UNIVERSIDAD DEL BÍO-BÍO

Facultad de Ciencias Empresariales

Escuela de Ingeniería Civil Informática

# Entrenamiento Automático de Personajes en Videojuegos empleando Descripción Lingüística de Fenómenos Complejos

Proyecto de título para optar el Título de Ingeniero Civil Informático

Heber Mitchel Flores Vigueras

Profesor Guía: Clemente Rubio Manzano

Diciembre 2015

Concepción, Chile

# Agradecimiento

En primer lugar quiero agradecer a mis padres, por brindarme su apoyo durante todos los años de mi educación, empezando desde la básica hasta llegar a la universitaria, estando en los momentos difíciles, siempre diciéndome que soy capaz de lograr, lo que me proponga. A mis hermanos, por alentarme y escuchar mis historias sobre hechos divertidos junto a mis compañeros.

Agradecer a mi profesor guía, el profesor, Clemente Rubio Manzano, por tener la paciencia, el tiempo y la dedicación para guiarme en la construcción de este informe, por brindarme consejos, motivarme en los momentos de flaqueza durante el tiempo en el que desarrolle mi investigación y ser sincero al momento de comunicarme en mis aciertos y errores.

Además agradezco a Edgar Faundez, una gran persona que conocí en un año muy difícil, durante mi formación profesional, siempre apoyándome, conversando y motivándome para que juntos seamos profesionales de calidad.

No olvidar a mi compañero, amigo, hermano en la universidad, Aaron Pavez, quien se convertiría en un pilar para mí, siempre dispuesto a tenderme una mano, cuando lo necesitaba.

No puede dejar fuera a mis compañeros y amigos Gabriela Aguayo, Yoham González, José Luis Acuña y Yerko Carrillo, que se convirtieron en mi grupo de trabajo, en las materias que lo ameritaban, agradecer su compañía y ayuda en situaciones difíciles, siempre fueron personas agradables con las que compartir, en esos momentos donde la espera era eterna, gracias a ellos se convertían en momentos de alegría y diversión. También agradecer a Leonardo Alarcon, un amigo siempre dispuesto a escuchar y aconsejar cuando el momento lo ameritaba.

Gracias a todas las personas que siempre estuvieron a mi lado en los buenos y malos momentos, se agradece de corazón su presencia en todo momento.

¡Muchísimas Gracias!

Heber Flores Vigueras.

# Índice Figuras



# Índice Tablas

# Índice

# Resumen

El objetivo general de este proyecto de título es, diseñar e implementar un Jugador Virtual, que pueda aprender por sí mismo a jugar juegos de ordenador en 2D.

Como objetivos específicos, se tiene realizar una revisión de la literatura, relacionado con la temática del proyecto, estudiar sobre el Modelo Lingüístico Granular de Fenómenos Complejos, los juegos de ordenador y la programación Lógica Lingüística.

La motivación para desarrollar este proyecto, nació en hacer algo novedoso como tema de investigación y desarrollo, para la obtención del Título de Ingeniero Civil Informático, en una materia con proyección como la Inteligencia Artificial. También se debe describir el Modelo Lingüístico Granular para jugar el juego de ordenador YADY. Por ultimo realizar pruebas de experimentación y evaluación de los resultados.

Embarcarse en una aventura de desarrollar un videojuego, del cual se pudieran obtener características del comportamiento de un Jugador Humano al momento de realizar una partida y a partir de los datos obtenidos, poder realizar un modelo de Descripción Lingüística. Con este modelo, se implementara un Jugador Virtual que pueda aprender a jugar y de esta forma poder moverse por el escenario del videojuego.

Para poder realizar el modelo de Descripción Lingüística, se debe implementar una Tabla de datos que vaya mostrando en tiempo real, información correspondiente a los movimientos del Jugador Humano. Esta información será procesada, para poder utilizar la Descripción Lingüística de Fenómenos Complejos y poder realizar un leguaje natural que deberá ser interpretado por la computadora y así poder crear un Jugador Virtual, que deberá aprender a jugar, utilizando este modelo.

El modelo de Descripción Lingüística, se obtendrá organizando la información conseguida por los datos que se generan cuando un Jugador Humano realiza una partida en el videojuego, desarrollado especialmente para esta investigación.

El desafío de lograr crear este modelo lingüístico involucraba investigar a fondo cómo funcionaba esta técnica, para poder aplicarla de forma correcta en el videojuego.

Todo esto debía complementarse con una interfaz agradable y sencilla hacia el usuario.

Se explicará de forma sencilla de que trata el Aprendizaje Automático, las ventajas y desventajas de usar en los videojuegos, la Inteligencia Artificial, técnicas que se pueden aplicar como alternativas al Aprendizaje Automático en videojuegos, dónde y cuándo se pueda usar el Aprendizaje Automático, de dónde podemos obtener información para el aprendizaje, se hablará de algunos enfoques importantes, para desarrollar el Aprendizaje Automático, tales como de que trata los Árboles de Decisión, Algoritmos Genéticos y sobre la Descripción Lingüística de Fenómenos Complejos y cómo abordar los problemas al aplicar Aprendizaje Automático.

A si mismo se dará a conocer la forma en que se implementaron ciertas funciones del videojuego.

También se mostrará un análisis de resultados, de las pruebas obtenidas de partidas realizadas por jugadores humanos y su contra parte obtenidas de las reglas de las partidas, realizadas por el Jugador Virtual. Se evaluaran los resultados en éxito o fracaso de la partida del Jugador Virtual.

**Palabras claves: Aprendizaje Automático, Descripción Lingüística, Jugador Humano, Jugador Virtual.**

## Estructura y organización del Informe

Este informe se encuentra organizado en 6 capítulos, cuyo contenido se describe a continuación:

En el capítulo 1 (**Introducción**) se describe el tema central de este trabajo, partiendo por las ventajas y desventajas y cómo simular el Aprendizaje Automático.

En el capítulo 2 (**Marco Teórico**) se describe, como obtener datos de comportamientos, algunas técnicas básicas y enfoques para trabajar el Aprendizaje Automático, el concepto de Inteligencia Artificial.

En el capítulo 3 (**Estado del Arte**) nos presenta trabajos relacionados con este proyecto, donde se implementó el Aprendizaje Automático.

En el capítulo 4 (**Implementación**) se describen las clases más importantes en el desarrollo de un Jugador virtual y se presenta un diagrama de estas clases.

En el capítulo 5 (**Pruebas**) se presentan pruebas hechas al ejecutar un entrenamiento sobre un Jugador Virtual y estadísticas de simulaciones hechas al probar el funcionamiento del Jugador virtual.

En el capítulo 6(**Conclusión**) presenta las conclusiones del proyecto, aportes y posibles desarrollos en trabajos futuros.

# 1. Introducción

El objetivo de este capítulo es introducir al lector en el Aprendizaje Automático y su aplicación a los videojuegos. El Aprendizaje Automático se ha utilizado en Ciencias de la Computación para dotar a los computadores de la capacidad de aprendizaje, también se puede emplear en videojuegos para entrenar a los NPC (Non-Player Character)

Para poder entrar en esta temática debemos hacernos unas cuantas preguntas:

- ***¿Qué es el Aprendizaje Automático?***
- ***¿Cuándo podemos usarlo en los videojuegos?***
- ***¿Cómo puedo usarlo en los videojuegos?***

Para esto primero debemos tener una idea de que es el Aprendizaje.

Se puede definir el Aprendizaje como el acto, proceso o experiencia de ganar conocimiento o alguna habilidad. La captura y transformación de la información en una forma utilizable para mejorar el rendimiento.

No se hablará de cómo utilizar el Aprendizaje Automático en los juegos de mesa, como el Monopoly o Scrabble, ni cómo aplicarlos a los juegos de cartas, como el Póker o 21 BlackJack. Este informe no será como un manual, donde se explica paso a paso como aplicar el Aprendizaje Automático en los videojuegos, sino que se explicara cómo funciona, dónde, cuándo y cómo utilizarlo.

También se darán ideas de como poder simularlo en situaciones donde no es fácil aplicarlo y de donde poder obtener la información necesaria, para poder desarrollar un buen Aprendizaje Automático.

## 1.1 Ventajas de usar el aprendizaje automático en los videojuegos

Porque mejora la jugabilidad para las personas, mejora el comportamiento de la Inteligencia Artificial, vuelve más interesante los videojuegos al proporcionar una experiencia más personalizada, la dificultad en los videojuegos va aumentando a medida que se desarrolla una partida, entrega un ajuste más dinámico a la dificultad, el desarrollo

de la Inteligencia Artificial se vuelve menos costoso, ya que el jugador con cada partida o etapa estará enseñando como jugar o desempeñarse a los personajes en el videojuego[1].

Estas serían unas ventajas de usar el aprendizaje en los videojuegos, pero también debemos pensar en las desventajas de usar este método.

## 1.2 Desventajas de usar el Aprendizaje Automático en los videojuegos

La principal desventaja se presenta en la escasa predicción, dificultando la validación en el comportamiento del personaje, ya que podrían surgir comportamientos difíciles de validar y predecir, la Inteligencia Artificial puede quedarse atascada en una rutina de aprendizaje, si el jugador no tiene un comportamiento inteligente o prudente la inteligencia tendría un comportamiento erróneo, en otras palabras, si el jugador se desenvuelve de forma estúpida a medida que avanza por el videojuego, la Inteligencia Artificial tendrá un desempeño similar [1].

Falta de programadores con la experiencia necesaria o muchas veces nada de experiencia en el ámbito del Aprendizaje Automático, provoca que el tiempo en el desarrollo de sea alto.

## 1.3 Alternativas para el Aprendizaje (Como Simularla)[1]

No siempre se puede implementar un Aprendizaje Automático puro, ya que hay situaciones donde la inteligencia Artificial, puede quedar atascada en ciertas rutinas o los datos de entrada, los cuales se utilizaran para enseñar a la máquina, no son los suficientemente adecuados. Para lidiar con estos inconvenientes es que se puede simular el comportamiento de Aprendizaje.

- Algunas alternativas para el aprendizaje serían simular comportamientos o situaciones, pre-programar múltiples niveles de actuación.
- Programar acciones, para situaciones donde se sabe que existirá algún problema.
- Cambiar dinámicamente entre niveles, mientras el jugador avanza por el videojuego.

Se puede realizar una adaptación indirecta al reunir datos pre-definidos, para el uso y decisiones de la IA. Algunos datos que pueden ser relevantes, para obtener un detalle del comportamiento de un jugador en un videojuego podrían ser:

- Tipo de MI muerte con cada arma del videojuego.
- Tipo de MI muerte en cada escenario o nivel del videojuego.
- Tipo de enemigo más probable de aparecer en cada escenario.
- Que oponente suele pasar por mi izquierda o derecha.

Dejar en claro que en videojuegos de un mismo género, los datos que podrían ser necesarios tener en cuenta, pueden ser similares o podrían variar, hasta considerar datos totalmente distintos, todo depende del videojuego dónde se decida integrar el Aprendizaje Automático.

El Aprendizaje Automático en los videojuegos se puede utilizar para enseñar a algunos personajes (NPC) a moverse de forma automática, esta forma de enseñar puede realizarse, después de que un jugador haya proporcionado datos relacionados con el comportamiento que se tuvo en el mismo videojuego o en el mismo momento que el jugador desarrolla su partida el o los personajes aprenderán como moverse por el escenario del videojuego.

## 1.4 Interrogantes sobres el Aprendizaje Automático

### 1.4.1 ¿Dónde podemos usar el Aprendizaje Automático?

El Aprendizaje Automático los podemos usar en entidades de Inteligencia Artificial para poder controlar su comportamiento y en el ambiente del Videojuego, es decir optimizar las reglas del juego, terreno, interfaces, infraestructuras, en personajes que deban intervenir con un desarrollo simple del jugador.

### 1.4.2 ¿Cuándo podemos usar el Aprendizaje Automático?

Podemos usar el aprendizaje de manera off-line y on-line.

Off-line: durante el desarrollo del juego, donde podremos hacer un balance y pruebas. Capacitar a la Inteligencia Artificial contra jugadores expertos y/o el terreno.

On-line: la Inteligencia Artificial se adapta al jugador y al entorno, se puede ajustar la dificultad del juego.

# 2 Marco Teórico

## 2.1 De dónde obtener datos del Comportamiento

Para obtener datos de comportamiento, se debe tener claro, qué comportamiento, queremos replicar, en un entrenamiento, por ejemplo podríamos identificar las situaciones en las que se encuentra un Jugador en un momento, durante realiza una partida, para esto podemos realizar un modelo interno de las situaciones, clasificándolas como buenas o malas, peligrosas o seguras, etc. Así mismo se pueden clasificar todos los componentes del escenario del videojuego, como los escondites o los lugares donde el Jugador podría quedar expuesto al peligro, clasificar las propiedades de los objetos, en comestibles, valiosos, peligros, etc.

Otra forma de obtener datos sobre el comportamiento de un Jugador, sería identificar el Comportamiento Táctico que tiene en la partida del videojuego, podríamos reconocer qué tácticas utiliza, ejemplo, que arma utilizó contra ciertos enemigos, que arma utilizó en cierto ambiente y clasificarlas en que armas funcionan mejor y cuando. Así mismo se puede registrar que tipo de acción realizó un Jugador, tales como correr, emboscar, huir y embocar, obtener arma, buscar enemigo y determinar si las tácticas utilizadas en ese momento fueron apropiadas. En videojuegos de formato multijugador, se puede realizar una captura del comportamiento en las órdenes y decisiones que realiza un Jugador a sus compañeros o los comandos utilizados para ordenar a los NPC del videojuego.

Así como se puede obtener información del Comportamiento Táctico y del comportamiento en las situaciones, podemos aprender del Comportamiento Estratégico. Esto es importante ya que muchas veces dependiendo de las estrategias que se elijan comenzando un videojuego, se pueden obtener beneficios y/o hacer que la dificultad del juego aumente o disminuya. Obtener información de cómo un jugador asignas recurso en un videojuego, tales como madera, comida, oro, tecnología, etc. Identificar como plantea un ataque, cuando lo realiza y dónde ataca, como plantea la defensa y dónde ubica las defensas. Este tipo de información se puede considerar si se quiere aprender del Comportamiento Estratégico [1].

## 2.2 Técnicas básicas del Aprendizaje Automático

Existen algunas técnicas para poder utilizar el Aprendizaje Automático, algunas que podrían considerarse son:

### 2.2.1 Aprender por observación del comportamiento humano

- Esto quiere decir que la IA debe intentar replicar el rendimiento individual de un jugador humano.
- Capturar variedades de conocimiento, personalidades y culturas.

### 2.2.2 Aprender por información

- Personas no programadoras instruyendo el comportamiento de la IA.
- Jugadores que le digan a la IA que haga su voluntad.

### 2.2.3 Aprender de la experiencia

- Jugar contra Humanos y la IA como prueba, durante el desarrollo.
- Buscar errores en el comportamiento y tratar de mejorar el comportamiento.
- Adaptar técnicas y estrategias contra el oponente humano.

## 2.3 Usos de la Inteligencia Artificial

### 2.3.1 IA juegos

- El entretenimiento es el objetivo principal.
  - El jugador debe ganar, pero después de una cerrada lucha.
- Limitaciones en el desarrollo comercial.

### 2.3.2 IA académica

- La exploración de nuevas ideas es el objetivo central.
  - La eficiencia y la optimización son deseables.
- Limitaciones en la investigación académica.

## 2.4 IA: una visión centrada en el aprendizaje

La inteligencia artificial requiere poder enfocarse y profundizar en varios temas, para poder lograr comprenderla y aplicar los conocimientos en otras temáticas.

- Arquitectura y algoritmos.
    - Buscar algoritmos.
    - Inferencia lógica y probabilística.
    - Planificadores.
    - Expertos en la cubierta del sistema.
    - Arquitecturas cognitivas.
    - Técnicas de aprendizaje automático.
- Conocimiento.
    - Representación del conocimiento.
        - Estados de la maquina finitos.
        - Reglas basadas en el sistema.
        - Operadores.
        - Arboles de decisión.
        - Clasificador.
        - Red neuronal.
        - Red bayesiana.
    - Adquisición del conocimiento.
        - Programación.
        - Ingeniería del conocimiento.
        - Aprendizaje automático.
- Interfaz con el medio ambiente.
    - Detección
        - Sensores robóticos (sonar, visión, IR, laser, radar).
        - Visión mecánica.
        - Reconocimiento de voz.
    - Actuación.
        - Navegación.
        - Locomoción.
        - Generador de voz.

## 2.5 ¿Cómo conseguir buenos datos de entradas, para aplicar el Aprendizaje Automático?

Obtener buenos ejemplos es esencial, por lo tanto conseguir suficientes ejemplos siempre es necesario, para realizar una generalización útil de datos de entrada. Debemos evitar los ejemplos repetitivos o que se enfoquen en un solo conjunto de espacios. Se debe considerar que realizar una alargada lista de buenos ejemplos puede tomar mucho tiempo, al momento de realizar pruebas.

Podemos obtener datos de entradas de Jugadores Expertos, que ellos consideren importante al momento de aplicar el Aprendiza Automático en algún videojuego.

También se puede considerar datos de entrada que proporcionen otros sistemas de Inteligencia Artificial o considerar datos de juegos similares.

## 2.6 ¿Cuándo el aprendizaje no está funcionando?

- El conocimiento aprendido es demasiado específico a los ejemplos proporcionados.
- Se ve muy bien en los datos de entrenamiento.
- Puede verse bien en datos de prueba.
- No generalizar a nuevas entradas.

## 2.7 Enfoques para trabajar el Aprendizaje Automatizado.

### 2.7.1 Árboles de Decisión [2]

Un árbol de decisión, es un modelo de predicción utilizado en la Inteligencia Artificial. Un árbol de decisión indica las acciones a realizar en función del valor de una o varias variables. Es una representación con forma de árbol, como lo dice su nombre, cuyas ramas se bifurcan en función de los valores tomados por las variables y que terminan en una acción concreta.

Los nodos representan pruebas de atributos, para cada valor posible de atributo, corresponde un hijo.

Situación que se podrían representar con un Árbol de decisión, sería un caso donde un Jugador, se encuentra con un nuevo Jugador, dependiendo de la situación se podrían dar varios casos.

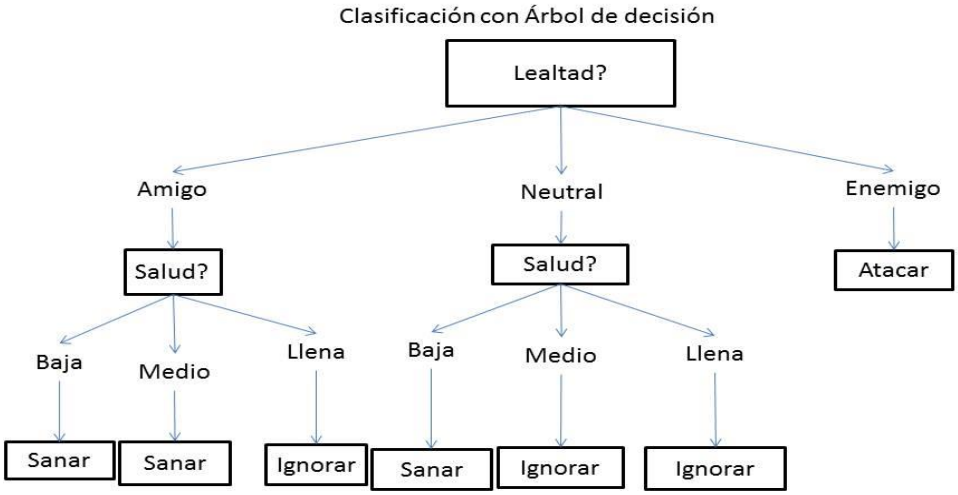


**Figura 1 Ejemplo 1 Diagrama de Árbol.**

Dependiendo de la lealtad del nuevo Jugador, se podría llegar a atacarlo, ignorarlo o sanarlo.

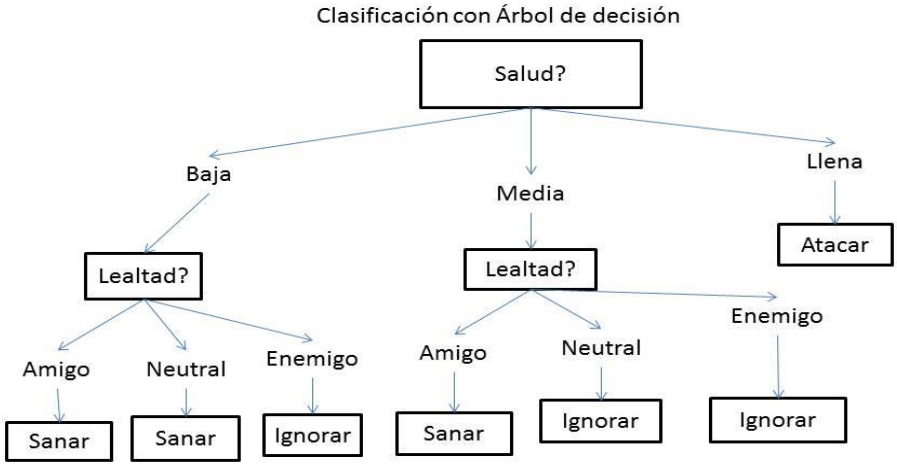


**Figura 2 Ejemplo 2 Diagrama de Árbol.**

En este caso, dependiendo de la Salud del nuevo Jugador, habría que decidir, que opción tomar.

Todos estos eventos se desencadenan evaluando la situación en la que se encuentra un Jugador en un momento X.

### 2.7.2 Algoritmos Genéticos [3]

Los algoritmos genéticos establecen una analogía entre el conjunto de soluciones de un problema y el conjunto de individuos de una población natural, codificando la información de cada solución en un String (vector binario) a modo de cromosoma.

“Se pueden encontrar soluciones aproximadas a problemas de gran complejidad computacional mediante un proceso de *evolución simulada*” John Holland.

Los Algoritmos Genéticos (AG) son métodos adaptativos que pueden ser utilizados para implementar búsquedas y problemas de optimización. Ellos están basados en los procesos genéticos de organismos biológicos, codificando una posible solución a un problema de un “cromosoma” compuesto por una cadena de bits o caracteres.

El algoritmo genético, tiene una estructura como todos los códigos de la programación, cual se compone de: Codificación, Población Inicial, Función Fitness, Selección, Cruzamiento, Mutación.

**Codificación:** los elementos característicos del problema se pueden representar de tal forma que resulte sencilla su implementación y comprensión.

**Población Inicial:** para construir la población inicial, que será la población base de las sucesivas generaciones, existen vario métodos.

**Función Fitness:** asigna a cada cromosoma un número real, que refleja el nivel de adaptación al problema del individuo representado por el cromosoma.

**Selección:** es el proceso por el cual se eligen una o varias parejas de individuos de la población inicial para que desempeñe el papel de progenitores, cruzándose posteriormente y obteniendo descendencia o permaneciendo en la siguiente generación.

**Cruzamiento:** el operador cruce permite el intercambio de información entre individuos de una población, recombinando los cromosomas, dando lugar a nuevos individuos.

**Mutación:** el operador mutación se aplica tras el cruce con el objetivo de incrementar la diversidad poblacional.

### 2.7.3 Descripción Lingüística de Fenómenos Complejos [6]

La Descripción Lingüística de Fenómenos Complejos, se trata de capturar e interpretar datos de fenómenos complejos, de esta forma se pueden producir como resultado informes en un lenguaje natural, los cuales deben ser fáciles de entender incluso para usuarios no expertos en este tema. De esta forma los informes que se generan de la interpretación de los datos, pueden ser personalizables para incluir la información verdaderamente necesaria y relevante, para la audiencia interesada.

Para poder ocupar la Descripción Lingüística de Fenómenos Complejos, se necesita hacer un proyecto de investigación en el área donde se quiere implementar, esta investigación comprende las siguientes etapas:

1.- Analizar cuidadosamente el uso del lenguaje natural en el contexto de la aplicación en la cual se desea implementar la descripción lingüística. La idea de este proceso es encontrar un conjunto de las expresiones más utilizadas para describir los aspectos más relevantes del fenómeno en estudio.

2.- Construir un modelo lingüístico granular del fenómeno en estudio (GLMP de las siglas en ingles Granular Linguistic Model Phenomenon). Esta es una estructura computacional que organiza todas las percepciones relacionadas de una manera similar como los seres humanos suelen organizar su experiencia por medio del lenguaje natural.

3.- Generar un informe personalizado de acuerdo a las necesidades expresadas por los usuarios finales.

4.- Generar un sistema computacional capaz de recoger datos en bruto, procesar e interpretarlos de acuerdo con el definido previamente GLMP, rindiendo el informe que se comunica a los usuarios finales.

La siguiente figura muestra la arquitectura principal de un sistema computacional para generar descripciones lingüísticas pertinentes de fenómenos en diferentes tipos de situaciones. El módulo de adquisición de datos depende de cada aplicación y que podría ser bastante compleja. En este enfoque de análisis, el modelo lingüístico está formado por una ontología con elementos pertenecientes a tres ámbitos diferentes:

a) Los conceptos que pertenecen a la propia aplicación.

b) Los conceptos desarrollados en el contexto de la teoría computacional de percepciones.

c) Los elementos que pertenecen a la propia lengua, por ejemplo: inglés o español, tomado dela lingüística sistémico funcional.

El modulo interprete utiliza el modelo lingüístico para reconocer el significado de los datos de entrada y luego generar expresiones lingüísticas pertinentes.

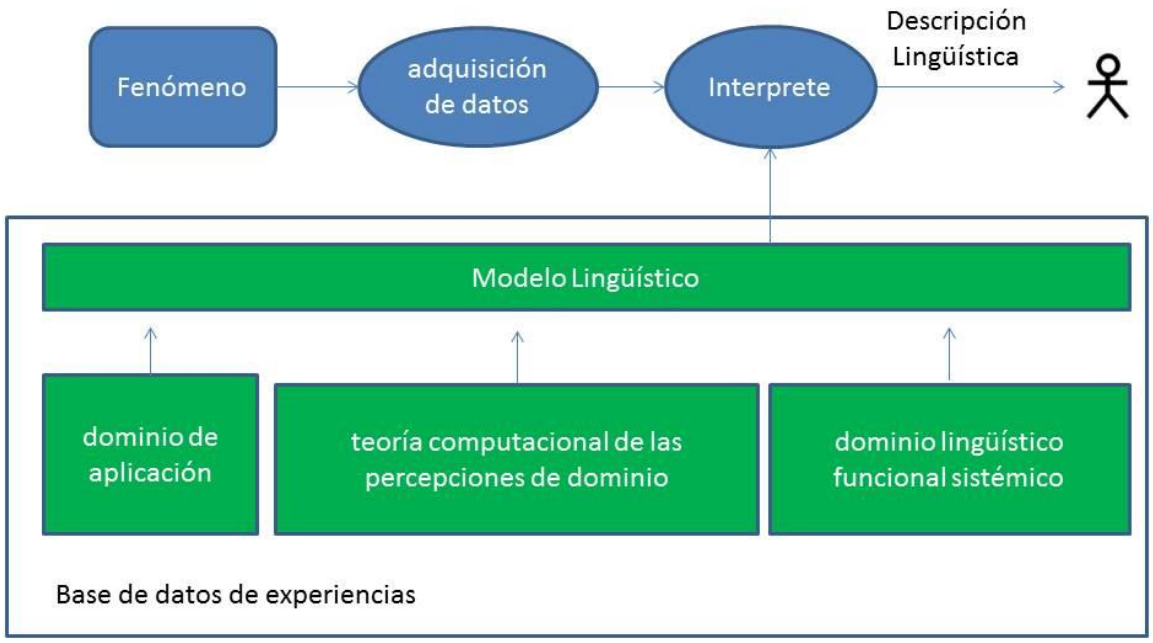


**Figura 3 Representación del Modelo Descripción Lingüística de Fenómenos Complejos.**

# 3 Estado del arte

El Aprendizaje Automático, se ha utilizado en videojuegos, para el entrenamiento de personajes, en el trabajo Jugar Super Mario Bros del Dr. Tom Murphy se aplicó la técnica *Lexicographic Orderings and Time Travel* y en el trabajo de Jugar Atari 2600 se aplicó la técnica de Deep Reinforcement Learning. Estos trabajos serán explicados a continuación.

## 3.1 Automatizar Consola Nintendo, con ordenamiento lexicográfico y tiempo de viajes

*"El primer nivel de Super Mario Bros es fácil con ordenamiento lexicográfico y tiempos de viajes… después de eso se pone un poco difícil." Dr.Tom Murphy, refiriéndose al título de su trabajo.*

Se trata sobre el trabajo del Dr. Tom Murphy [4], quien implementó la automatización en la consola de videojuegos Nintendo.

El sistema de entretenimiento Nintendo es un computador muy pequeño, el cual sólo consta con una memoria de 2kbs de RAM, la cual realmente es una memoria muy pequeña. La idea básica que desarrolló el Dr. Tom, fue grabar el desarrollo de una partida en algunos de los videojuegos de la consola, "grabar", se refiere a registrar los botones que son presionados mientras juega, en este trabajo específicamente mientras juega Super Mario Bros. El Dr. Tom juega Super Mario un poco, pasa un nivel, salta mientras avanza por el juego, pisa algunos gumbas, baja algunas tuberías, llega a alguna bandera o realizar cualquier movimiento. Por cada paso se guarda en la memoria las entradas de los botones. Entonces como la memoria de la consola tiene 2048 bites, estos bites son la memoria que representa un frame en los movimientos de Mario, 2048 en el siguiente frame y así una y otra vez por el tiempo que se haya jugado, en toda la partida del videojuego, se podrían tener unos cuantos miles de frames.

Con todos los botones registrados durante la partida, se tienen unos miles de datos almacenados en la memoria, el siguiente paso es tener claro que significa ganar Super Mario Bros, se identifica que los bites de la memoria deberían ir aumentando o por lo menos algunos de los bites deberían estar creciendo, porque estar ganando en un juego significa que el puntaje vaya creciendo.

Pero esto es realmente un poco más complicado que bites aumentando, porque si uno lo piensa, en el puntaje son varios los bites distintos. Un ejemplo simple podría ser, Super Mario empieza en el nivel 1-1, luego pasa al 1-2,1-3,1-4, pero luego llega al 2-1, así que el número después del guion sube por un tiempo, pero luego baja y el siguiente número aumenta y luego continua aumentando, en el videojuego este patrón es bastante común, es un patrón matemático, llamado Ordenamiento Lexicográfico, es el mismo tipo de patrón que presenta un diccionario o sus similares, así que la idea es tomar todos esos frames, de cuando se jugó Super Mario y tratar de encontrar serie de bites, por su ubicación en la memoria, que aumenten de acuerdo al Ordenamiento Lexicográfico, eso generaliza cosas, como el puntaje subiendo, la posición de Mario en la pantalla como una coordenada X dentro del nivel en que se encuentra y luego los niveles del mundo.

El computador simplemente tomará la ROM, lo emulará y probará algunas entradas, estas entradas son botones que hacen moverse o realizar una acción a Mario, para ver si alguna de las entradas aumentan los bites en la memoria, luego ignora cualquier cosa que un humano estaría viendo o escuchando mientras juega, como el video o los efecto de sonidos, sólo verá los bites y buscará la secuencia de entradas, que los haga aumentar.

El Dr. Tom, dice que esta técnica funciona muy bien. Al aplicar este método para automatizar los videojuegos de la consola Nintendo, suelen suceder cosas bastante inesperadas, como movimiento estúpidos u otros que son realmente sorprendentes ya que son movimiento que un humano no podría realizar por la precisión que se logra al ejecutarlos.

El segundo trabajo es:

## 3.2 Jugar Atari con Aprendizaje por Refuerzo Profundo [5]

Este trabajo trata de cómo se aplicó el aprendizaje por refuerzo profundo, para automatizar los juegos de la consola Atari 2600. El algoritmo de aprendizaje por reforzamiento profundo, intenta aprender de una recompensa escalar, normalmente encuentran secuencias de estados altamente correlacionados [5].

El objetivo de este trabajo fue crear una red neuronal, que fuera capaz de aprender con éxito a jugar la mayor cantidad de juegos posibles. La red neuronal que se desarrolló no recibía ninguna información sobre el diseño o alguna característica visual de los juegos, en la cual se aplicaría más adelante, toda la información que sería importante para poder

jugar la recibiría de la misma forma que lo hace un Jugador Humano cuando prueba un juego.

Para cuando se había concluido el desarrollo de la red neuronal, esta había superado a los algoritmos de aprendizaje reforzado anteriores, en seis de los siete juegos, donde se probó su funcionamiento y había superado a personas encargadas de probar juegos, que se consideran expertas en esta tarea, en los puntajes de tres de los siete juegos en los que se llevó a cabo su implementación.

El funcionamiento de esta red neuronal, era observar una serie de movimientos que son representados por cambios en los pixeles de la imagen del juego. Un cambio en las imágenes debe representar un premio, que es el cambio en el puntaje.

Para la red neuronal es imposible comprender totalmente la situación actual que se está emulando solamente de la pantalla, para que la red pueda comprender mejor la información de la pantalla, se consideran secuencias de acciones y observaciones, aprender de las estrategias que generan estas acciones y observaciones.

Todas estas secuencias se suponen que terminan en un número finito de pasos en el tiempo. Esto trae como consecuencia un largo, pero finito proceso de decisión de Markov, en el que cada secuencia es un estado distinto.

Como resultado se pueden usar métodos simples de aprendizajes por refuerzo para procesos de Markov, simplemente utilizando una secuencia completa y un estado que represente el tiempo.

Para logar el objetivo de automatizar los juegos de la Atari 2600 era necesario conectar un algoritmo de aprendizaje por refuerzo a una red neuronal profunda que opera directamente en las imágenes RGB y procesar eficientemente datos de entrenamientos mediante el uso de cambios de gradiente estocásticas.

En este trabajo las entradas de datos necesarias son frames, los frames que se obtienen de la consola Atari son imágenes de 210 X 160 pixeles de una gama de 128 colores. Los frames obtenidos son pre-procesados, primero convirtiéndolos a una escala de grises en representación RGB y escalando las imágenes a un tamaño de 110 X 84. La representación final de las entradas es obtenida recortando una región de 84 X 84 de la imagen que capta aproximadamente el área del juego.

Como resultado de aplicar estas técnicas a la combinación del aprendizaje por refuerzo a una red neuronal se obtiene un algoritmo denominado Deep Q-Network (DQN).

Este trabajo presenta un nuevo modelo de aprendizaje profundo, por aprendizaje por refuerzo y demuestra su capacidad de dominar las políticas de control de la Atari 2600 en los juegos de computador usando únicamente pixeles primos como entrada de datos.

# 4 Implementación

En el siguiente apartado se dará a conocer al lector las funciones implementadas para la captura de información en la partida de un Jugador Humano, para poder crear una descripción lingüística de fenómenos complejos y poder desarrollar un Jugador Virtual, en el cual su objetivo será aprender de la descripción generada, hacia donde moverse según una situación X.

Para poder realizar una captura de información sobre la partida realizada por un jugador era necesario tener en cuenta que datos serían importantes capturar para poder crear una descripción lingüística. Para el videojuego donde se implementó el aprendizaje automático, los datos relevantes, fueron:

La distancia que existe desde el Jugador al Enemigo en el videojuego, esta distancia será llamada DistJugEne, para hacer más simple la explicación. La forma de obtener esta distancia, será utilizando la distancia euclidiana. Para hacer este cálculo se implementa un método, en el cual recibirá por atributos, las coordenadas en el eje X e Y del Jugador y el Enemigo.

De esta misma manera también se obtendrán dos datos que serán relevantes para la creación de la descripción lingüística.

Los siguientes datos serán, la distancias entre el Jugador al Premio más cercano a él, la cual será llamada como DistJugPre y la distancias entre el Enemigo al Premio más cercano al Jugador, que será llamada DistEnePre.

Otro dato relevante será la protección que hay entre el Jugador y el Enemigo, esto se refiere a la cantidad de obstáculos que se encuentran en el área formada por sus coordenadas. Este dato será conocido como Protección.

Por último se obtendrá la dirección hacia donde se movió el Jugador en todo instante, Arriba, Abajo, Izquierda, Derecha.

La información numérica que será insertada en la Tabla de datos [7], es necesaria agruparla en rangos, los cuales serán clasificados en relación a la “distancia hacia a...” y el nivel de protección en el cual se encuentra el Jugador.

Para los valores numéricos de la DistJugEne, se utilizó el siguiente rango de clasificación:

Si la DistJugEne es un valor entre 0 y 3 se considera very_close.

Si la DistJugEne es un valor entre 4 y 8 se considera close.

Si la DistJugEne es un valor entre 9 y 18 se considera far.

Si la DistJugEne es un valor entre 19 y 25 se considera very_far.

Para los valores numéricos de la DistJugPre, se utilizó el siguiente rango de clasificación:

Si la DistJugPre es un valor entre 0 y 2 se considera very_close.

Si la DistJugPre es un valor entre 3 y 6 se considera close.

Si la DistJugPre es un valor entre 7 y 13 se considera far.

Si la DistJugPre es un valor entre 14 y 21 se considera very_far.

Para los valores numéricos de la DistEnePre, se utilizó el siguiente rango de clasificación:

Si la DistEnePre es un valor entre 0 y 3 se considera very_close.

Si la DistEnePre es un valor entre 3 y 8 se considera close.

Si la DistEnePre es un valor entre 9 y 18 se considera far.

Si la DistEnePre es un valor entre 19 y 25 se considera very_far.

Para los valores numéricos de la Proteccion, se utilizó el siguiente rango de clasificación:

Si la Proteccion es un valor entre 0 y 5 se considera very_low.

Si la Proteccion es un valor entre 6 y 20 se considera low.

Si la Proteccion es un valor entre 21 y 45 se considera high.

Si la Proteccion es un valor entre 46 y 100 se considera very_high.

En la siguiente parte se detallarán los constructores y métodos de las clases, más importantes en la ejecución de los modos Entrenamiento y Entrenado del Proyecto.

## 4.1 Descripción de las clases importantes

### 4.1.1 Clase Jugador

Es la clase encargada de gestionar el movimiento del Personaje Jugador. En esta clase se obtiene la posición en los ejes X e Y, cada vez que el Jugador se mueve por el escenario, utilizando estos datos se puede calcular las distancias que aparecerán en la Tabla de Datos. Recibe una referencia de la Tabla de Datos para poder entregar los datos de distancias. Gestiona el modo Jugador Entrenado, en el cual el Jugador deberá recorrer el escenario, por medio de la lectura de un archivo de Descripción Lingüística, que le proporcionan los movimientos a seguir.

# Detalles Constructor

**public Jugador(Enemigo Enemigo, GestionDistancias Distancia, Premios Premio, TablaDistancias Tabla, boolean TipoJugador)**

**Parámetros:**

Enemigo - Referencia a la Clase Enemigo.

Distancia - Referencia hacia la Clase GestionDistancias.

Premio - Referencia hacia la lista de premios(Posiciones).

Tabla - Referencia hacia la tabla de datos.

TipoJugador - Indica el valor si el jugador será entrenado o no. TRUE: Modo Entrenamiento FALSE: Modo Entrenado

# Detalles Métodos

**public void setDireccionAccion(int DireccionAccion)**

Entrega un valor numérico, que indica cual imagen deberá pintarse.

**Parámetros:** DireccionAccion - valor numérico.

**public Image getSpriteAccion()**

Devuelve la imagen del Jugador que debe pintarse en cada momento.

**Returns:** devuelve un dato tipo Image.

**public void keyPressed(KeyEvent ke)**

Captura el valor de una tecla presionada.

Specified by: keyPressed in interface KeyListener

**Parámetros:** ke - valor de la tecla presionada.

**public void leerTXT() throws java.io.FileNotFoundException, java.io.IOException**

Método que lee el archivo de descripción lingüística.

Throws: java.io.FileNotFoundException - Excepción de fichero.

java.io.IOException - Excepción de leer fichero.

**public void JugadorEntrenado()**

Método que mueve al Jugador Virtual, utilizando la descripción lingüística como base, para sus movimientos.

**public boolean MoverAbajo(String DireccionAnterior)**

Método que verifica si hay camino disponible en el escenario para que el Jugador pueda desplazarse.

**Parámetros:** DireccionAnterior - recibe el movimiento anterior que hizo el Jugador.

**Returns:** true si es que puede mover hacia Abajo, false si no es posible moverse.

**public boolean MoverArriba(String DireccionAnterior)**

Método que verifica si hay camino disponible en el escenario para que el Jugador pueda desplazarse.

**Parámetros**: DireccionAnterior - recibe el movimiento anterior que hizo el Jugador.

**Returns:** true si es que puede mover hacia Arriba, false si no es posible moverse.

**public boolean MoverDerecha(String DireccionAnterior)**

Método que verifica si hay camino disponible en el escenario para que el Jugador pueda desplazarse.

**Parámetros:** DireccionAnterior - recibe el movimiento anterior que hizo el Jugador.

**Returns:** true si es que puede mover hacia Derecha, false si no es posible moverse.

**public boolean MoverIzquierda(String DireccionAnterior)**

Método que verifica si hay camino disponible en el escenario para que el Jugador pueda desplazarse.

**Parámetros:** DireccionAnterior - recibe el movimiento anterior que hizo el Jugador.

**Returns:** true si es que puede mover hacia Izquierda, false si no es posible moverse.

**public void moverAutomatico()**

Método que mueve al Jugador en forma automática, haciendo una búsqueda por anchura de los premios en el escenario.

**public String Combinacion4(EstadosDistancias4 Combinacion, ArrayList<EstadosDistancias4> Lista, ArrayList<Movimientos> ListaMovimientos, String MoverAux)**

Método, que compara la combinación de parámetros DistJugEne, DistJugPre, DistEnePre, Proteccion, con alguna coincidencia en la descripción lingüística y utiliza la dirección correspondiente a esa combinación.

**Parámetros:**

Combinacion - Combinación de parámetros, que se van apareciendo en la Tabla a cada 400 milésimas de segundos.

Lista - que contiene la combinación generada en la descripción lingüística.

ListaMovimientos - Lista que además de contener la combinación generada en la descripción lingüística contiene la dirección que se hizo en el momento que se generó esa combinación.

MoverAux - parámetro que permite que se genere un nuevo movimiento, si es que hay más de una coincidencia en la descripción lingüística.

**Returns:** devuelve una dirección si encuentra combinación, si no lo encuentra devuelve un "2".

**public void keyTyped(KeyEvent ke)**

**Specified by:** keyTyped in interface java.awt.event.KeyListener

**public void keyReleased(java.awt.event.KeyEvent ke)**

**Specified by:**

keyReleased in interface java.awt.event.KeyListener

**public void Colision()**

Método que verifica cada 300 milésimas de segundo, si el Jugador y el Enemigo Colisionan.

**public double getPixelX()**

Devuelve el valor del Pixel en el Eje X del Jugador.

**Returns:** double.

**public double getPixelY()**

Devuelve el valor del Pixel en el Eje Y del Jugador.

**Returns:** double.

**public void setCuadro(JPanel Cuadro)**

**public int getX()**

Devuelve el valor en la Matriz, en el eje X donde se encuentra el Jugador.

**Returns:** devuelve un int..

**public int getY()**

Devuelve el valor en la Matriz, en el eje Y donde se encuentra el Jugador.

**Returns:** devuelve un int.

### 4.1.2 Clase ReglasJugador

Es la clase encargada de gestionar las reglas sobre el movimiento del Jugador. Es la clase que le proporciona al jugador la información sobre si puede moverse hacia una dirección, mientras haya un camino disponible.

## Detalles Constructor

**public ReglasJugador(ArrayList<Point> Premios, GestionDistancias Distancia, boolean TipoJugador)**

Parámetros:

Premios - Lista de los Premios.

Distancia - Referencia a la Clases GestionDistancias.

TipoJugador - referencia si es Jugador Humano o Jugador Virtual.

## Detalles Métodos

**public boolean getCaminoArriba(byte[][] Laberinto, int Y, int X)**

Reglas para que los personajes se puedan mover por el escenario, siempre que haya un camino disponible.

**Parámetros:**

Laberinto - Nivel del Escenario

Y - Posición del Jugador en el Eje Y

X - Posición del Jugador en el Eje X

**Returns:** True, si es posible moverse. False, si no es posible moverse.

**public boolean getCaminoAbajo(byte[][] Laberinto, int Y, int X)**

Reglas para que los personajes se puedan mover por el escenario, siempre que haya un camino disponible.

**Parámetros:**

Laberinto - Nivel del Escenario

Y - Posición del Jugador en el Eje Y

X - Posición del Jugador en el Eje X

**Returns:** True, si es posible moverse. False, si no es posible moverse.

**public boolean getCaminoIzquierda(byte[][] Laberinto, int Y, int X)**

Reglas para que los personajes se puedan mover por el escenario, siempre que haya un camino disponible.

**Parámetros:**

Laberinto - Nivel del Escenario

Y - Posición del Jugador en el Eje Y

X - Posición del Jugador en el Eje X

**Returns:** True, si es posible moverse. False, si no es posible moverse.

**public boolean getCaminoDerecha(byte[][] Laberinto, int Y, int X)**

Reglas para que los personajes se puedan mover por el escenario, siempre que haya un camino disponible.

**Parámetros:**

Laberinto - Nivel del Escenario

Y - Posición del Jugador en el Eje Y

X - Posición del Jugador en el Eje X

**Returns:** True, si es posible moverse. False, si no es posible moverse.

**public void EliminarPremio(int Y, int X)**

Método que Elimina los premios cuando el jugador pasar por la posición donde están pintados

**Parámetros:**

Y - Posición del Jugador en el eje Y

X - Posición del Jugador en el eje X

### 4.1.3 Clase Enemigo

Esta clase es la encargada de gestionar el movimiento del Enemigo en el escenario. De esta clase se puede obtener la posición del Enemigo en los ejes X e Y, para poder calcular las distancias que serán necesarias para poder obtener unas Descripción Lingüística.

## Detalles Constructor

**public Enemigo(GestionDistancias Distancia, TablaDistancias Tabla, boolean JugadorTipo)**

**Parámetros:**

Distancia - Referencia hacia la Clase GestionDistancias.

Tabla - Referencia hacia la Tabla.

JugadorTipo - Indica el valor si el jugador será entrenado o no. TRUE: Modo Entrenamiento. FALSE: Modo Entrenado

## Detalles Métodos.

**public void setDireccion(int DireccionAccion)**

Entrega un valor numérico, que indica cual imagen deberá pintarse.

**Parámetros:** DireccionAccion - valor numérico.

**public Image getSpriteAccion()**

Devuelve la imagen del Jugador que debe pintarse en cada momento.

**Returns:** devuelve un dato tipo Image.

**public void moverEnemigo()**

Método que mueve al Enemigo Cuando se ejecuta el Modo Entrenamiento.

**public void moverEnemigoAlterno()**

Método que mueve al Enemigo cuando se ejecuta el Modo Jugador Entrenado.

**public double getPixelX()**

**Returns:** Devuelve el valor del Pixel en el Eje X del Enemigo.

**public double getPixelY()**

**Returns:** Devuelve el valor del Pixel en el Eje Y del Enemigo.

**public void setCuadro(JPanel Cuadro)**

**public int getX()**

**Returns:** Devuelve el valor en la Matriz, en el eje X donde se encuentra el Enemigo.

**public int getY()**

**Returns:** Devuelve el valor en la Matriz, en el eje Y donde se encuentra el Jugador.

### 4.1.4 Clase ReglasEnemigo.

Es la clase encargada de gestionar las reglas sobre el movimiento del Enemigo. Es la clase que le proporciona al Enemigo la información sobre si puede moverse hacia una dirección, mientras haya un camino disponible. Además se encarga de detener al enemigo en el momento que el Jugador y el Enemigo colisionan en su recorrido.

## Detalles Constructor.

**public ReglasEnemigo(GestionDistancias distancia)**

Parámetros:

Distancia - Referencia a la Clases GestionDistancias.

## Detalles Métodos.

**public boolean getCaminoArribaEnemigo(byte[][] Laberinto, int Y, int X)**

Reglas para que los personajes se puedan mover por el escenario, siempre que haya un camino disponible.

**Parámetros:**

Laberinto - Nivel del Escenario

Y - Posición del Enemigo en el Eje Y

X - Posición del Enemigo en el Eje X

**Returns:** True, si es posible moverse. False, si no es posible moverse.

**public boolean getCaminoAbajoEnemigo(byte[][] Laberinto, int Y, int X)**

Reglas para que los personajes se puedan mover por el escenario, siempre que haya un camino disponible.

**Parámetros:**

Laberinto - Nivel del Escenario

Y - Posición del Enemigo en el Eje Y

X - Posición del Enemigo en el Eje X

**Returns:** True, si es posible moverse. False, si no es posible moverse.

**public boolean getCaminoIzquierdaEnemigo(byte[][] Laberinto, int Y, int X)**

Reglas para que los personajes se puedan mover por el escenario, siempre que haya un camino disponible.

**Parámetros:**

Laberinto - Nivel del Escenario

Y - Posición del Enemigo en el Eje Y

X - Posición del Enemigo en el Eje X

**Returns:** True, si es posible moverse. False, si no es posible moverse.

**public boolean getCaminoDerechaEnemigo(byte[][] Laberinto, int Y, int X)**

Reglas para que los personajes se puedan mover por el escenario, siempre que haya un camino disponible.

**Parámetros:**

Laberinto - Nivel del Escenario

Y - Posición del Enemigo en el Eje Y

X - Posición del Enemigo en el Eje X

**Returns:** True, si es posible moverse. False, si no es posible moverse.

**public void Perder(double X, double Y, double X1, double Y1)**

Método que ayuda a detener al jugador si hay colisión entre el Enemigo y el Jugador.

**Parámetros:**

X - Posición en Pixel en el eje X del Enemigo.

Y - Posición en Pixel en el eje Y del Enemigo.

X1 - Posición en Pixel en el eje X del Jugador.

Y1 - Posición en Pixel en el eje X del Jugador.

### 4.1.5 Clase TablaDistancias.

Es la clase encargada de la gestión de la Tabla de Datos, inicializa el nombre de las columnas y proporciona los métodos necesarios para obtener los valores de una celda en particular y entregar la cantidad de filas que se obtuvieron de una partida en el Juego.

## Detalles Constructor.

**public TablaDistancias()**

Inicializa el nombre de las columnas.

## Detalles Métodos.

**public int getCelda(int i, int j)**

Devuelve el valor de la celda en la Coordenada i,j.

**Parámetros:**

i - Valor de la coordenada eje Y

j - Valor de la coordenada eje X

**Returns:** Valor de la Celda i,j.

**public String getCeldaDireccion(int i, int j)**

Devuelve la dirección de la celda en la Coordenada i,j.

**Parámetros:**

i - Valor de la coordenada eje Y

j - Valor de la coordenada eje X

**Returns:** Valor de la Celda i,j.

**public void setFila(int Jug_Enem, int Prem_Jug, int Prem_Enem, int Murallas, String Direccion)**

**Parámetros:**

Jug_Enem - Distancia Jugador-Enemigo.

Prem_Jug - Distancia Jugador-Premio.

Prem_Enem - Distancia Enemigo-Premio.

Murallas - Nivel de Protección.

Direccion - Movimiento que hizo el Jugador.

**public int totalFilas()**

**Returns:** Devuelve la cantidad de Filas que tiene la Tabla.

### 4.1.6 Clase GestionArchivos

Es la clase encargada de generar archivos .txt, con información sobre la Tabla de Datos. Se encarga de generar un archivo que contiene una descripción lingüística sobre la partida realizada por un Jugador Humano. Además se encarga de llenar la tabla de datos cuando se utiliza el Modo Jugador Entrenado.

## Detalles Constructor.

**public GestionaArchivos(Jugador jugador, Enemigo enemigo, GestionDistancias distancia, TablaDistancias tabladistancias, boolean estado)**

**Parámetros:**

jugador - referencia a la clase Jugador.

enemigo - referencia a la clase Enemigo.

distancia - referencia a la clase GestionDistancias.

tabladistancias - referencia a la Tabla de Datos.

estado - boolean que dependiendo del valor que recibe, permite la ejecución de ciertas funciones.

## Detalles Métodos.

**public void CrearArchivos()**

Es la encargada de crear un archivo con la descripción lingüística del Jugador Humano, a partir de los datos que aparecen en la Tabla de Datos.

**public void tabla()**

Inserta datos a la tabla cada 400 milésimas de segundos, cuando está en funcionamiento el Jugador virtual.

**public void Archivos()**

Se encarga que se creen los archivos solo cuando el Jugador Humano desarrolla una partida.

### 4.1.7 Clase GestionDistancias

Es la clase encargada de hacer los cálculos, sobre las distancias DistJugEne, DistJugPre, DistEnePre y la Protección.

## Detalles Métodos

**public void setTipoJugador(boolean TipoJugador)**

Cuando se elige un Modo de Juego, se le entrega un valor boolean.

**Parámetros:**

TipoJugador - true, Modo Entrenar Jugador, false, Modo Jugador Entrenado.

**public double getPixelJugX()**

Devuelve el pixel en el eje X del Jugador.

**Returns:** double.

**public double getPixelJugY()**

Devuelve el pixel en el eje Y del Jugador.

**Returns:** double.

**public int getPremioX()**

**Returns:** Devuelve la posición en el Eje X del premio más cercano al Jugador.

**public int getPremioY()**

**Returns:** Devuelve la posición en el Eje X del premio más cercano al Jugador.

**public int getPosEneX()**

**Returns:** Devuelve la posición en el Eje X del Enemigo.

**public int getPosEneY()**

**Returns:** Devuelve la posición en el Eje X del Enemigo.

**public double getPixelEneX()**

**Returns:** Devuelve la posición del pixel en el Eje X del enemigo.

**public double getPixelEneY()**

**Returns:** Devuelve la posición del pixel en el Eje Y del enemigo.

**public void setPosEneX(int PosEneX, double PixelEneX)**

Recibe la Posición en la Matriz y en Pixel del Enemigo en el Eje X.

**Parámetros:**

PosEneX - posición en la matriz de Enemigo

PixelEneX - posición en pixel del Enemigo.

**public void setPosEneY(int PosEneY, double PixelEneY)**

Recibe la posición en la Matriz y en Pixel del Enemigo en el Eje Y.

**Parámetros:**

PosEneY - posición en la matriz de Enemigo.

PixelEneY - posición en pixel del Enemigo.

**public String getPremio()**

**Returns:** Devuelve el valor en formato coordenada del premio más cercano al Jugador.

**public int getPosJugX()**

**Returns:** Devuelve la posición del Jugador en el eje X.

**public void setPosJugX(int PosJugX, double PixelJugX)**

**Parámetros:**

PosJugX - Posición del Jugador en el eje X.

PixelJugX - Pixel donde se encuentra el Jugador en el eje X.

**public int getPosJugY()**

**Returns:** Devuelve la posición del Jugador en el eje Y.

**public void setPosJugY(int PosJugY, double PixelJugY)**

**Parámetros:**

PosJugY - Posición del Jugador en el eje Y.

PixelJugY - Pixel donde se encuentra el Jugador en el eje Y.

**public boolean isSeMueve()**

Método para saber si el oponente se está moviendo aún

**Returns:** true, aun se mueve. false, no se mueve.

**public void setSeMueve(boolean seMueve)**

Método para hacer que el oponente deje de moverse.

**Parámetros:**

seMueve - true, puede moverse. false, no puede moverse.

**public boolean isMeta()**

Método para saber si el jugador llegó a la Meta.

**Returns:** true, Jugador, llegó a la meta. false, Jugador aún no llega a la meta.

**public void setMeta(boolean Meta)**

Método que cambia el valor, para saber que el jugador ya llegó a la meta.

**Parámetros:**

Meta - true, Jugador, llegó a la meta. false, Jugador aún no llega a la meta.

**public void setListaPremios(ArrayList<Point> Lista)**

Recibe una lista de las posiciones de los premios en la Matriz para realizar cálculo de las distancias a cada premio.

**Parámetros:**

Lista - Lista de posiciones de los premios.

**public void setDistJugEne(int X, int Y, int X1, int Y1)**

Método que calcula la Distancia entre el Jugador y el Oponente.

**Parámetros:**

X - Posición del Jugador en el Eje X.

Y - Posición del Jugador en el Eje Y.

X1 - Posición del Enemigo en el Eje X.

Y1 - Posición del Enemigo en el Eje Y.

**public void setMurallas(int Y, int X, int Y1, int X1)**

Método que calcula la cantidad de obstáculos entre el Jugador y el Oponente.

**Parámetros:**

Y - Posición del Jugador en el Eje Y.

X - Posición del Jugador en el Eje X.

Y1 - Posición del Enemigo en el Eje Y.

X1 - Posición del Enemigo en el Eje X.

**public void setDistPre(int Y, int X, int Y1, int X1)**

Método que calcula la Distancia entre el Jugador y el premio más cercano, la Distancia entre el Oponente y el premio más cercano al Jugador. X obtiene las coordenadas del premio cercano el jugador.

**Parámetros:**

Y - Posición del Jugador en el Eje Y.

X - Posición del Jugador en el Eje X.

Y1 - Posición del Enemigo en el Eje Y.

X1 - Posición del Enemigo en el Eje X.

**public void PremioSeguro(ArrayList<PremiosDistancias> Lista)**

Método que hace huir a Jugador Virtual cuando el enemigo se encuentra cerca.

**Parámetros:**

Lista - Recibe la lista de Premios(Posiciones), que se encuentran en el escenario.

**public void setDireccion(String Direccion)**

**Parámetros:**

Direccion - Direccion hacia donde se Mueve el Jugador.

**public int getDistJugEne()**

**Returns:** Devuelve la distancia del Jugador al Enemigo.

**public int getDistJugPre()**

**Returns:** Devuelve la distancia del Jugador al premio más cercano.

**public int getDistEnePre()**

**Returns:** Devuelve la distancia del Enemigo al premio más cercano al Jugador.

**public int getMurallas()**

**Returns:** Devuelve la cantidad de obstáculos que se encuentra entre el área formada, por las posiciones del Jugador y el Enemigo.

**public String getDireccion()**

**Returns:** Devuelve la dirección hacia donde se movió el Jugador.

## 4.2 Diagrama de clases

En el siguiente diagrama se representan las clases en directa relación, con las funciones de Entrenar Jugador, dónde se obtiene la información para el aprendizaje y la función Jugador Entrenado, dónde se obtiene un jugador virtual.

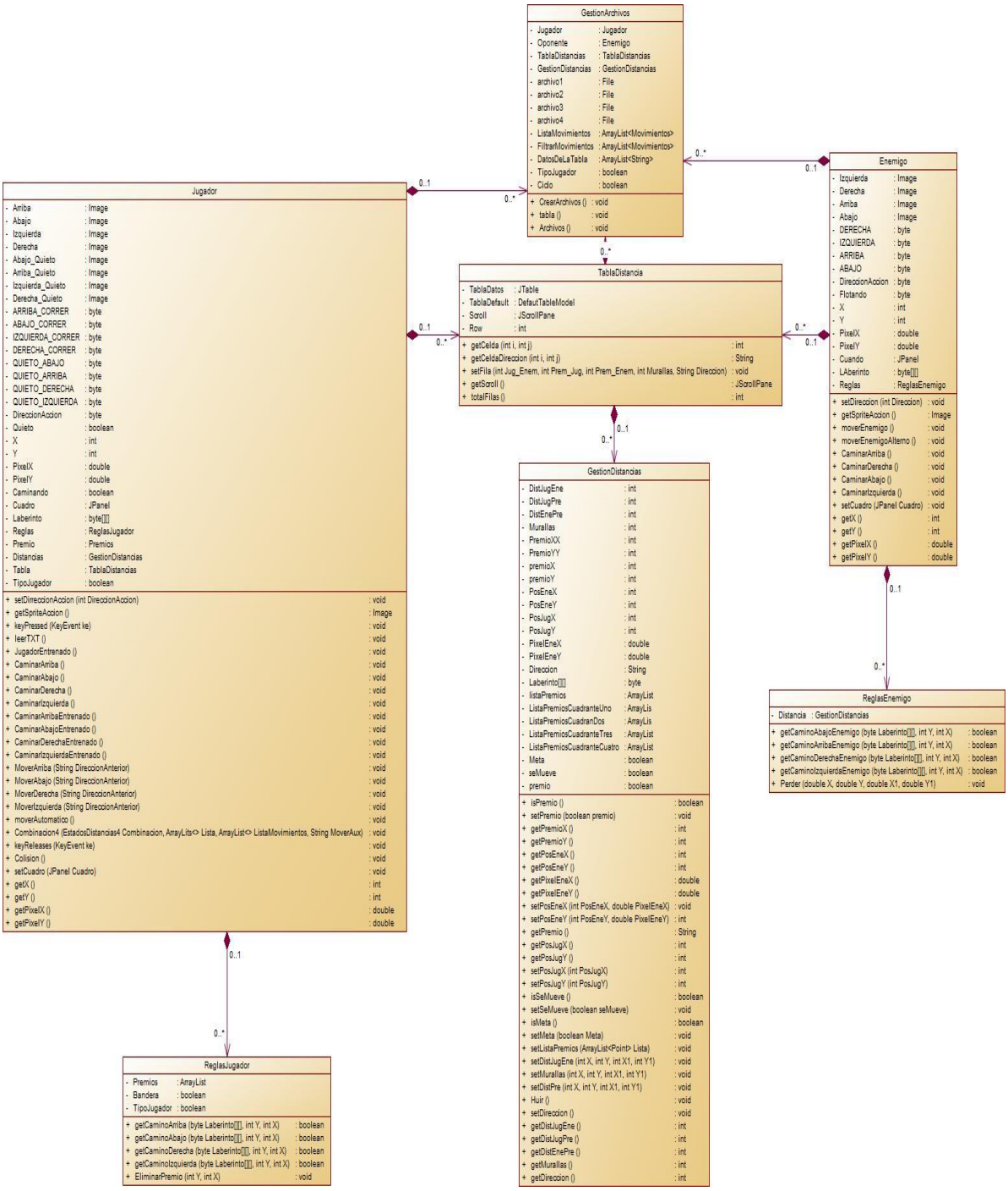


**Figura 4 Relación completa de las clases más importantes en el entrenamiento del Jugador Virtual.**

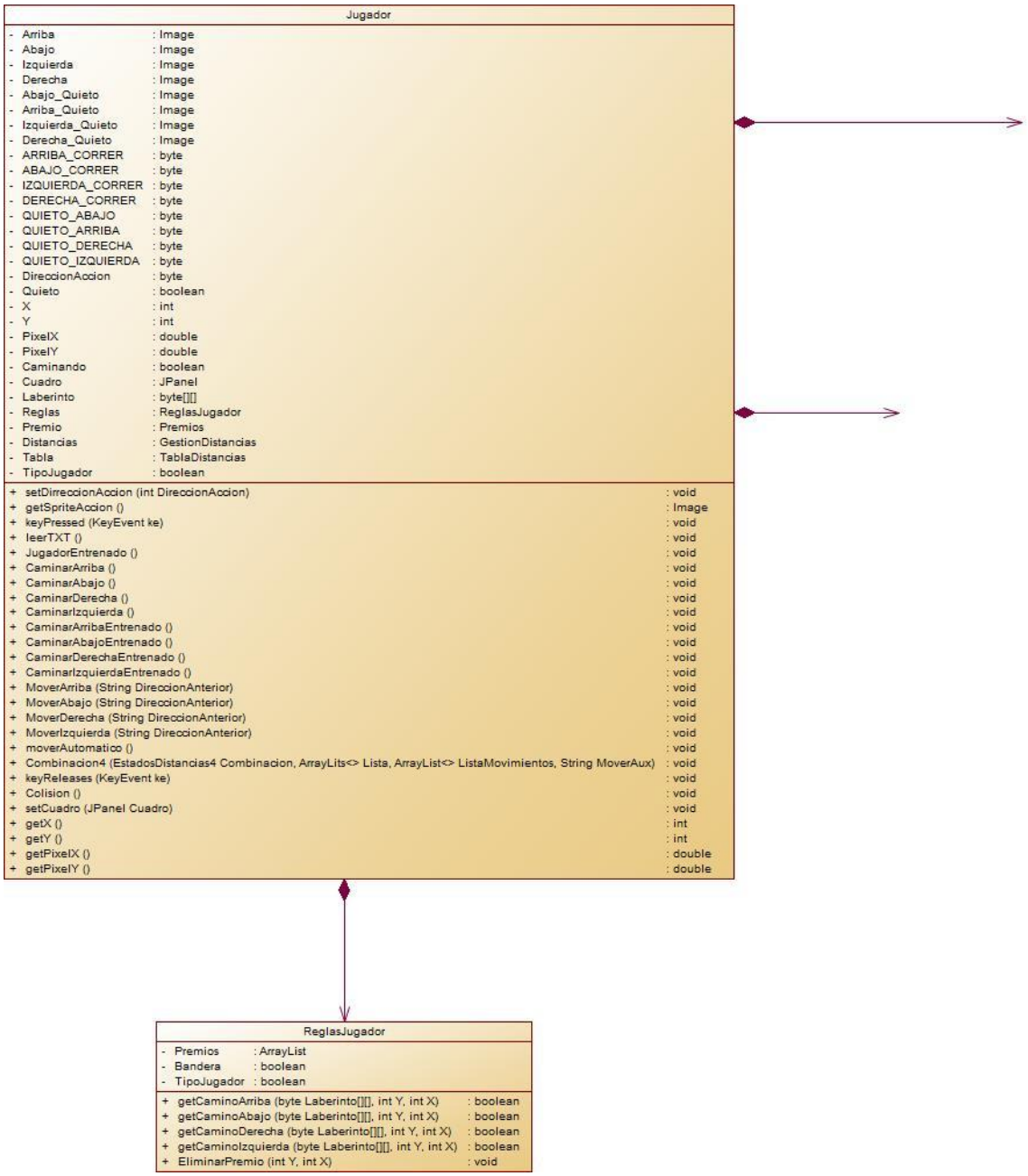


**Figura 5 Relación de las clases más importantes, parte 1.**

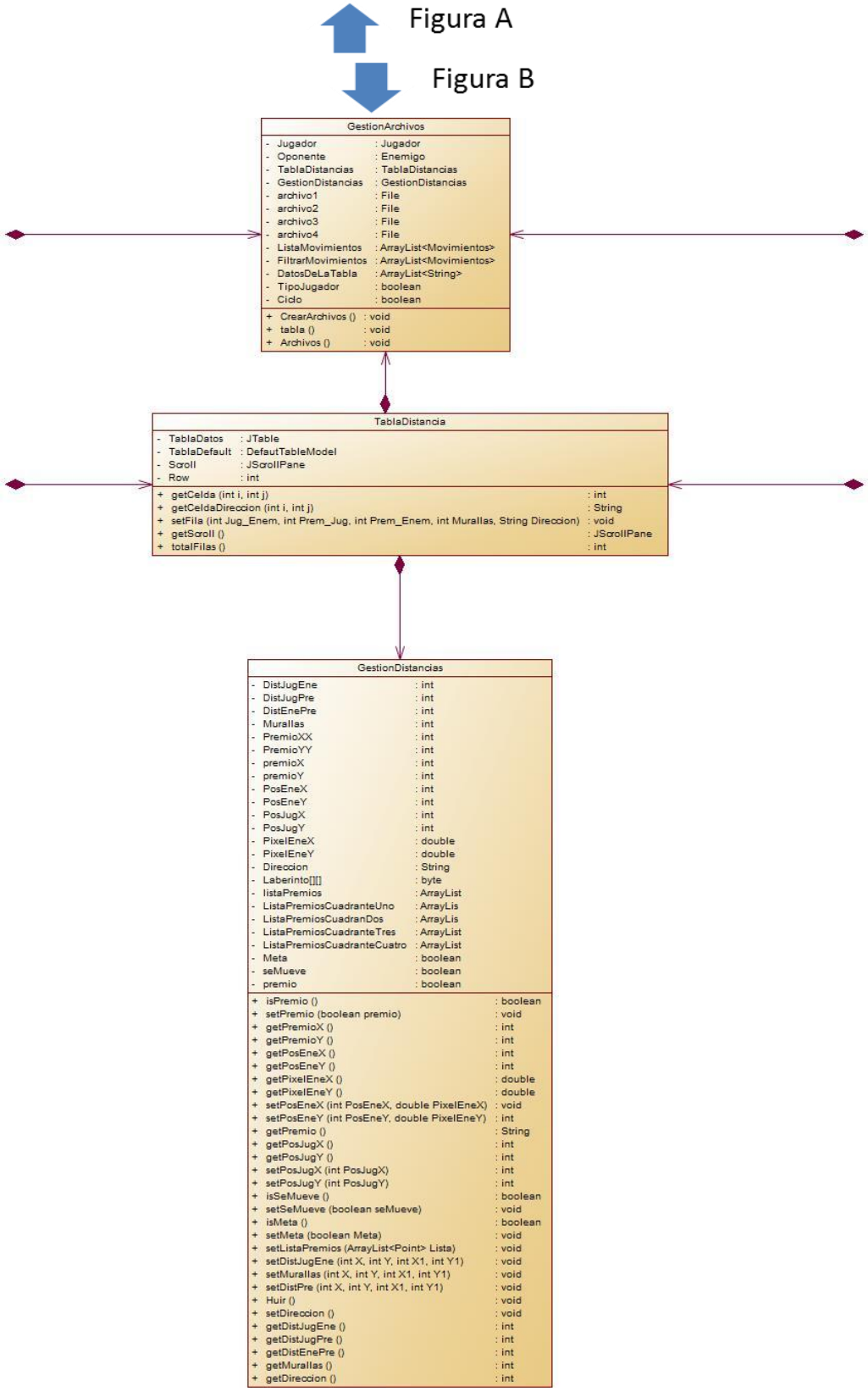


**Figura 6 Relación de las clases más importantes, parte 2.**

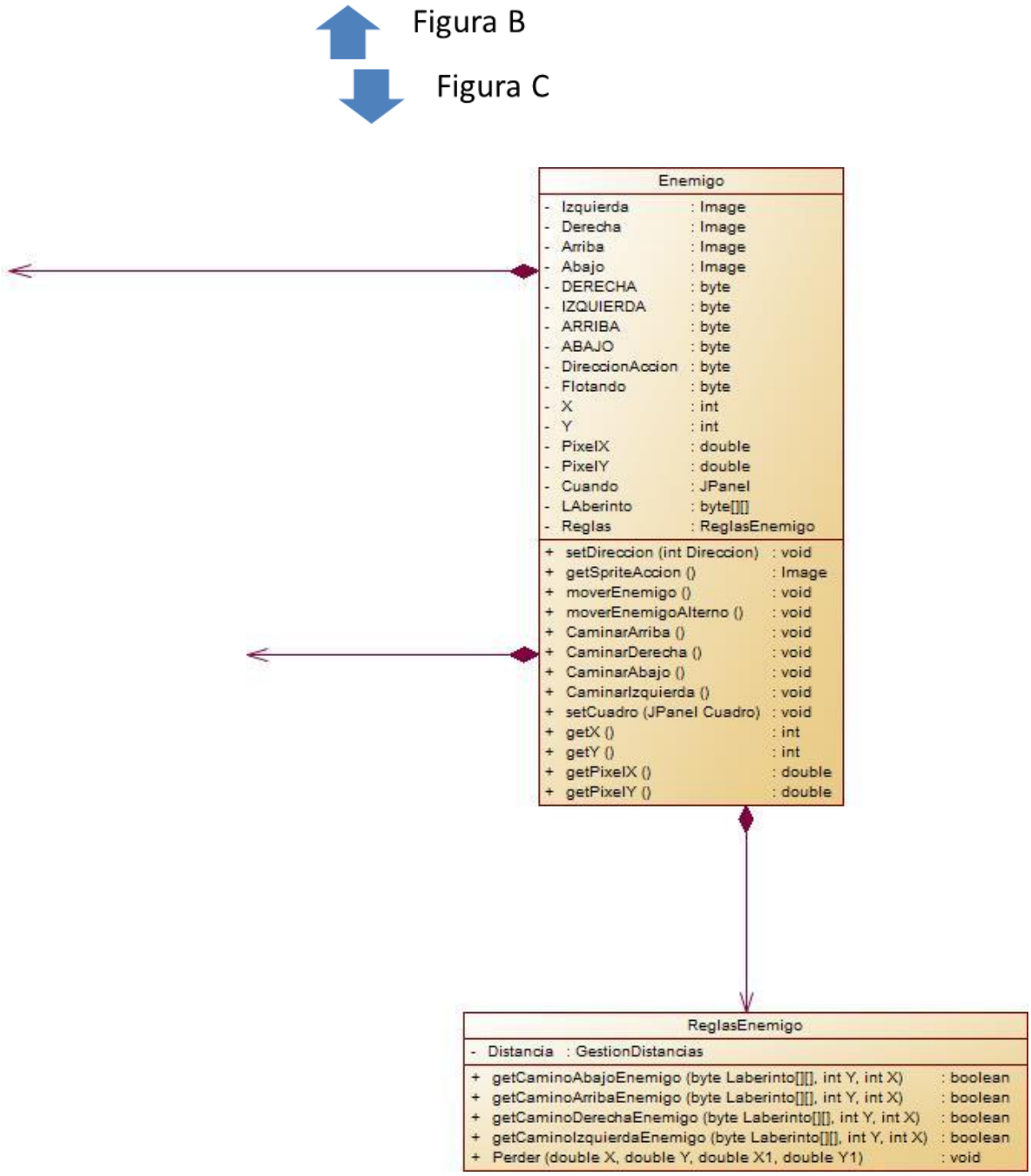


**Figura 7 Relación de las clases más importantes, parte 3.**

# 5 Pruebas

Para comprobar la efectividad del uso de la Descripción Lingüística de Fenómeno Complejos, se realizaron una serie de pruebas, en la cuales, se le pidió a personas que jugaran unas cuantas partidas para poder obtener datos que reflejan la forma en que se desempeñan jugando, estos datos son registrados en archivos .txt. Ya con los suficientes archivos, se procedió a realizar la simulación de un Jugador Virtual.

## 5.1 Datos de Prueba

Para poder realizar las pruebas es necesario contar con la información, que se transformara en un modelo de Descripción Lingüística. Para esto hay que tener claro cómo se obtiene esta información, a continuación se ilustrará con imágenes la secuencia para obtener los datos de pruebas.

Los datos se obtienen al realizar una partida en el juego.

En la pantalla aparece el escenario del juego, un Laberinto simple, donde el objetivo es capturar los premios que se encuentran por el escenario. Al lado derecho aparecerá una Tabla de Datos, que irá mostrando las distancias y el nivel de protección, aparecerá una fila con los valores de las distancias y la protección sólo cuando el jugador haga un movimiento.


Figura 4 Pantalla al iniciar el Entrenamiento.


Figura 5 Ejemplo de datos insertándose en la Tabla.

El Jugador se desplaza una posición hacia abajo se insertará una fila con la combinación de datos.


Figura 6 Ejemplo Enemigo moviéndose sin insertar datos en la Tabla.

Si el Jugador no se mueve, pero si lo hace el Enemigo, no se insertará ninguna otra combinación de datos en la Tabla.

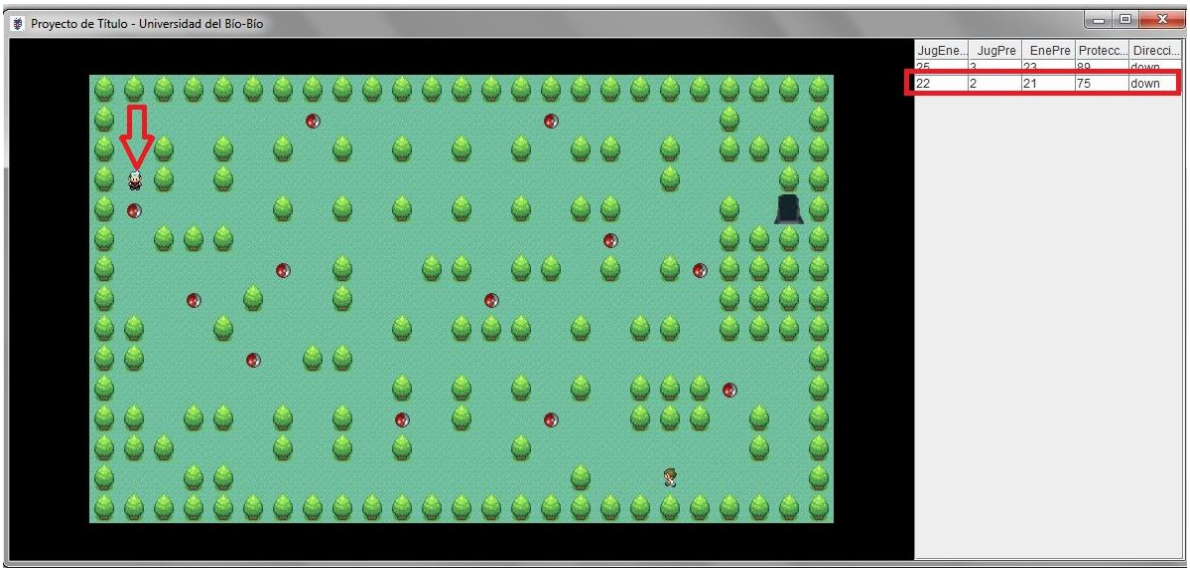


**Figura 7 Ejemplo Jugador moviéndose e insertando datos en la Tabla.**

Cuando el Jugador vuelva a realizar un movimiento se insertará una fila con las combinaciones correspondientes.

Una vez quedado claro cómo funciona la inserción de datos en la tabla, continuaremos con la explicación de los datos de pruebas.

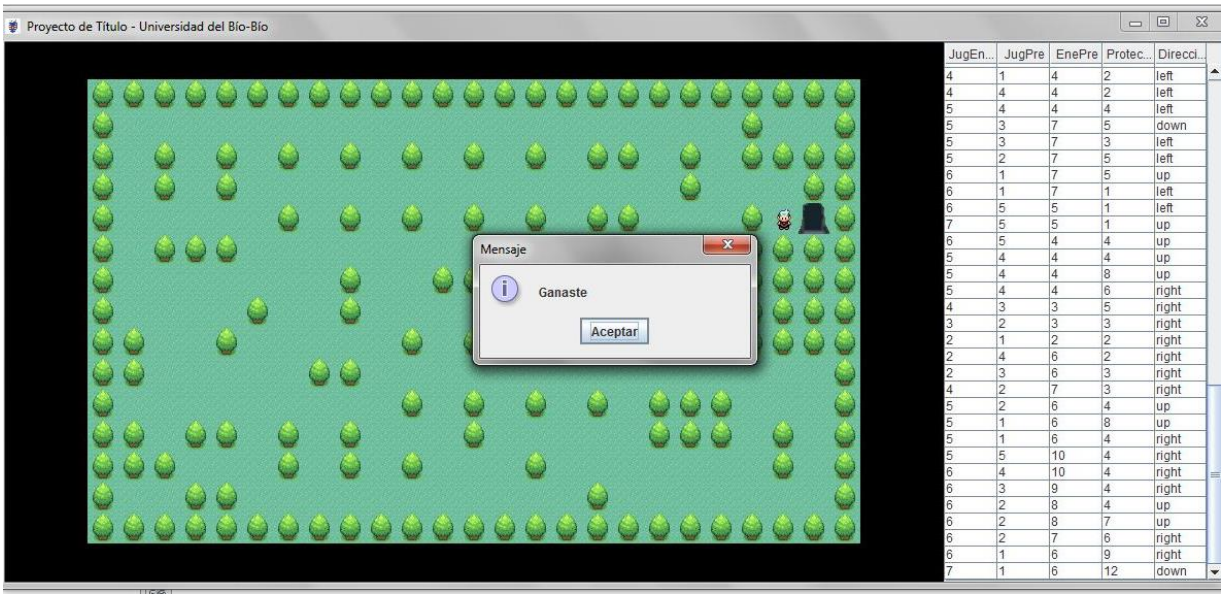


**Figura 8 Partida de Entrenamiento finalizada.**

Cuando la partida finaliza de forma correcta, esto quiere decir, que el Jugador gane, la información que se encuentra en la Tabla de datos será traspasada a unos archivos txt.

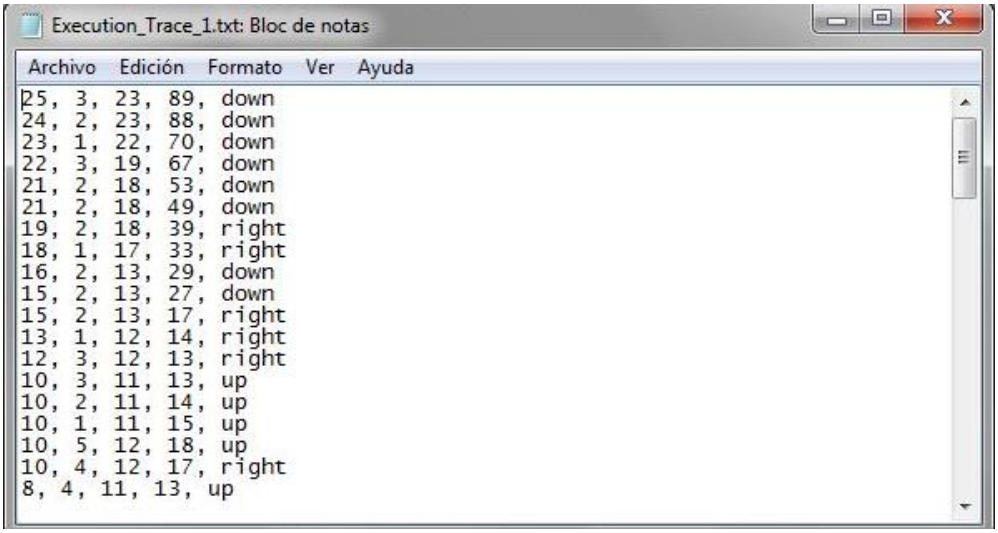


**Figura 9 Archivo txt, datos de la Tabla.**

Estos datos son procesados, para transformarlos en un modelo de descripción lingüística en el cual los datos son filtrados para evitar repeticiones de datos. Este es el archivo con el cual se procede a crear un Jugador Virtual, que deberá intentar capturar todos los premios según los movimientos del Jugador Humano en algunas situaciones.

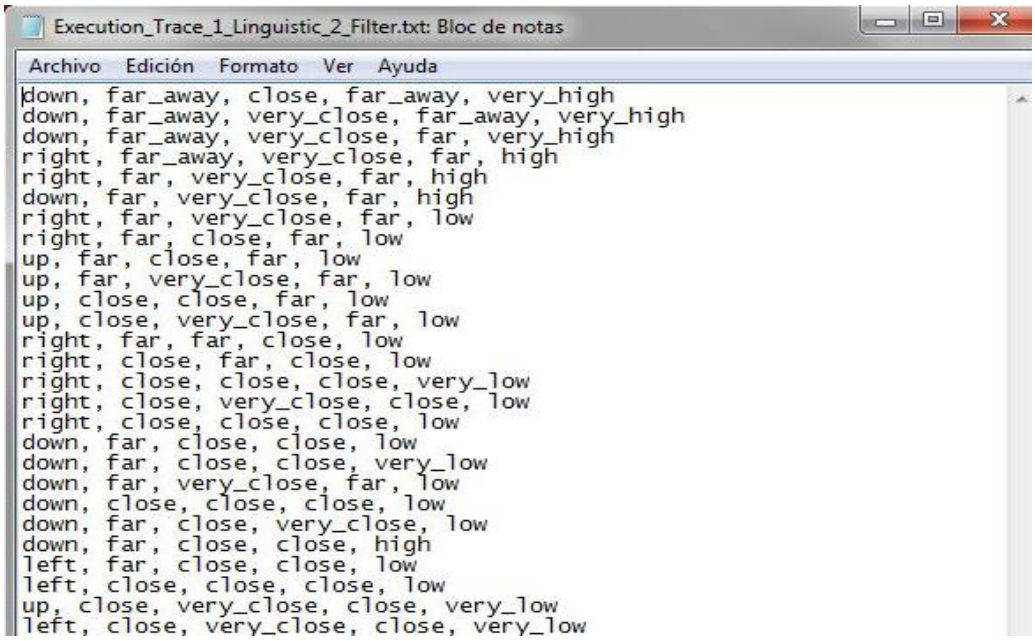


**Figura 10 Archivo txt, Descripción Lingüística.**

## 5.2 Traza de Ejecución de una partida

| Dist JugEne | Dist JugPre | Dist EnePre | Protección | Dirección |
|---|---|---|---|---|
| 25 | 3 | 23 | 89 | down |
| 24 | 2 | 23 | 88 | down |
| 23 | 1 | 22 | 70 | down |
| 22 | 3 | 19 | 67 | down |
| 21 | 2 | 18 | 53 | down |
| 21 | 2 | 18 | 49 | down |
| 19 | 2 | 18 | 39 | right |
| 18 | 1 | 17 | 33 | right |
| 16 | 2 | 13 | 29 | down |
| 15 | 2 | 13 | 27 | down |
| 15 | 2 | 13 | 17 | right |
| 13 | 1 | 12 | 14 | right |
| 12 | 3 | 12 | 13 | right |
| 10 | 3 | 11 | 13 | up |
| 10 | 2 | 11 | 14 | up |
| 10 | 1 | 11 | 15 | up |
| 10 | 5 | 12 | 18 | up |
| 10 | 4 | 12 | 18 | right |
| 8 | 4 | 11 | 13 | up |
| 8 | 3 | 10 | 14 | up |
| 8 | 2 | 10 | 12 | up |
| 9 | 1 | 10 | 15 | up |
| 9 | 8 | 8 | 14 | right |
| 8 | 7 | 8 | 9 | right |
| 8 | 6 | 8 | 4 | right |
| 7 | 5 | 8 | 4 | right |
| 7 | 4 | 8 | 1 | right |
| 6 | 3 | 7 | 4 | right |
| 6 | 2 | 7 | 8 | right |

| | | | | |
|---|---|---|---|---|
| 7 | 1 | 7 | 9 | right |
| 7 | 4 | 7 | 10 | right |
| 7 | 4 | 7 | 12 | right |
| 8 | 4 | 7 | 10 | right |
| 8 | 4 | 6 | 8 | right |
| 8 | 4 | 6 | 10 | right |
| 9 | 5 | 6 | 10 | down |
| 9 | 4 | 6 | 6 | down |
| 9 | 3 | 6 | 1 | down |
| 9 | 2 | 9 | 10 | down |
| 9 | 1 | 9 | 10 | down |
| 9 | 3 | 6 | 11 | down |
| 8 | 3 | 5 | 10 | down |
| 9 | 2 | 11 | 16 | down |
| 9 | 1 | 10 | 13 | right |
| 9 | 1 | 9 | 17 | down |
| 9 | 6 | 2 | 18 | down |
| 10 | 6 | 8 | 21 | down |
| 10 | 6 | 7 | 20 | left |
| 8 | 5 | 6 | 13 | left |
| 8 | 4 | 6 | 11 | left |
| 7 | 3 | 6 | 6 | left |
| 6 | 2 | 5 | 3 | up |
| 4 | 2 | 4 | 2 | left |
| 4 | 1 | 4 | 2 | left |
| 4 | 4 | 4 | 2 | left |
| 5 | 4 | 4 | 4 | left |
| 5 | 3 | 7 | 5 | down |
| 5 | 3 | 7 | 3 | left |
| 5 | 2 | 7 | 5 | left |
| 6 | 1 | 7 | 5 | up |
| 6 | 1 | 7 | 1 | left |
| 6 | 5 | 5 | 1 | left |
| 7 | 5 | 5 | 1 | up |

| | | | | |
|---|---|---|---|---|
| 6 | 5 | 4 | 4 | up |
| 5 | 4 | 4 | 4 | up |
| 5 | 4 | 4 | 8 | up |
| 5 | 4 | 4 | 6 | right |
| 4 | 3 | 3 | 5 | right |
| 3 | 2 | 3 | 3 | right |
| 2 | 1 | 2 | 2 | right |
| 2 | 4 | 6 | 2 | right |
| 2 | 3 | 6 | 3 | right |
| 4 | 2 | 7 | 3 | right |
| 5 | 2 | 6 | 4 | up |
| 5 | 1 | 6 | 8 | up |
| 5 | 1 | 6 | 4 | right |
| 5 | 5 | 10 | 4 | right |
| 6 | 4 | 10 | 5 | right |
| 6 | 3 | 9 | 4 | right |
| 6 | 2 | 8 | 4 | up |
| 6 | 2 | 8 | 7 | up |
| 6 | 2 | 7 | 6 | right |
| 6 | 1 | 6 | 9 | right |

Tabla 1 Traza de Ejecución de una partida.

## 5.3 Modelo de Descripción Lingüística

| Dist JugEne | Dist JugPre | Dist EnePre | Protección | Dirección |
|---|---|---|---|---|
| far_away | close | far_away | very_high | down |
| far_away | very_close | far_away | very_high | down |
| far_away | very_close | far | very_high | down |
| far_away | very_close | far | high | right |
| far | very_close | far | high | right |
| far | very_close | far | high | down |
| far | very_close | far | low | right |
| far | close | far | low | right |
| far | close | far | low | up |
| far | very_close | far | low | up |
| close | close | far | low | up |
| close | very_close | far | low | up |
| far | far | close | low | right |
| close | far | close | low | right |
| close | close | close | very_low | right |
| close | very_close | close | low | right |
| close | close | close | low | right |
| far | close | close | low | down |
| far | close | close | very_low | down |
| far | very_close | far | low | down |
| close | close | close | low | down |
| far | close | very_close | low | down |
| far | close | close | high | down |
| far | close | close | low | left |
| close | close | close | low | left |
| close | very_close | close | very_low | up |
| close | very_close | close | very_low | left |
| close | close | close | very_low | left |
| close | close | close | very_low | down |
| close | close | close | very_low | up |

| close | close | close | low | up |
|---|---|---|---|---|
| very_close | very_close | close | very_low | right |
| very_close | very_close | very_close | very_low | right |
| very_close | close | close | very_low | right |
| close | very_close | close | very_low | right |
| close | very_close | close | low | up |
| close | close | far | very_low | right |
| close | very_close | close | low | down |

Tabla 2 Modelo de Descripción Lingüística.

## 5.4 Estadísticas de simulaciones

Se realizaron pruebas para la simulación de un Jugador Virtual. Para obtener las pruebas, se le pidió a tres personas que realizaran tres partidas, por cada partida se realizaron 10 pruebas de simulación.

### Simulación Partida 1

| | Éxito | Fracaso |
|---|---|---|
| **Jugador 1** | **5** | **5** |
| **Jugador 2** | **7** | **3** |
| **Jugador 3** | **6** | **4** |

Tabla 3 Resultados Simulación 1.

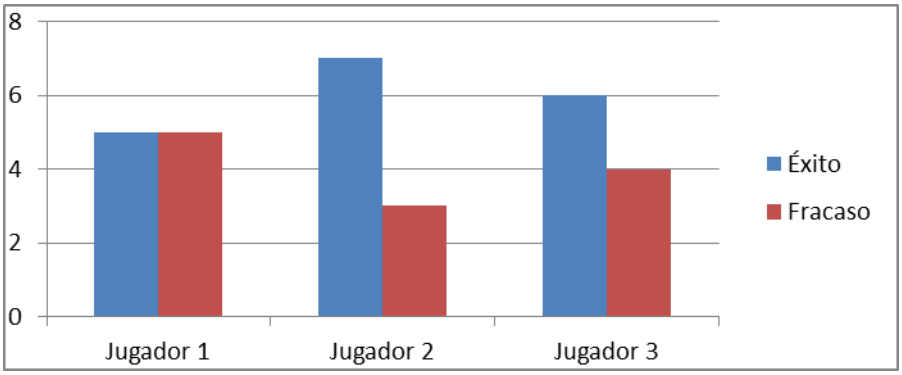


Figura 11 Grafico Resultados Simulación 1.

## Simulación Partida 2

| | Éxito | Fracaso |
|---|---|---|
| **Jugador 1** | **4** | **6** |
| **Jugador 2** | **7** | **3** |
| **Jugador 3** | **8** | **2** |

Tabla 4 Resultados Simulación 2.

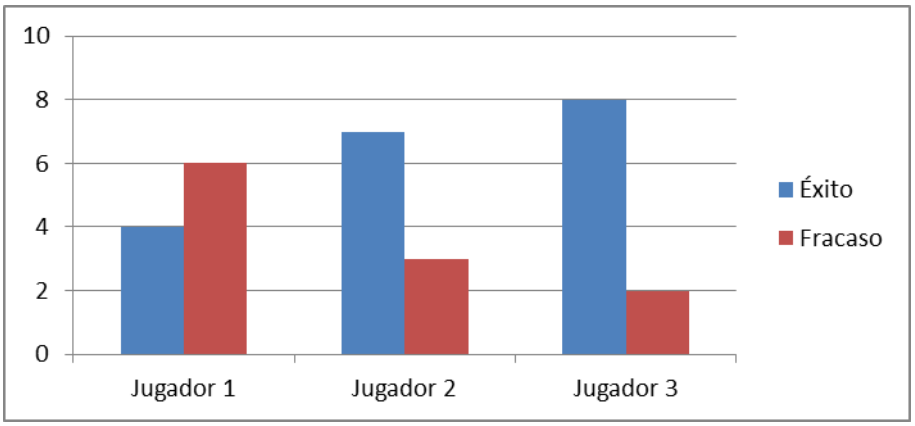


Figura 12 Grafico Resultados Simulación 2.

## Simulación Prueba 3

| | Éxito | Fracaso |
|---|---|---|
| **Jugador 1** | **6** | **4** |
| **Jugador 2** | **5** | **5** |
| **Jugador 3** | **3** | **7** |

Tabla 5Resultados Simulación 3.

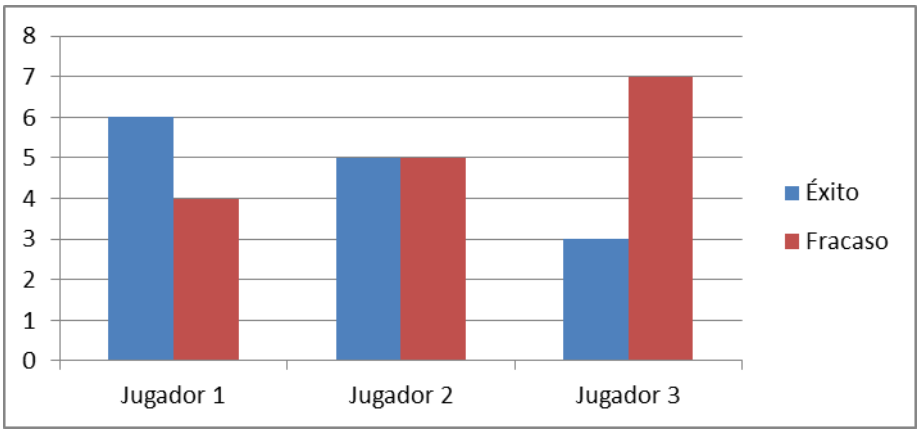


Figura 13 Grafico Resultados Simulación 3.

Total pruebas realizadas noventa.

| | Éxito | Fracaso |
|---|---|---|
| **Pruebas** | **51** | **39** |

Tabla 6 Resultados Pruebas

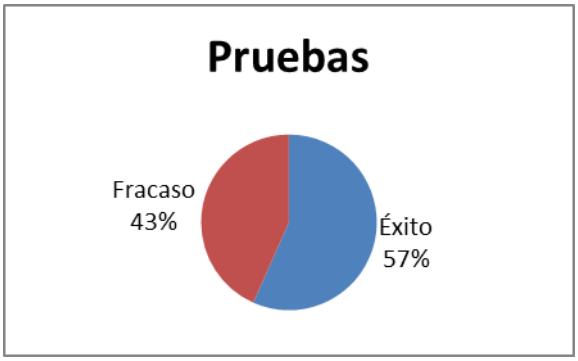


Figura 14 Grafico Resultado Pruebas.

Se obtuvo un 57% de éxito en el total de pruebas realizadas.

**Video de demostración funcionamiento de Modo Entrenar Jugador y Modo Jugador Entrenado:** https://www.youtube.com/watch?v=fgMaKhaVcO8

# 6 Conclusiones del Proyecto

Existen variadas técnicas para entrenar personajes y lograr que puedan moverse de forma automática, aprendiendo de sí mismos.

En este proyecto se estudió e implementó la Descripción Lingüística de Fenómenos Complejos en el entrenamiento de un Jugador Virtual, se profundizó en la rama de la Inteligencia Artificial, el Aprendizaje Automático, para lograr comprender como funcionan las técnicas de entrenamientos en las computadoras, y lograr entrenar a un NPC.

Se hizo una investigación sobre las variadas técnicas que se pueden aplicar para implementar el Aprendizaje Automático en los videojuegos, la cual se enfocó en explicar las técnicas de Árboles de Decisión, Algoritmos Genéticos y La Descripción Lingüística de Fenómenos Complejos, esta última es la que se utilizó para el entrenamiento automático de un Jugador Virtual.

Se realizó una revisión de trabajos que tenían relación que este proyecto, en los cuales se hizo un resumen, del trabajo desarrollado y una descripción breve de los métodos aplicados y los resultados obtenidos. Estos trabajos tratan sobre como automatizar los videojuegos de dos consolas, la Nintendo y la Atari 2600.

El proyecto alcanzó su objetivo general, el diseñar e implementar un Jugador virtual, el cual pudiera aprender por sí mismo a jugar un juego de computadora 2D, esto se logró gracias a la utilización de las Descripción Lingüística de Fenómenos Complejos, donde se lograr crear un modelo que contiene los movimientos, a partir de una partida realizada por un Jugador Humano, este modelo es filtrado evitando repeticiones de movimientos. El Jugador Virtual utiliza los movimientos disponibles en el modelo de Descripción Lingüística, para poder desplazarse por el escenario del videojuego. A pesar de esto, en la pruebas realizadas queda demostrado que la implementación no es 100% efectiva, ya que en algunas pruebas no logra cumplir el objetivo del videojuego. Este resultado puede deberse a que en el modelo de Descripción Lingüística creado, solo se incluyen los movimientos de un Jugador Humano, en una sola partida del videojuego.

Para lograr una mayor tasa de éxito en el entrenamiento se propone, desarrollar un modelo de Descripción Lingüística más completo, que no sólo contenga información de

una partida realizada por un Jugador Humano, si no que se haga un modelo de varias partidas, de esta forma se lograría capturar más información del comportamiento del Jugador Humano y el entrenamiento al Jugador Virtual podría volverse más eficiente. Se podría incluir en una continuación y/o nueva investigación, indagar qué nuevos datos serían relevantes para considerarlos en un modelo de Descripción Lingüística.

# 7 Referencias

[1] Machine Learning for Computer Games, 2005.

http://research.microsoft.com/en-us/collaboration/papers/machinelearningforcomputergames-gdc2005.pdf

[2] Mitchell. (1197). *Machine Learning.* McGraw Hill. Decision Tree Learning. pp. 52-78.

[3] Mitchell. (1197). *Machine Learning.* McGraw Hill. Gentic Algorithms. pp. 249-20.

[4] Murphy, T. (2013). *The First Level of Super Mario Bros.*

[5]Volodymyr Mnih, K. K. (2013). *Playing Atari with Deep Reinforcement Learning.*

[6]Linguistic Description of Complex Phenomena.
http://www.softcomputing.es/metaspace/portal/13/388-home?pms=1,306,388002,view,normal,0

[7] JTable.
http://static1.1.sqspcdn.com/static/f/923743/14411722/1317390535280/jtable_1.pdf?token=krKclqPzMmH0S70AwEv8dVKU%2FnQ%3D