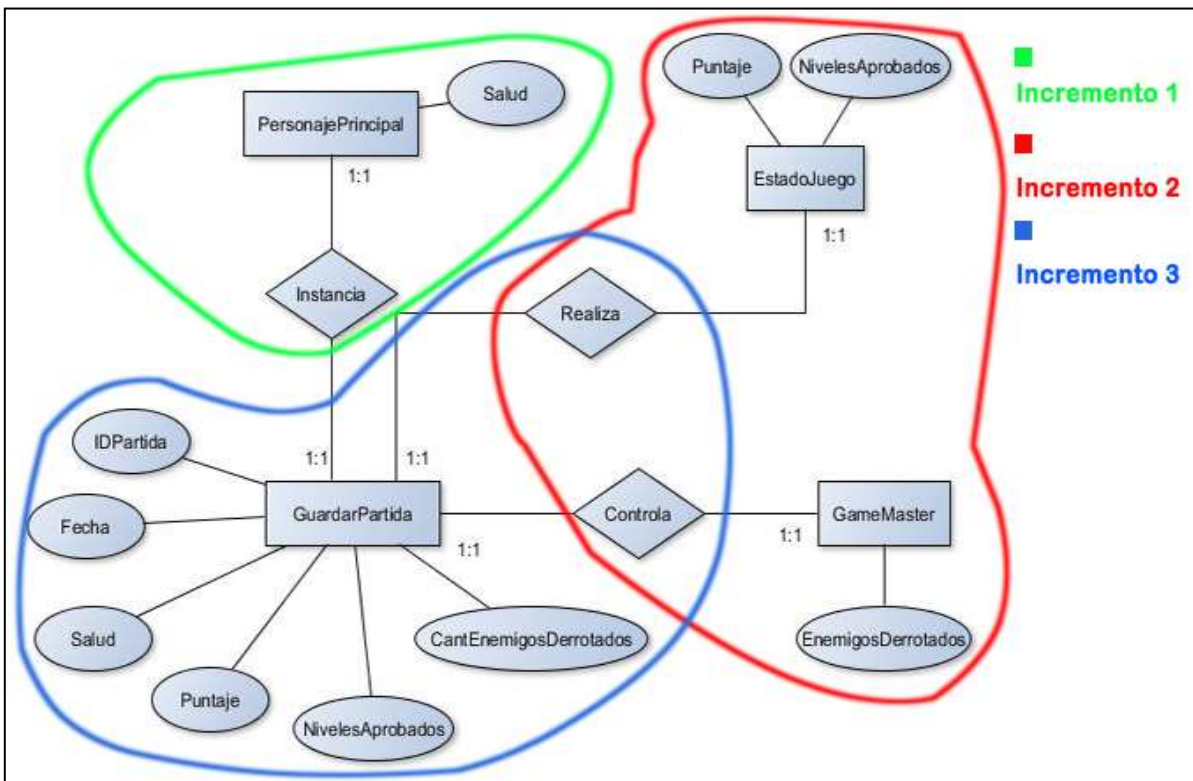


Figura 39, Modelo de Datos GeekNator.

### **5.4.3.2 Guardar y Recuperar Partida**

Para que el videojuego no sea monótono o aburrido, se utiliza la opción de guardar y recuperar partidas, ya que si no existiera esta opción, el juego se tornaría algo repetitivo y lineal (haciendo que el jugador siempre juegue el mismo nivel o le dedique más tiempo del necesario para terminarse este mismo).

Para la aplicación de esta solución ante este problema, se utiliza la clase de unity llamada PlayerPrefs, la cual está orientada a darle configuraciones individuales y extraordinarias a cualquier componente seleccionado mediante scripting (logrando así simular una base de datos convencional).

Para dicha configuración, se establecen parámetros limitantes o lineales, como variables atómicas del lenguaje de Scripting, con el objetivo de simular un hash, que se guardará en memoria y posteriormente se recuperará con una función creada por el desarrollador para hacer tal labor y que será direccionado hacia una instancia del juego.

### **5.4.3.3 Registro del Ranking**

Para hacer aún más entretenido y desafiante el videojuego, se crea también un ranking de los mejores puntajes a lograr, el cual se calcula con un script que proporciona el tiempo que tarda el usuario en terminar cada nivel, añadiendo además el puntaje que ha generado en el transcurso del nivel combatiendo los enemigos, que generan puntaje al morir y objetos que proporcionen puntaje por sí mismos.

Para lograr un ranking eficiente para un dispositivo móvil, se incorpora el uso de las librerías que proporciona google play ante el uso de videojuegos, el cual tiene un sistema unificado que hace gran parte de la labor y ordena de forma eficiente la información, por lo que solo se necesita enviarle la información al gestor, y el ordenará y visualizará de forma automática el puntaje.

Para que el usuario pueda visualizar dicha información deben ocurrir dos cosas en segundo plano: que éste tenga acceso a internet al momento de consultar dicha información y que éste inicie sesión en google play services (el cual se hace de forma automática al intentar visualizar el ranking).

#### **5.4.3.4 Gestión de los niveles**

Finalmente, para lograr que el videojuego tenga un lineamiento eficiente, se utiliza la gestión de los niveles, la cual se asegura que los escenarios han sido recorridos y el puntaje obtenido durante el transcurso pueda ser almacenado de forma correcta, para que al momento de volver a jugar el nivel no se infle el marcador global (prácticamente haciendo trampa en el ranking).

Para hacer esto se utilizó un Objeto que se comparte con todas las escenas, el cual tiene un verificador de nivel en un Script, que comprueba que este ya haya sido jugado anteriormente, para posteriormente comparar los puntajes, si este es menor no hace nada, pero si este es superior se reemplaza para agregar la diferencia con la anterior participación (logrando subir un poco en el ranking).

En caso de que este no se haya jugado anteriormente simplemente el script hace caso omiso de la consulta anteriormente mencionada y termina de ejecutarse de forma normal.

# Capítulo VI

## PRUEBAS

---

En el siguiente capítulo se presentan las pruebas realizadas al videojuego GeekNator, con el fin de conocer su comportamiento en diferentes plataformas y escenarios, esto es, jugando de forma continua y jugando y guardando, para posteriormente volver a jugar. Es por esto que se realizarán 3 tipos de pruebas, las cuales están orientadas a potenciar y aprovechar de forma óptima los recursos disponibles.

Las pruebas realizadas al videojuego son: Pruebas de usabilidad, las cuales determinarán la jugabilidad y nivel de entretenimiento de GeekNator, pruebas de rendimiento, determinando el mínimo de recursos para ejecutar el videojuego y finalmente pruebas unitarias, que determinan que tan eficientemente se ejecuta el código y si su respuesta a estímulos es la esperada.

### 6.1 Pruebas de Usabilidad

La jugabilidad es un factor que se debe cumplir en todo videojuego, pero que se comprueba solamente en la fase de prueba del producto. Para medir la jugabilidad se utilizan técnicas en la fase de prueba siguiendo técnicas heurísticas similares a las propuestas por Nielsen [11] para la usabilidad.

Usamos un videojuego porque nos divierte, porque nos entretiene, porque nos hace sentir felices, porque nos ayuda a pasar el tiempo, etc. Un sistema interactivo, como un videojuego, no nace para que el usuario pueda realizar unas tareas específicas, nace con un objetivo específico muy concreto: hacer sentir bien al jugador mientras lo usa, un objetivo mucho más difuso y subjetivo que el de un producto software de aplicación general.



En [12] se define la Jugabilidad como el conjunto de propiedades que describen la experiencia del jugador ante un sistema de juego específico, cuyo principal objetivo es entretener y divertir de forma satisfactoria y creíble cuando se juega solo o acompañado.

Para obtener una medida de la jugabilidad de GeekNator usaremos una métrica, definida en [13], que permite medir la satisfacción del usuario en juegos educativos. Se adaptará este instrumento para medir las características que se desea.

### **6.1.1 Evaluación del Prototipo**

La primera prueba completa realizada al videojuego, tiene como objetivo evaluar y detectar problemas en:

- El diseño base del juego
- Jugabilidad básica del juego
- El uso de controles del juego
- El uso de menús de configuración del juego

Se contó con una muestra de 11 estudiantes de la carrera Ingeniería Civil en Informática los cuales ejecutarán el videojuego y jugarán hasta terminarlo (mencionados en Anexo 4).

### **6.1.2 Procedimiento**

Cada evaluador cuenta con un dispositivo móvil o Smartphone en el cual se procede a instalar una versión del videojuego, además se llevan copias impresas de la pauta de observación y cuestionario de feedback de usuario (la cual puede visualizarse en el anexo 4).

Individualmente se entregan las indicaciones del proceso de evaluación, para esquematizar las pruebas de usabilidad. Para eso se siguieron los siguientes pasos:

- Se pide a los evaluadores leer las instrucciones entregadas en forma impresa (documento de observación no participante) y lograr los hitos ahí mencionados.
- Posteriormente se le pide iniciar el videojuego y descubrir las funcionalidades del menú principal (configuración de sonido y visualización del ranking).
- Se prosigue con el inicio del nivel 1, en el cual ellos mismos deben descubrir el objetivo del escenario (orientado por el guía en el videojuego) probando así el movimiento del personaje, el ataque, el salto y las opciones de pausa.

Al terminar el escenario final de la versión demo del videojuego, se le pide al evaluador rellenar el cuestionario de feedback del usuario.

### 6.1.1.2 Resultados

De los datos recogidos de la aplicación de la pauta de observación se detectó lo siguiente:

- El sistema de publicación y distribución del videojuego, en este caso google play y la web propia, es muy eficiente e intuitiva.
- El videojuego es inclusivo y refleja el objetivo de asemejarse con la universidad.
- La dificultad es proporcionada para todo tipo de usuarios, ya sea jugadores frecuentes o personas que no dedican su tiempo libre a jugar.
- Los jugadores siempre supieron que hacer en distintas situaciones, ya que el juego es lineal y fácil de jugar.

También se muestra en la tabla 21 la métrica aplicada al término del juego, donde las preguntas contienen una ponderación, en la escala del 1 al 7, siendo 1 la ponderación con menor éxito y 7 la ponderación con mayor éxito; junto a ella se muestra el promedio resultante de las pruebas, que se encuentran detalladas en el Anexo N°4.

Pregunta	X
¿Me gusta el videojuego?	6.3
¿El juego es entretenido?	5.6
¿El juego es desafiante?	4.6
¿Me sentí controlando las situaciones en el videojuego?	6.8
¿Supe que hacer en cada situación del juego?	5.6
¿El juego es fácil de utilizar?	7
¿Las imágenes del juego son claramente identificables?	7
¿El videojuego relata una buena historia?	5.3
¿Los enemigos son difíciles de vencer?	3.8
¿La música del videojuego es acorde a las situaciones?	6.5
¿El menú principal es fácil de utilizar?	5.6
¿El menú de pausa es fácil de utilizar?	7

Tabla 21, Métrica utilizada para evaluar el desempeño del videojuego.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

En cada una de las preguntas de Feedback de usuario final se recopilaron las siguientes ideas:

¿Qué te gustó del juego?

- Fácil de Jugar
- Está centrado en cosas de la universidad.

¿Qué no te gustó del juego?

- Muy corto Hasta ahora.
- Las colisiones de los enemigos no son claras.

¿Qué le agregarías al juego?

- Cambio de armas
- Un jugador femenino.
- Un elemento para recuperar vida.

Observaciones.

- Poder subirse a elementos del juego (como el microbús "Lentos").
- Agregarle más niveles.

## **6.2 Prueba de Rendimiento**

Estas pruebas tienen como objetivo determinar los requisitos mínimos para el uso óptimo del videojuego en distintos dispositivos.

Para determinar el rendimiento en dispositivos móviles, se utilizó una librería incorporada con Unity, que al ejecutarse con el dispositivo a evaluar vía USB, despliega estadísticas de funcionamiento del videojuego.

### **6.2.1 Procedimiento**

Se utilizó una resolución de proporciones 16:9 y se recorrió el juego de principio a fin, es decir, hasta derrotar al último enemigo. Para ello, se utilizaron 3 dispositivos móviles, cada uno con capacidades diferentes. Los cuales estarán mencionados en el Anexo 4.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

Para Evaluarlos entre sí, se utilizó la herramienta de Testing incorporada con Unity, que permite al Desarrollador visualizar ciertas características en modo depuración, para terminar aplicando una nota a la prueba.

### 6.2.2 Resultados

En la Tabla N°22 se visualiza los resultados de la prueba.

Característica	Lg g3 Beat	Motorola D1	Lg L3
<b>Frames por Segundo (FPS)</b>	56 fps	45 fps	25 fps
<b>Resolución proporcional</b>	Si	Si	No
<b>Funcionamiento</b>	7.1 de 10	6.5 de 10	4.3 de 10
<b>VRAM Utilizada</b>	71Mb de 512 MB	71Mb de 256 Mb	71Mb de 128 Mb
<b>RAM utilizada (promedio)</b>	200 Mb de 1Gb	200 MB de 512Mb	200 MB de 256 MB

Tabla 22, Especificaciones de características en Prueba.

Así, al analizar los datos obtenidos luego de realizar las pruebas, se puede concluir que para ejecutar fluidamente el videojuego, se requiere como requisitos mínimos las siguientes especificaciones:

- **CPU:** 1 GHz dual core Qualcomm Snapdragon 400 o similar.
- **RAM:** 512 MB para dispositivos de gama media o baja.
- **Video:** tarjeta aceleradora de gráficos con memoria dedicada de video de 256MB o superior

Notar que las especificaciones, son aproximaciones en base a los resultados obtenidos, que además, se ajustan a los requisitos de la mayoría de los videojuegos que están disponibles en el mercado objetivo (Google Play).

## 6.3 Pruebas unitarias

Las pruebas unitarias se realizan para controlar el funcionamiento de pequeñas porciones de código como ser subprogramas (en la programación estructurada) o métodos (en POO).

Generalmente son realizadas por los mismos programadores puesto que al conocer con mayor detalle el código, se les simplifica la tarea de elaborar conjuntos de datos de prueba para testearlo.

Los métodos de cobertura de caja blanca tratan de recorrer todos los caminos posibles por lo menos una vez, lo que no garantiza que no haya errores pero pretende encontrar la mayor parte.

El tipo de prueba a la cual se someterá a cada uno de los módulos dependerá de su complejidad. Recordemos que nuestro objetivo aquí es encontrar la mayor cantidad de errores posible, Es por eso que en GeekNator se realizaron estas pruebas solo en los Scripts con mayor dificultad y más extensos, logrando así reducir considerablemente la complejidad de estos.

### 6.3.1 Procedimiento

Este tipo de pruebas, a diferencia de las anteriormente explicadas, se realiza durante el proceso de desarrollo del proyecto para corroborar que los scripts tabulados tengan la funcionalidad requerida, reduciendo el costo en memoria principal y eliminando las llamadas de objetos y componentes que no se utilizan, para esto se bosqueja un diagrama de flujos de lo que el script está realizando, para posteriormente reducirlo y fragmentarlo en rutinas más pequeñas y serializables, que finalmente serán escritas nuevamente en el Script.

Un ejemplo de esto, es el ataque del personaje principal, que necesita ser independiente del movimiento, para esto se tomó en cuenta que ambos movimientos son independientes entre ellos, por lo que naturalmente se harían dos scripts independientes, pero eso utilizaría un uso excesivo de memoria en momentos de ataque continuo y alto índice de enemigos (donde el procesador necesitará instanciar más objetos). Es por eso que la unificación de ambos movimientos en una sola clase y

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

que compartan objetos y variables hacen que el script se comporte de forma correcta y necesaria.

El Script del movimiento del personaje principal resultante se visualiza en el anexo 3.

### **6.3.2 Resultados**

Al Finalizar éstas pruebas, se obtienen script mucho más eficientes y que realizan sus tareas de forma metódica y reutilizable, lo cual no tiene resultados visibles en esta prueba, pero que demuestran un resultado en la jugabilidad y las pruebas de Usabilidad.

# Capítulo VII

## PUESTA EN MARCHA

---

Todo proyecto está destinado a finalizarse en un plazo predeterminado, culminando en la puesta en marcha del sistema desarrollado, comprobando que funciona adecuadamente y responde a las especificaciones en su momento aprobadas. Esta fase es también muy importante no sólo por representar la culminación de la operación sino por las dificultades que suele presentar en la práctica, alargándose excesivamente y provocando retrasos y costos imprevistos.

Es por eso que en este capítulo se hablará sobre las plataformas donde se ejecutará el videojuego y cómo será la publicación de este en las distintas plataformas.

## **7.1 Plataformas de Ejecución**

Como anteriormente se ha mencionado, este proyecto consta de dos plataformas donde se ejecutará el videojuego, y es por lo mismo que se requiere de una explicación de cada una de una forma más detallada y en que difiere cada una de las versiones entregables.

### **7.1.1 PC**

Uno de los requisitos principales para poder utilizar el videojuego en un Computador o Notebook, es que éste disponga de una conexión a internet, puesto que esta versión se encontrará disponible en la página web del videojuego para jugar de forma online como también para descargar y jugar desde el escritorio (previa instalación).

Ésta versión difiere un poco de la versión para dispositivo móvil Android ya que ésta contiene un apartado en donde serán configurable los botones que realizarán los movimientos en el videojuego (de forma nativa en Unity).

### **7.1.2 Android**

Ésta plataforma de juego es la que más requisitos trae consigo, puesto que tiene ciertas limitaciones de hardware y por la cual se recomienda poseer un dispositivo con las características mínimas anteriormente descritas (en el Cap. 5.1.2).

La versión para Android es un poco distinta a la versión Web/PC puesto que se limita un poco la calidad gráfica del juego (por calidad de pantalla y resolución) y que ha sido reducida para un amplio y fluido modo de juego.

Para poder jugar esta versión del juego se requiere del uso de una cuenta Google Games (incorporada con los dispositivos Android) y descargar la aplicación con extensión .apk desde Google Play o directamente desde la página web del juego.



## **7.2 Publicación de la Aplicación**

Como anteriormente se mencionaba, para poder lograr un alcance notorio con los usuarios del videojuego, primero es necesario publicar en todos los canales distribuibles el juego, para esto se explicará cómo se realiza cada una de las tareas para llevar esto a cabo.

### **7.2.1 Google Play**

Google Play es sin duda, una de las tiendas de aplicaciones con mayor número de descargas desde su fundación en agosto del 2008, por lo mismo una gran estrategia al momento de crear un juego, es tener como objetivo un dispositivo Android como plataforma de juego.

Para hacer la publicación de una aplicación en Google Play, primero es necesario contar con una cuenta Google Play Developer, que permite acceder a una consola de desarrollo y publicación de aplicaciones, junto con un sistema de seguimiento y publicidad.

Posteriormente es necesario crear un perfil con información de la aplicación a publicar, que permitirá al usuario informarse de características funcionales y estéticas de la aplicación antes de descargarla.

Antes de lanzar la aplicación a producción, es necesario responder una encuesta de calidad y rango objetivo de clientes, con el cual Google traza las categorías en las que se publicará el juego (por ejemplo: Plataformas, Adolescente).

En la Figura 40 se ilustra el perfil de GeekNator en Google Play Store.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

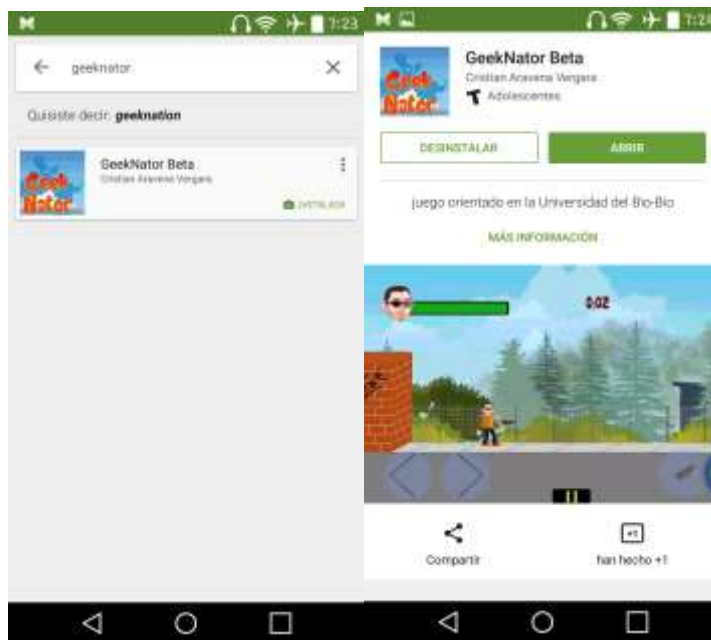


Figura 40, Perfil de GeekNator en Google Play Store.

### 7.2.2 Servidor Web

WebGL es una especificación estándar que está siendo desarrollada actualmente para mostrar gráficos en 3D en navegadores web. El WebGL permite mostrar gráficos en 3D acelerados por hardware (GPU) en páginas web, sin la necesidad de plug-ins en cualquier plataforma que soporte OpenGL 2.0 u OpenGL ES 2.0. Nos permitirá correr nuestro juego hecho en Unity en un navegador sin necesidad de instalar ningún plug-in.

Ésta tecnología genera un Script en JavaScript, que convertirá nuestros Scripts C# en JavaScript para posteriormente ejecutarse en la página Web.

Cuando compilamos un proyecto WebGL, Unity creará una carpeta con los siguientes archivos:

- un archivo index.html que incrusta el contenido en una página web.
- un archivo JavaScript que contiene el código para el reproductor.
- un archivo .mem que contiene la imagen binaria para inicializar la memoria heap para el reproductor.
- un archivo .data que contiene los datos de assets y escenas.

Algunos archivos JavaScript de ayuda para inicializar y cargar el reproductor.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

Para Reproducir de forma estable el juego, la página web que contiene toda la información del videojuego es iniciada a partir de los archivos generados por Unity WebGL.

Para instalar el Gestor y Reproductor en el servidor, es necesario importar librerías de Apache antes de generar el ejecutable del videojuego, para ahorrar tiempos de carga y contenido en segundo plano (teniendo en cuenta que el servidor local de la carrera es Apache).

Finalmente, se realiza una Compilación de lanzamiento, que generará los archivos anteriormente mencionados y una página web simple donde estará almacenado el Plug-in de Unity que permitirá la reproducción del videojuego.

En las figuras 41 y 42 se ilustra la carga del plug in y posterior juego en funcionamiento en la web.



Figura 41, carga de plug-in en página web.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity



Figura 42, Juego cargado en WebGL

# Conclusiones

---

Después haber finalizado la implementación del proyecto, se pueden sacar las siguientes conclusiones:

- El desarrollo de un videojuego, implica una fuerte participación de diversas disciplinas artísticas, como el diseño gráfico, por lo que se debió lidiar con nuevos desafíos, tanto como adquirir nuevos conocimientos, así como también establecer una comunicación con integrantes de otras especialidades. Es aquí donde entra en juego la participación y cooperación de la empresa Jerbosis Art, cuyo líder ayudó a dar movimiento, vida y el toque artístico a GeekNator.
- Sin duda, la utilización de herramientas especializadas, como el caso del motor Unity, ayudan de sobremano en la producción y desarrollo de software complejo, como lo es un videojuego. Permitiendo así, por ejemplo, que un equipo de trabajo reduzca costos, tiempo y complejidades, a la hora de implementar un proyecto. Por lo tanto, es muy útil buscar tecnologías apropiadas, que entreguen éste tipo de ventajas a la hora de emprender un proyecto.
- Al utilizar una herramienta, como Unity, la cual basa la mayor parte de su funcionamiento en la utilización de componentes, se tiene un software extensible y de fácil mantenimiento.
- Además, es conveniente mencionar la importancia de seleccionar herramientas de desarrollos con una completa documentación y soporte, como lo es el caso de Unity (a diferencia de otras como Torque o GameMaker), siendo un factor clave a la hora de utilizar y conocer todas las funcionalidades que la plataforma ofrece, no dejando de lado la gran ayuda que aporta a la solución de problemas.
- Con éste trabajo, se mostró que no está muy alejado de la realidad de las empresas chilenas, implementar un videojuego 2D Móvil, con un notable rendimiento y en un plazo justo.

Diseño e implementación de un videojuego de acción-aventura 2D para dispositivos Android utilizando el motor Unity

- El logro de la implementación de la totalidad de las funcionalidades propuestas al inicio de éste desarrollo, cumpliendo así, con uno de los objetivos fundamentales de todo proyecto, la planificación trazada.

## **Resultados Obtenidos**

Como principales resultados obtenidos, luego de haber finalizado el desarrollo del videojuego, se pueden mencionar los siguientes:

- Un videojuego respaldado con componentes y funcionalidades de nivel profesional, que sumerge al usuario final en una experiencia única.
- Una aplicación descargable desde Google Play y el sitio web propio construido y suministrado por la Universidad del Bio-Bío.

## **Proyección a Futuro**

En cuanto a los futuros trabajos y proyectos que podrían realizarse en base a GeekNator, existen diversas opciones, destacando especialmente las que siguen:

- Agregar más escenas, enemigos y niveles de dificultad, permitiendo al jugador explorar otros mundos, con características y retos propios.
- Permitir a los usuarios personalizar su personaje.
- Incorporar Realidad Aumentada y Eventos en fechas especiales, lo cual está de moda en los videojuegos de este año
- Agregar nuevas habilidades al personaje principal, como por ejemplo, subir de nivel (fortalecer su fuerza, vida o armas), obtener nuevas armas e ítems, la posibilidad de poder nadar, volar, etc.

En fin, un mundo de posibilidades, que solo se ven limitadas por la imaginación y creatividad de quienes desarrollen el proyecto.

# Bibliografía

---

- [1] Goldstone, Will. Unity Game Development Essentials [En línea]  
<<http://www.enucomp.com.br/2012/conteudos/minicursos/unity.pdf>>.  
[Consulta: 6 de Octubre 2015]
- [2] Definición de Transformaciones Geométricas [En línea]  
<<http://www.sangakoo.com/es/temas/transformaciones-geometricas>>. [Consulta: 6 de Octubre 2015]
- [3] UNITY TECHNOLOGIES. Unity3D: Manual. 2010. [En línea].  
<<http://docs.unity3d.com/es/current/Manual/class-Camera.html>>.  
[Consulta: 21 de Octubre 2015]
- [4] UNITY TECHNOLOGIES. Unity3D: Manual. 2010. [En línea].  
<<http://docs.unity3d.com/es/current/Manual/class-SpriteRenderer.html>>.  
[Consulta: 21 de Octubre 2015]
- [5] Diez, Marcos. 2012, ¿Qué es un motor gráfico? [En línea]  
<<https://marcosdiez.wordpress.com/2012/11/14/motor-grafico-y-eso-que-es/>>.  
[Consulta: 21 de Octubre 2015]
- [6] Candil, Daniel. Cuatro motores gráficos para perder el miedo y lanzarse al desarrollo de videojuegos [En línea]  
<<http://www.vidaextra.com/listas/4-motores-graficos-para-perder-el-miedo-y-lanzarse-al-desarrollo-de-videojuegos>>  
[Consulta: 28 de Octubre 2015]
- [7] Shiva Technologies SAS [En línea]  
<<http://www.shivaengine.com/>>  
[Consulta: 09 de Noviembre 2015]

[8] Game Maker Studio [En línea]

<<http://www.yoyogames.com/studio>>.

[Consulta: 09 de Noviembre 2015]

[9] UNITY TECHNOLOGIES. Unity2D [En línea]

<<http://unity3d.com/es/pages/2d-power>>.

[Consulta: 09 de Noviembre 2015]

[10] Torque Technologies Inc. [En línea]

<<http://www.garagegames.com/products/torque-3d>>

[Consulta: 09 de Noviembre 2015]

[11] Nielsen, J.: Heuristic evaluation. Ed. Nielsen, J., and Mack, R.L.: Usability Inspection Methods, John Wiley & Sons, New York, NY (2004).

[12] González Sanchez, J. L.; Padilla Zea, N.; Gutiérrez, F. L.; Cabrera, C.: De la Usabilidad a la Jugabilidad: Diseño de Videojuegos Centrado en el Jugador, Actas de Interacción 2008, pp. 99-109 (2008).

[13] Espinoza Vivanco, M.: Desarrollo de Juego Educativo RPG en teléfonos Móviles. Memoria para optar al título de Ingeniero Civil en Computación, Universidad de Chile. (Octubre 2009).

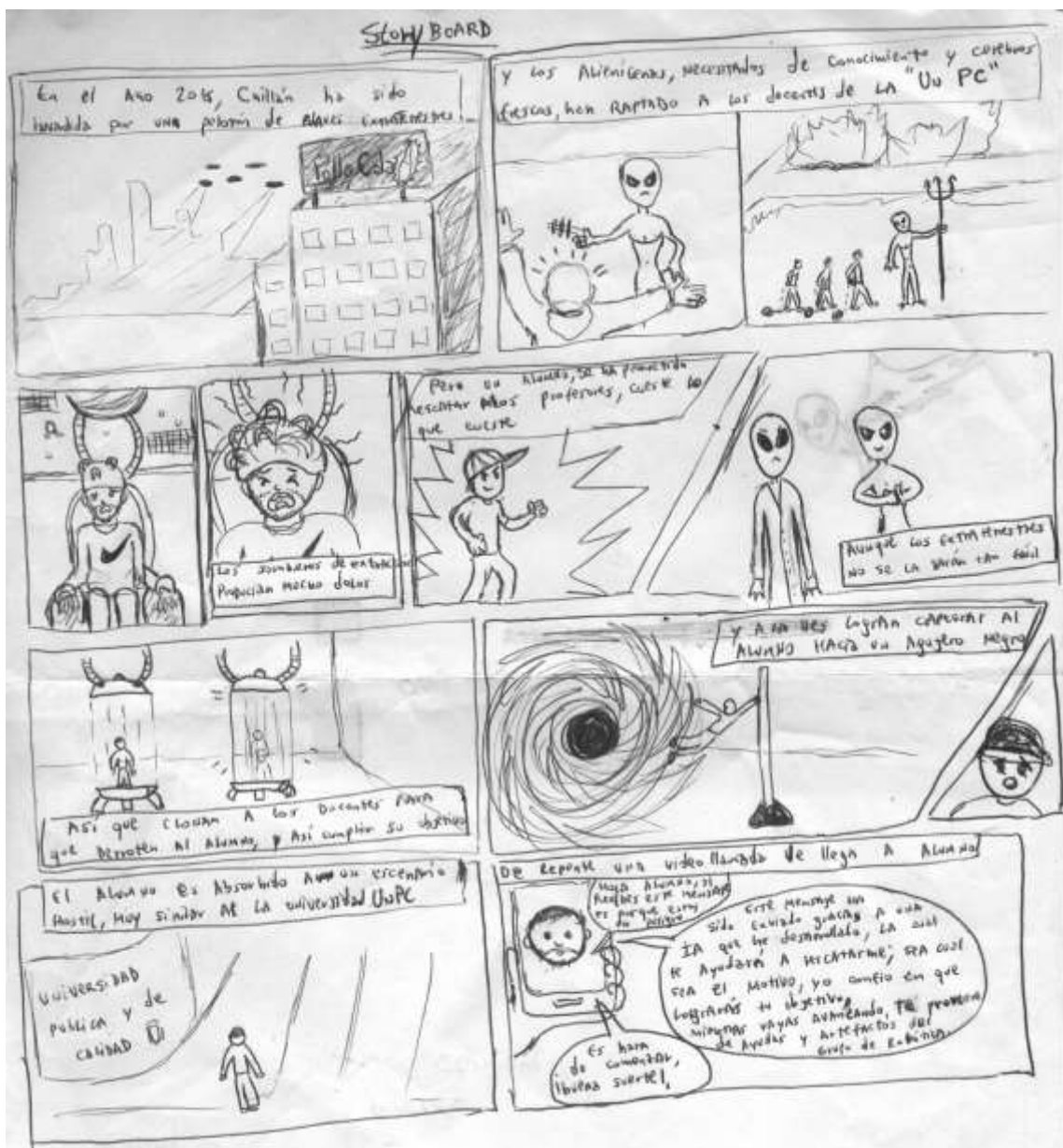


# Anexos.

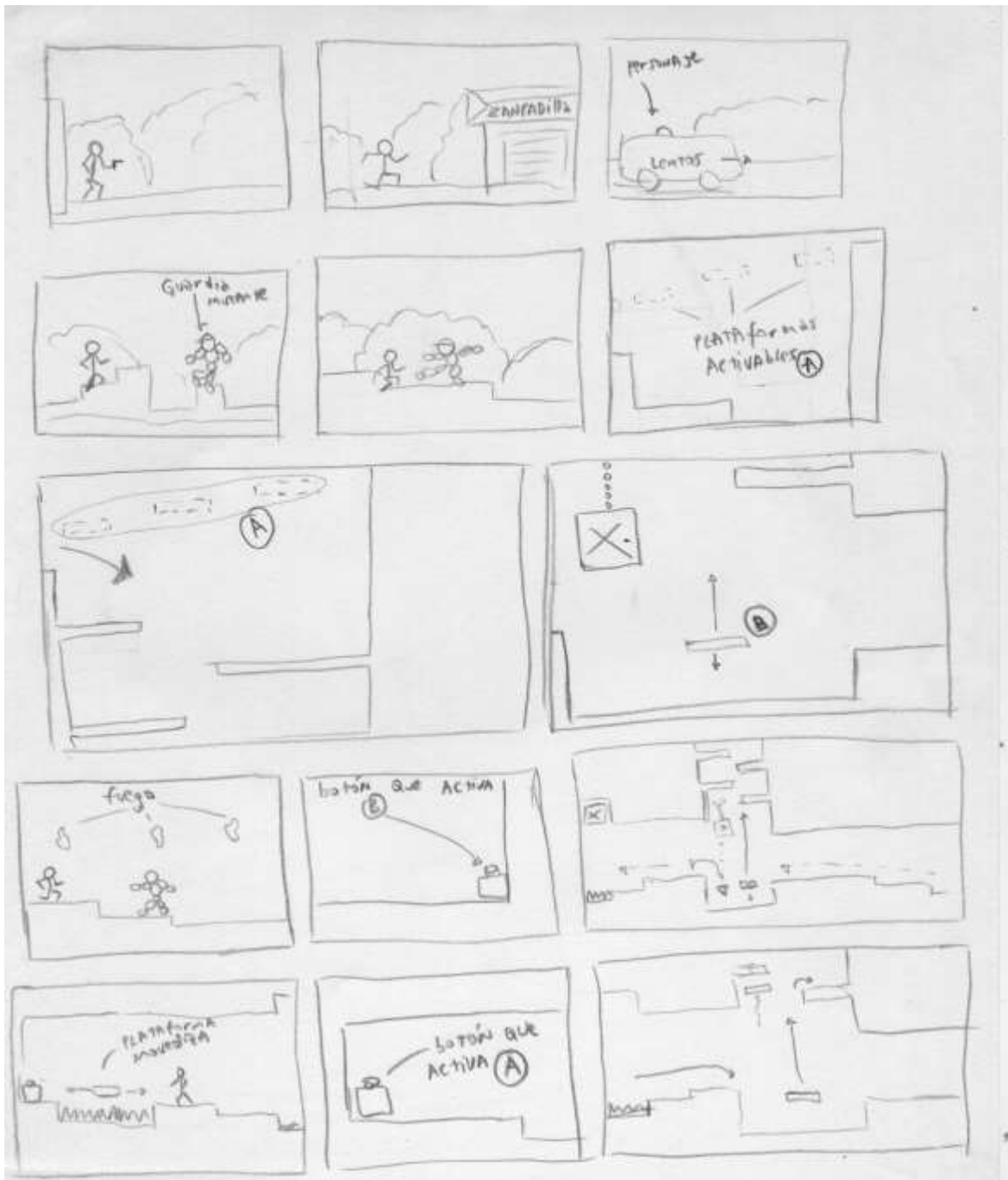
## Anexo 1: Storyboard

En este Anexo, se ilustra los bocetos realizados para bosquejar la jugabilidad e interacción del videojuego.

### Inicio:

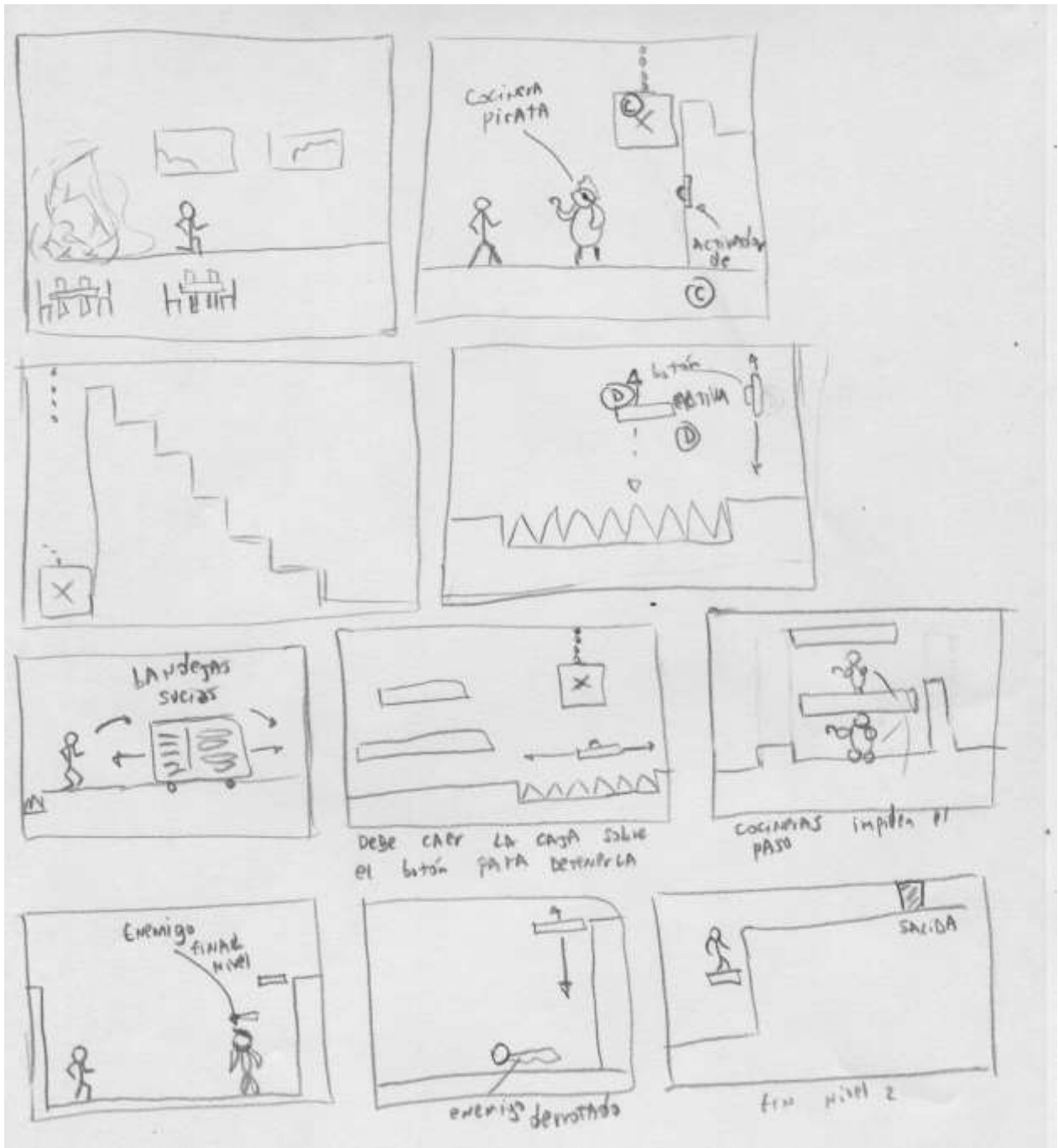


### Escenario 1:

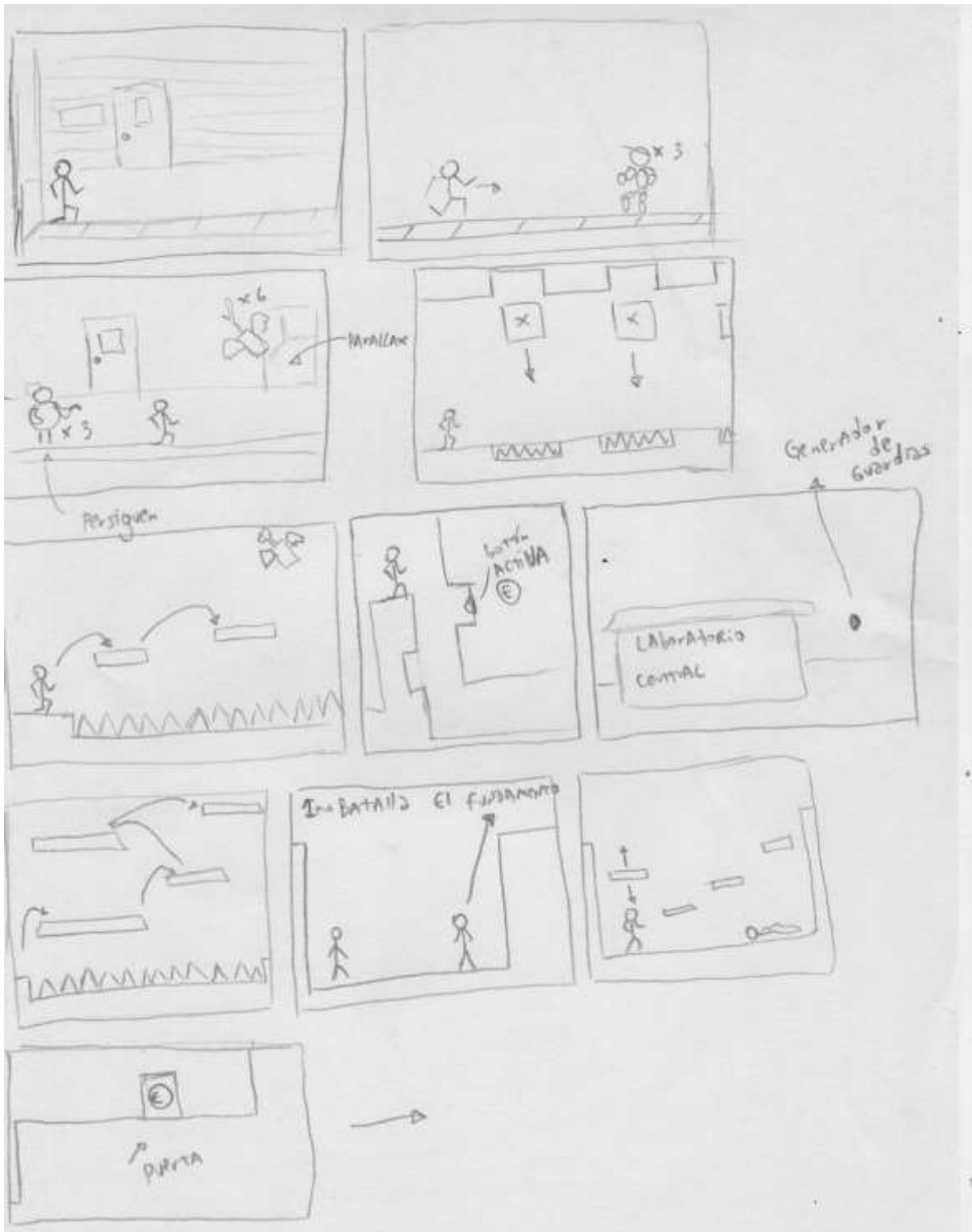


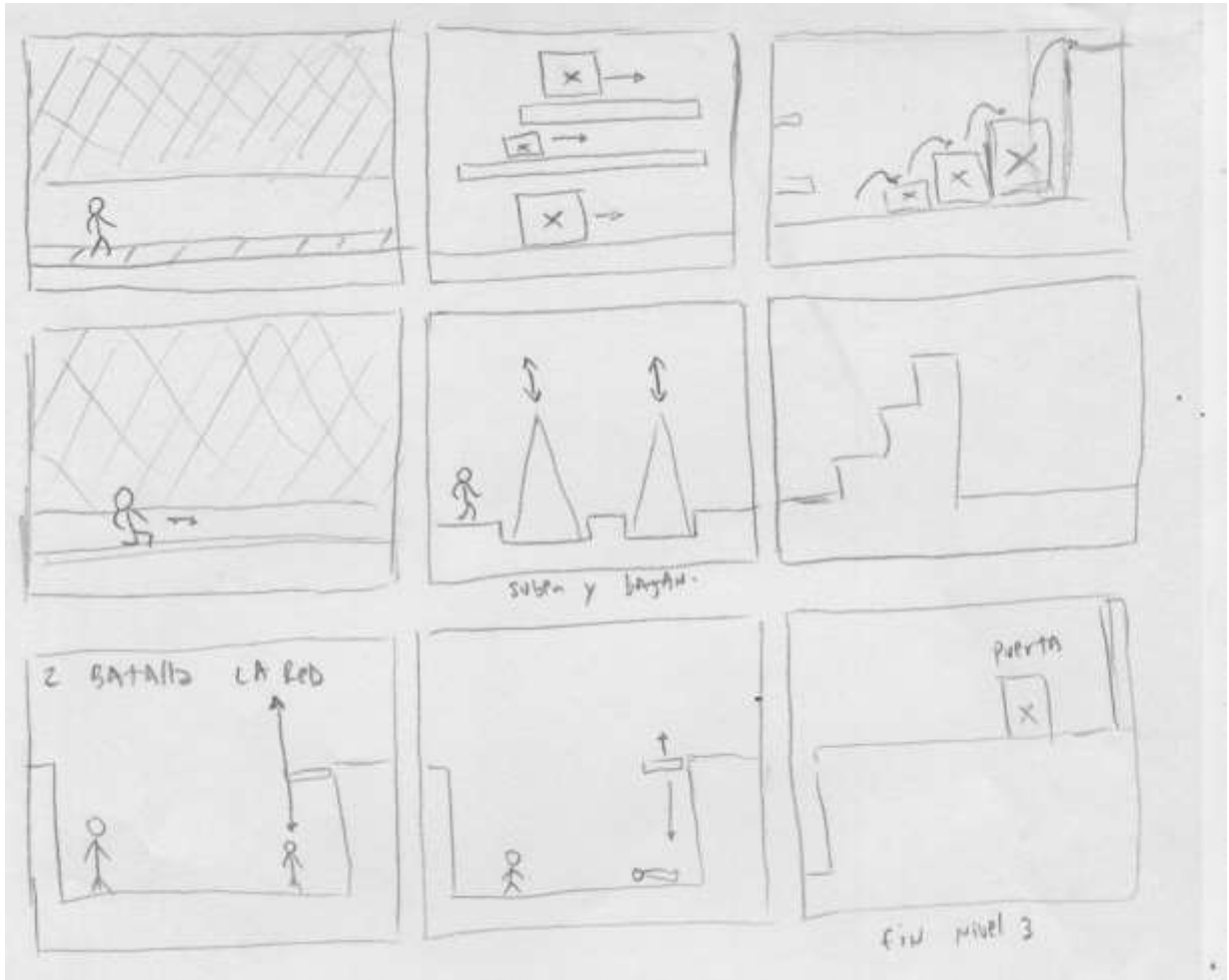


## Escenario 2:



### Escenario 3:





## Anexo 2: Paleta de colores

En el siguiente anexo se ilustrará el código HTML de la paleta de colores con la que se realizó todo el diseño del videojuego.

	
HTML code: #92D5F0	HTML code: #607151
	
HTML code: #83BED0	HTML code: #566649
	
HTML code: #A55542	HTML code: #7EA296
	
HTML code: #934D3C	HTML code: #729089
	
HTML code: #40BFE0	HTML code: #94C1C6
	
HTML code: #37AFCA	HTML code: #87ACB3
	
HTML code: #AFE1EC	HTML code: #6B8426
	
HTML code: #9DCAD8	HTML code: #5F7723
	
HTML code: #1AB2D9	HTML code: #484E22
	
HTML code: #19A0C6	HTML code: #42461E
	
HTML code: #1D9FD1	HTML code: #9D8767
	
HTML code: #1A91BC	HTML code: #91795D
	
HTML code: #2780BC	HTML code: #C69C8C
	
HTML code: #2374AB	HTML code: #B48D7F
	
HTML code: #1C90D4	HTML code: #7D473B
	
HTML code: #1E80B5	HTML code: #6F4036
	
HTML code: #505F35	HTML code: #303636
	
HTML code: #4A5430	HTML code: #2B312F
	
HTML code: #D2BEB6	HTML code: #AD6D5B
	
HTML code: #C8AFA7	HTML code: #332C33
	
HTML code: #8C5B13	HTML code: #AC6D5B
	
HTML code: #825211	HTML code: #9B6352
	
HTML code: #6E8571	HTML code: #354448
	
HTML code: #637865	HTML code: #C8B9B6

## Anexo 3: Scripting

### Clase AIFollow:

```

using UnityEngine;
using System.Collections;
using Pathfinding;

[RequireComponent (typeof(Seeker))]
[RequireComponent (typeof(CharacterController))]
[AddComponentMenu("Pathfinding/AI/AIFollow (deprecated)")]
public class AIFollow : MonoBehaviour {

    public Transform target;
    public float repathRate = 0.1F;
    public float pickNextWaypointDistance = 1F;
    public float targetReached = 0.2F;
    public float speed = 5;
    public float rotationSpeed = 1;

    public bool drawGizmos = false;
    public bool canSearch = true;
    public bool canMove = true;
    protected Seeker seeker;
    protected CharacterController controller;
    protected NavmeshController navmeshController;
    protected Transform tr;
    protected float lastPathSearch = -9999;
    protected int pathIndex = 0;
    protected Vector3[] path;

    public void Start () {
        seeker = GetComponent<Seeker>();
        controller = GetComponent<CharacterController>();
        navmeshController = GetComponent<NavmeshController>();
        tr = transform;
        Repath ();
    }

    public void Reset () {
        path = null;
    }

    public void OnPathComplete (Path p) {

        StartCoroutine (WaitToRepath ());

        if (p.error) {
            return;
        }

        path = p.vectorPath.ToArray();

        float minDist = Mathf.Infinity;
        int notCloserHits = 0;

        for (int i=0;i<path.Length-1;i++) {
            float dist = AstarMath.DistancePointSegmentStrict
(path[i],path[i+1],tr.position);
            if (dist < minDist) {
                notCloserHits = 0;
                minDist = dist;
                pathIndex = i+1;
            } else if (notCloserHits > 6) {
                break;
            }
        }
    }
}

```



```

public IEnumerator WaitToRepath () {
    float timeLeft = repathRate - (Time.time-lastPathSearch);

    yield return new WaitForSeconds (timeLeft);
    Repath ();
}

public void Stop () {
    canMove = false;
    canSearch = false;
}

public void Resume () {
    canMove = true;
    canSearch = true;
}

public virtual void Repath () {
    lastPathSearch = Time.time;

    if (seeker == null || target == null || !canSearch || !seeker.IsDone ()) {
        StartCoroutine (WaitToRepath ());
        return;
    }

    Path p = ABPath.Construct(transform.position,target.position,null);
    seeker.StartPath (p,OnPathComplete);

}

/** Start a new path moving to \a targetPoint */
public void PathToTarget (Vector3 targetPoint) {
    lastPathSearch = Time.time;

    if (seeker == null) {
        return;
    }

    seeker.StartPath (transform.position,targetPoint,OnPathComplete);
}

public virtual void ReachedEndOfPath () {
}

public void Update () {

    if (path == null || pathIndex >= path.Length || pathIndex < 0 || !canMove) {
        return;
    }

    Vector3 currentWaypoint = path[pathIndex];
    currentWaypoint.y = tr.position.y;
    while ((currentWaypoint - tr.position).sqrMagnitude <
pickNextWaypointDistance*pickNextWaypointDistance) {
        pathIndex++;
        if (pathIndex >= path.Length) {
            if ((currentWaypoint - tr.position).sqrMagnitude <
(pickNextWaypointDistance*targetReached)*(pickNextWaypointDistance*targetReached)) {
                ReachedEndOfPath ();
                return;
            } else {
                pathIndex--;
                break;
            }
        }
        currentWaypoint = path[pathIndex];
        currentWaypoint.y = tr.position.y;
    }

    Vector3 dir = currentWaypoint - tr.position;
}

```

```

        tr.rotation = Quaternion.Slerp (tr.rotation, Quaternion.LookRotation(dir),
rotationSpeed * Time.deltaTime);
        tr.eulerAngles = new Vector3(0, tr.eulerAngles.y, 0);
        Vector3 forwardDir = transform.forward;
        forwardDir = forwardDir * speed;
        forwardDir *= Mathf.Clamp01 (Vector3.Dot (dir.normalized, tr.forward));
        if (navmeshController != null) {
        } else if (controller != null) {
            controller.SimpleMove (forwardDir);
        } else {
            transform.Translate (forwardDir*Time.deltaTime, Space.World);
        }
    }

    public void OnDrawGizmos () {

        if (!drawGizmos || path == null || pathIndex >= path.Length || pathIndex < 0) {
            return;
        }

        Vector3 currentWaypoint = path[pathIndex];
        currentWaypoint.y = tr.position.y;

        Debug.DrawLine (transform.position,currentWaypoint,Color.blue);

        float rad = pickNextWaypointDistance;
        if (pathIndex == path.Length-1) {
            rad *= targetReached;
        }

        Vector3 pP = currentWaypoint + rad*new Vector3 (1,0,0);
        for (float i=0;i<2*System.Math.PI;i+= 0.1F) {
            Vector3 cP = currentWaypoint + new Vector3 ((float)System.Math.Cos
(i)*rad,0,(float)System.Math.Sin(i)*rad);
            Debug.DrawLine (pP,cP,Color.yellow);
            pP = cP;
        }
        Debug.DrawLine (pP, currentWaypoint + rad*new Vector3 (1,0,0),Color.yellow);
    }
}

```

### Clase Parallaxing:

```

using UnityEngine;
using System.Collections;
public class Parallaxing : MonoBehaviour {

    public Transform[] backgrounds;
    private float[] parallaxScales;
    public float smoothing;
    private Vector3 previousCameraPosition;

    void Start () {
        previousCameraPosition = transform.position;
        parallaxScales = new float [backgrounds.Length];
        for (int i =0; i<parallaxScales.Length;i++){
            parallaxScales[i] = backgrounds[i].position.z*-1;
        }
    }
    void LateUpdate(){
        for (int i=0; i<backgrounds.Length;i++){
            Vector3 parallax = (previousCameraPosition - transform.position)*
(parallaxScales[i] /smoothing);
            backgrounds[i].position = new Vector3 (backgrounds[i].position.x+ parallax.x,
backgrounds[i].position.y+parallax.y,backgrounds[i].position.z);
        }
        previousCameraPosition = transform.position;
    }
}

```

## Clase GameMaster:

```

using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class GameMaster : MonoBehaviour {
    public static GameMaster instancia;
    public GameObject player;
    public Text puntuationText;
    public Text relojTiempo;
    public Canvas fondoPausa;
    public bool isPause;
    public Canvas BotonesPause;
    public Canvas VerificadorReinicio;
    public Canvas VerificadorSalir;
    public Transform particulas;
    public GameObject camaraPuntaje;
    public Canvas ojo;
    public AudioClip itemSoundClip;
    public GameObject CanvasMenu;
    public Camera camaraParallax;
    public GameObject MiguelPincheira;
    public bool pausatiempo = false;

    //=====
    private bool clockRunning;
    private int time;
    private int minutos=0;
    //=====
    private Vector3 startPosition;
    private int puntuation;

    // =====
    void Start () {
        GameMaster.instancia = this;
        this.puntuation = 0;
        BotonesPause.enabled = false;
        VerificadorReinicio.enabled = false;
        VerificadorSalir.enabled = false;

        this.startPosition = GameObject.Find("personajePrincipal new").transform.position;
        UpdatePuntText ();
        Time.timeScale = 1;

        this.clockRunning = true;
        this.time = 0;
        //instancia.audio.Play ();
        StartCoroutine (Clock());
    }
    // =====

    void UpdatePuntText () {
        this.puntuationText.text = "- " + this.puntuation + " -";
    }
    // =====
    public void AddPuntuation(int amount){
        this.puntuation += amount;
        this.UpdatePuntText ();
    }
    // =====

    public static void KillEnemy(GameObject enemy){
        instancia._KillEnemy (enemy);
    }

    public void _KillEnemy (GameObject _enemy){
        instancia.AddPuntuation (_enemy.GetComponent<SaludScript>().puntaje);
        GameObject _clone= Instantiate(_enemy.GetComponent<SaludScript>().particulas,
        _enemy.transform.position, Quaternion.identity)as GameObject;
        Destroy (_clone, 1f);
    }
}

```

```

        Destroy (_enemy.gameObject);
        audio.Play ();

    }
    // =====

    public void UpdateTiempo( int valor){
        if (valor <= 9) {
            this.relojTiempo.text = " " + minutos + ":0" + valor + " ";
        } else {
            this.relojTiempo.text = " " + minutos + ":" + valor + " ";
        }
        if (valor == 60) {
            this.relojTiempo.text = " " + minutos + ":00" ;
        }
    }

    IEnumerator Clock(){
        while (this.clockRunning) {
            if(pausatiempo==false){
                Debug.Log("tiempo: "+time);
                time++;
                this.UpdateTiempo(time);
                if(time== 59)
                    minutos++;

                if(time >=60){
                    time=0;

                }
            }
            yield return new WaitForSeconds(1.0f);
        }
    }

    public void OnPause(){
        BotonesPause.enabled = true;
        fondoPausa.enabled = true;
        Time.timeScale = 0;

    }

    public void Continuar(){
        fondoPausa.enabled = false;
        BotonesPause.enabled = false;
        Time.timeScale = 1;

    }

    public void ReiniciarNivel(){
        BotonesPause.enabled = false;
        VerificadorReinicio.enabled = true;

    }

    public void BotonNO(){
        BotonesPause.enabled = true;
        VerificadorReinicio.enabled = false;
        VerificadorSalir.enabled = false;

    }

    public void BotonSiReinicio(){
        Application.LoadLevel(Application.loadedLevelName);

    }

    public void BotonSalir(){
        BotonesPause.enabled = false;
        VerificadorReinicio.enabled = false;
        VerificadorSalir.enabled = true;

    }

    public void IsExit(){
        Application.Quit ();

    }

    //=====

```

```

public void ChangeLevelCO(){
    StartCoroutine (ChangeLevel ());
}

IEnumerator ChangeLevel(){

    float fadeTime = GameManager.instancia.GetComponent<Fading>().BeginFade(1);
    yield return new WaitForSeconds(fadeTime);
    ojo.enabled = false;
    camaraPuntaje.SetActive(true);
    GameObject.Find ("personajePrincipal new").GetComponent<movimientoPlayer>
().enabled=false;
    fadeTime = GameManager.instancia.GetComponent<Fading>().BeginFade(-1);

}

public void ActivaCamaraFinalNivel(){
    StartCoroutine (CamaraFinalNivelCR ());
}

IEnumerator CamaraFinalNivelCR(){
    float fadeTime = GameManager.instancia.GetComponent<Fading>().BeginFade(1);
    yield return new WaitForSeconds(fadeTime);

    Application.LoadLevel (Application.loadedLevel+1);
}

public void puntuaPantalla(){
    noDestruye.estado.setPunt (puntuacion);
}
public void VolverAMenuPrin(){
    Application.LoadLevel ("menuPrincipal");
}

public void MuerePlayer(){
    camaraParallax.audio.Stop ();
    CanvasMenu.SetActive (true);
}

public void activaMiguel(){
    MiguelPincheira.SetActive (true);
    pausatiempo = true;
}

public void desactivaMiguel(){
    Time.timeScale = 1;
    pausatiempo = false;
    MiguelPincheira.SetActive (false);
}
}
}

```

### Clase MovimientoPlayer:

```

using UnityEngine;
using System.Collections;

public class movimientoPlayer : MonoBehaviour {
    public float speed = 7.0f;
    public float rotationSpeed = 7.0f;
    public float velocidadSalto = 12.0f;
    bool mirarDerecha =true;
    Animator anim;
    public bool enTierra;
    public float tiempoSalto, retrasoSalto=0.5f;
    private bool salto;
    public Transform bordeTierra;
    public float direccion=1;
    public float move;
    Rigidbody2D mybody;
    public float hInput=0;
}

```

```

void Start () {
    anim= GetComponent<Animator>();
    hInput = 0;
}

void Update() {

    if (Input.GetKey (KeyCode.K)) {
        saltar ();
    }
    if (Input.GetKey (KeyCode.J)) {
        Disparar();
    }

    Raycasting ();
    movimiento (hInput);
}

void Raycasting(){
    Debug.DrawLine (this.transform.position, bordeTierra.position, Color.red);
    enTierra = Physics2D.Linecast (this.transform.position, bordeTierra.position, 1 <<
LayerMask.NameToLayer ("Piso"));
    tiempoSalto -= Time.deltaTime;
    if (tiempoSalto<=0 && enTierra && salto) {
        anim.SetTrigger("Caer");
        salto = false;
    }
}

public void saltar ()
{
    if (enTierra) {
        audio.Play ();
        rigidbody2D.velocity = new Vector2 (rigidbody2D.velocity.x, velocidadSalto);
        anim.SetTrigger ("Saltar");
        tiempoSalto = retrasoSalto;
        salto = true;
    }
}

public void movimiento(float hInput){
    if (hInput!=0) {
        float movimiento = hInput;
        move= movimiento;
        anim.SetBool ("caminar", true);
        rigidbody2D.velocity = new Vector2 (movimiento * speed, rigidbody2D.velocity.y);
        if (movimiento > 0 && !mirarDerecha) {
            direccion = 1;
            GirarPersonaje ();
        } else if (movimiento < 0 && mirarDerecha) {
            direccion=-1;
            GirarPersonaje ();
        }
    } else {
        anim.SetBool ("caminar", false);
    }
}

void Move(float horizontalInput){
    Vector2 moveVel = mybody.velocity;
    moveVel.x = horizontalInput * speed;
    mybody.velocity = moveVel;
}

public void startMoving(float horizontalInput){
    hInput = horizontalInput;
}

void GirarPersonaje () {

```

```
        mirarDerecha = !mirarDerecha;
        Vector3 escalada = transform.localScale;
        escalada.x *= -1;
        transform.localScale = escalada;
    }

    public float ZeroToOne(float other){
        float var = 0;
        float delta = 0.25f;
        if (var <= other) {
            var += delta;
            Debug.Log ("aca va: " + var);
        }
        return var;
    }
    public float ZeroToMinusOne(float other){
        float var = 0;
        float delta = 0.25f;
        if (var >= other) {
            var -= delta;
            Debug.Log ("aca va: " + var);
        }
        return var;
    }
    public void Disparar(){
        ArmaScript arma = GetComponentInChildren<ArmaScript> ();
        if (arma != null) {
            arma.Ataque (false);
        }
    }
}
```

#### **Anexo 4: Pruebas**

En este Anexo se mencionan los beta-tester integrantes de las pruebas de usabilidad y posteriormente la plantilla que se utilizó para las pruebas.

<b>Nombre</b>	<b>Rut</b>	<b>Año Ingreso Universidad</b>
Leonel Muñoz	19.074.439-6	2013
Remigio Fernández	18.789.537-5	2013
Felipe Guzmán	18.430.057-5	2012
Felipe Chávez	19.072.324-0	2014
Lorena Soto	19.071.457-8	2015
Eliseo Suazo	17.989.038-0	2011
Miguel Castillo	18.451.564-4	2012
Jorge Silva	19.823.617-4	2016
Martín Ríos	19.597.736-4	2016
Sebastián Rohr	19.652.168-2	2016
Felipe Sotomayor	19.072.350-K	2013



# Plantilla de Pruebas GeekNator

Nombre: \_\_\_\_\_

Año Ingreso UBB: \_\_\_\_\_ Rut: \_\_\_\_\_

Fecha: \_\_\_\_\_

1.-Siga las instrucciones nombradas por el Evaluador, posteriormente proceda a Responder la siguiente Encuesta.

Pregunta	Ponderación						
	1	2	3	4	5	6	7
¿Me gusta el videojuego?	1	2	3	4	5	6	7
¿El juego es entretenido?	1	2	3	4	5	6	7
¿El juego es desafiante?	1	2	3	4	5	6	7
¿Me sentí controlando las situaciones en el videojuego?	1	2	3	4	5	6	7
¿Supe que hacer en cada situación del juego?	1	2	3	4	5	6	7
¿El juego es fácil de utilizar?	1	2	3	4	5	6	7
¿Las imágenes del juego son claramente identificables?	1	2	3	4	5	6	7
¿El videojuego relata una buena historia?	1	2	3	4	5	6	7
¿Los enemigos son difíciles de vencer?	1	2	3	4	5	6	7
¿La música del videojuego es acorde a las situaciones?	1	2	3	4	5	6	7
¿El menú principal es fácil de utilizar?	1	2	3	4	5	6	7
¿El menú de pausa es fácil de utilizar?	1	2	3	4	5	6	7

- ¿Qué te gustó del juego?

---

---

---

- ¿Qué no te gustó del juego?

---

---

---

- ¿Qué le agregarías al juego?

---

---

---

- Observaciones

---

---

---

---

Firma:

---

¡Muchas Gracias!