Universidad Tecnológica de Panamá

Facultad de Ingeniería de Sistemas Computacionales

Departamento de Simulación de Sistemas

Proyecto N°2

Modalidad del Trabajo (Teórico / Teórico Práctico)

Integrantes

Castro Brenda, Colucci Gustavo, Quevedo David, Ramírez Dahian, Rodríguez Fernando, Vásquez Miguel

Asesor

Profesor Nicholas Béliz Osorio

2020

Universidad Tecnológica de Panamá

Facultad de Ingeniería de Sistemas Computacionales

Estrategias de solución de juegos con técnicas Minimax y Alfa-Beta

Asesor

Profesor Nicholas Béliz Osorio

Integrantes

Castro Brenda, Colucci Gustavo, Quevedo David, Ramírez Dahian, Rodríguez Fernando, Vásquez Miguel

Trabajo escrito del proyecto n°2

2020

**Abstract**

Los algoritmos de búsqueda en problemas de juegos tienen una gran importancia en el campo de la inteligencia artificial, ya que nos permiten, mediante teoría de juegos, realizar juegos en los que una IA juegue contra una persona y que esta pueda elegir movimientos efectivos para poder derrotar al oponente. Mediante la estructura árbol, Podemos representar los posibles movimientos que puede realizar la IA para contrarrestar un movimiento realizado por el oponente. En este Proyecto buscaremos hacer lo dicho anteriormente en el juego de estrategias Worms.

Lo que se busca en este trabajo es implementar lo aprendido durante este semestre y ver su aplicación en el campo de los videojuegos, esto, mediante las técnicas de Minimax y Alfa Beta, técnicas propuestas para aplicar la definición de la teoría de juegos, en este caso con Inteligencia Artificial.

En la implementación del algoritmo, usaremos una cantidad de 1 jugador por equipo con 1 gusano a su disposición de cada uno, así como las respectivas armas. Decidimos aplicar una interfaz gráfica sencilla, ya que hoy en día no contamos con el aprendizaje necesario para una realización de un videojuego en su parte gráfica. Es por eso por lo que preferimos utilizar el lenguaje de programación orientado a objetos Java, ya que este ofrece un modo gráfico bastante fácil de utilizar.

Palabras claves: Alfa-Beta, Búsqueda, Estrategia, Heurística, Inteligencia Artificial, Minimax, Worms.

**Tabla de Contenidos**

[**Capítulo I. Introducción** 1](#_Toc44685396)

[**Capítulo II: El Problema** 2](#_Toc44685397)

[**Capítulo III: Marco Teórico** 14](#_Toc44685398)

[**Capítulo IV: Marco Metodológico** 2](#_Toc44685399)

[**Diseño de la investigación** 2](#_Toc44685400)

[**Definición de investigación** 2](#_Toc44685401)

[**Tipo de investigación** 2](#_Toc44685402)

[**Tipos de diseño** 4](#_Toc44685403)

[**Selección de la muestra** 5](#_Toc44685404)

[**Población** 5](#_Toc44685405)

[**Muestra** 6](#_Toc44685406)

[**Instrumentos de medición** 7](#_Toc44685407)

[**Capítulo V: Desarrollo de las Aplicaciones** 2](#_Toc44685408)

[**Análisis del Problema** 2](#_Toc44685409)

[**Diseño** 4](#_Toc44685410)

[**Implementación en el lenguaje seleccionado** 12](#_Toc44685411)

[**Pruebas** 14](#_Toc44685412)

[**Capítulo VI: Presentación y análisis de resultados** 2](#_Toc44685413)

[**Recolección y medición de datos** 2](#_Toc44685414)

[**Análisis de los datos (Modelo matemático)** 2](#_Toc44685415)

[**Análisis de estadística descriptiva** 4](#_Toc44685416)

[**Análisis de la confiabilidad de los resultados** 5](#_Toc44685417)

[**Conclusiones** 1](#_Toc44685418)

[**Referencias** 4](#_Toc44685419)

[**Anexos** 8](#_Toc44685420)

[**Pruebas realizadas al programa** 9](#_Toc44685421)

[**Guía de instalación de NetBeans** 13](#_Toc44685422)

**Índice de Figuras**

[Figura 3.1. Worms. 16](#_Toc44685036)

[Figura 3.2. Representación de una pila. 17](#_Toc44685037)

[Figura 3.3. Cola lineal. 18](#_Toc44685038)

[Figura 3.4. Cola Circular 19](#_Toc44685039)

[Figura 3.5. Cola de prioridad 20](#_Toc44685040)

[Figura 3.6. Árboles 22](#_Toc44685041)

[Figura 3.7. Muestra que es un nodo padre y un nodo hijo 23](#_Toc44685042)

[Figura 3.8. Representación de grafos dirigidos. 25](#_Toc44685043)

[Figura 3.9. Grafo no dirigido 26](#_Toc44685044)

[Figura 3.10. Matriz de adyacencia de un grafo (sin pesos). 29](#_Toc44685045)

[Figura 3.11. Matriz de adyacencia (con pesos). 29](#_Toc44685046)

[Figura 3.12. Inteligencia Artificial y el ser humano… Fuente. (Pascual, 2019). 32](#_Toc44685047)

[Figura 3.13. Representación de un árbol de búsqueda Fuente. (Rossel, 2008) 35](#_Toc44685048)

[Figura 3.14. Búsqueda con profundidad iterativa Fuente: (Búsqueda por Amplitud Iterativa, 2017) 37](#_Toc44685049)

[Figura 3.15. Búsqueda Heurística, Ascenso de la colina 40](#_Toc44685050)

[Figura 3.16. Algoritmo minimax. 44](#_Toc44685051)

[Figura 3.17. Algoritmo Alfa-Beta. 45](#_Toc44685052)

[Figura 3.18. Algoritmo de Floyd-Warshall 47](#_Toc44685053)

[Figura 3.19. Algoritmo de Dijkstra. 48](#_Toc44685054)

[Figura 3.20. Agentes Inteligentes. 48](#_Toc44685055)

[Figura 3.21. Entorno de NetBeans 52](#_Toc44685056)

[Figura 3.22. Logo de Eclipse. 53](#_Toc44685057)

[Figura 3.23. Entorno de IntelliJ IDEA. Fuente. Autoría propia 54](#_Toc44685058)

[Figura 3.24. Pastebin 54](#_Toc44685059)

[Figura 4.1. Gameplay de Worms 2: Armageddon. 6](#_Toc44685060)

[Figura 4.2. Logo de IntelliJ IDEA. 7](#_Toc44685061)

[Figura 4.3. Ejemplo de las herramientas graficas de Excel presentada por Microsoft. 8](#_Toc44685062)

[Figura 4.4. Ejemplo de cómo usar las herramientas para compartir datos en Microsoft Teams 9](#_Toc44685063)

[Figura 5.1 Recorrido de un Árbol con el algoritmo de MiniMax 8](#_Toc44685064)

[Figura 5.2 Recorrido de un Árbol con el algoritmo de Alpha-Beta 9](#_Toc44685065)

[Figura 5.4 Interfaz del programa 10](#_Toc44685066)

[Figura 5.5 Interfaz del programa 10](#_Toc44685067)

[Figura 5.6 Interfaz del programa 11](#_Toc44685068)

[Figura 5.7 Algoritmo MiniMax en Java 12](#_Toc44685069)

[Figura 5.8 Algoritmo de Alpha-Beta 13](#_Toc44685070)

[Figura 5.9 MiniMax vs AlfaBeta con una profundidad de 6 niveles 14](#_Toc44685071)

[Figura 5.10 MiniMax vs AlfaBeta con una profundidad de 3 niveles 15](#_Toc44685072)

[Figura 5.11 MiniMax vs AlfaBeta con una profundidad de 1 nivel 16](#_Toc44685073)

[Figura 6.1. Comparación de medidas de tendencia central 4](#_Toc44685074)

[Figura 6.2. Comparación de medidas de tendencia central 4](#_Toc44685075)

[Figura 6.3. Comparación de Estados recorridos en ejecución de una partida 5](#_Toc44685076)

[Figura 6.4. Comparación de tiempo en ejecución de una partida 5](#_Toc44685077)

[Figura 6.5. Comparación de confiabilidad 6](#_Toc44685078)

[Figura 6.6. Comparación de confiabilidad 6](#_Toc44685079)

**Índice de Tablas**

[Tabla 2.1 Relación y orden jerárquico de los problemas. 6](#_Toc44685080)

[Tabla 2.2 Objetivos específicos para la resolución de los problemas. 10](#_Toc44685081)

[Tabla 3.1. Diferencia entre las colas lineales, circulares y de prioridad 20](#_Toc44685082)

[Tabla 3.2. Comparación entre los 3 algoritmos 38](#_Toc44685083)

[Tabla 6.1. Análisis de los datos 3](#_Toc44685084)

[Tabla 6.2. Análisis Estadístico de los datos 3](#_Toc44685085)

[Tabla 6.3. Análisis Estadístico de los datos 3](#_Toc44685086)

[Tabla 6.4. Confiabilidad de los resultados 6](#_Toc44685087)

# **Capítulo I. Introducción**

**Capítulo 1: Introducción**

Los algoritmos de búsqueda en problemas de juegos son técnicas que se basan en encontrar una manera en la que una inteligencia artificial realiza movimientos para evitar que su contrincante, en este caso un ser humano, logre conseguir su objetivo en el juego. Actualmente, existen 2 algoritmos que se usan para que la inteligencia artificial logre contrarrestar los movimientos del usuario, los cuales son: Minimax y Alfa-Beta. Estos algoritmos consisten en recorrer los nodos de un árbol y encontrar cual es el mejor movimiento para que la inteligencia artificial lo realice. Estos algoritmos mezclas, en su funcionamiento, funciones heurísticas y ciertas características de los algoritmos de búsqueda a ciegas o sin información.

Como objetivo general de este proyecto nos planteamos alcanzar un completo funcionamiento del programa (juego) cumpliendo con las pautas y reglamentos establecidos en él, permitiendo así una experiencia agradable para el usuario, dando con esto una completa compresión y usabilidad de este. Para alcanzar este objetivo debemos desenvolvernos adecuadamente a medida que avanza el juego, según los lineamientos establecidos. Una vez terminado, debemos brindar el resultado correcto y que este sea de rápido uso y entendimiento para el usuario.

En este proyecto se busca implementar las técnicas ya mencionadas en el juego de estrategia Worms. En la implementación del algoritmo, usaremos una cantidad de 1 jugador por equipo con 1 gusano a su disposición de cada uno, así como las respectivas armas. Decidimos aplicar una interfaz gráfica sencilla, ya que hoy en día no contamos con el aprendizaje necesario para una realización de un videojuego en su parte gráfica. Es por eso por lo que preferimos utilizar el lenguaje de programación orientado a objetos Java, ya que este ofrece un modo gráfico bastante fácil de utilizar. Para el análisis de los datos obtenidos como resultado de la aplicación de los algoritmos ya mencionados, decidimos utilizar la herramienta de Office Excel, ya que es la que más sabemos utilizar y a nivel mundial es la más usada. Como herramienta adicional hicimos uso de PasteBin, una herramienta para almacenar y compartir proyectos de programación y así facilitar su exportación a todos los integrantes del grupo.

Procedemos a usar las técnicas de investigación cualitativas mediante observación, investigación bibliográfica y estudios de casos, guiándonos a través de los reglamentos del juego y desarrollo en otras plataformas, pero siguiendo la misma lógica.

Nuestra Hipótesis planteada es que el algoritmo Alfa-Beta debe tener una mejor capacidad de respuesta al movimiento realizado por el usuario que, en comparación al Minimax, este encuentre una estrategia rápida sin necesidad de recorrer todas las posibilidades disponibles. Al final de este proyecto veremos si se cumplió, o no, la Hipótesis que planteamos.

El trabajo está constituido de 5 capítulos, en donde desarrollaremos los términos importantes para que se comprenda este trabajo, siendo este capítulo el primero de todos. En el capítulo 2 trataremos el problema que se nos plantea en este trabajo, desde la problemática hasta la solución que proponemos. En el capítulo 3 trataremos quizás la parte más importante y en sí el núcleo de este proyecto, el marco teórico. En el marco teórico desarrollaremos los temas que necesitamos que se comprendan antes de llegar a los resultados de nuestro trabajo, desde que es el juego Worms hasta que es el algoritmo Minimax y el Alfa-Beta, incluyendo también otras soluciones actuales que existen similares a los algoritmos ya mencionados. En el capítulo 4 trataremos el marco metodológico, parte fundamental para llevar a cabo los procesos en una investigación. En el capítulo 5 y 6 presentaremos como funciona nuestra solución y los resultados obtenidos, presentando estos con un análisis estadístico detallado. En la conclusión se verá si se cumplió con los objetivos y/o Hipótesis formulados en esta parte del trabajo.

**Capítulo II.   
El problema**

# **Capítulo 2: El Problema**

* **Planeamiento del problema a tratar**

Nuestro problema se trata acerca de la implementación del videojuego WORMS (1995) de 1er generación. Se pretende que este sea desarrollado usando los algoritmos MiniMax y Alpha-Beta en un lenguaje multiplataforma, con el objetivo de que este se pueda jugar contra una máquina de tal modo que ella pueda manejar y tomar decisiones por sí sola, según las condiciones y reglamentos establecidos en el juego y, de esta manera, poder realizar un análisis comparativo entre los algoritmos y determinar con cuál algoritmo se toma menos tiempo para llegar a la soluciones o estado ganador.

* **Antecedentes**

**WORMS**

Worms es una serie de videojuegos basados en estrategia militar, ya que posee cierta cantidad de jugadores, así como armas para atacar además de la elección de movimientos y posicionamientos.

Esta serie de videojuegos posee hasta la actualidad cuatro generaciones y sus versiones 3D, cada generación con sus respectivos juegos de Worms, los cuales son los siguientes:

**1ra generación**

* Worms (1995)
* Worms Reinforcements
* Worms United
* Worms: The Director's Cut

**2da generación**

* Worms 2
* Worms Armageddon
* Worms World Party

**3ra generación**

* Worms Open Warfare
* Worms (2007)
* Worms Open Warfare 2
* Worms: A Space Oddity
* Worms 2: Armageddon
* Worms Reloaded
* Worms: Battle Islands

**4ta generación**

* Worms Revolution
* Worms 3
* Worms Clan Wars
* Worms Battlegrounds
* Worms W.M.D

**3D**

* Worms 3D
* Worms 4: Mayhem
* Worms Forts: Under Siege
* Worms: Ultimate Mayhem

**Worms (1995)**

Es el primero en esta serie de videojuegos de artillería mayormente desarrollados por Team17, una compañía de desarrollo y edición de videojuegos completamente independiente. El juego fue creado originalmente por Andy Davidson.

El juego se basa en la artillería clásica 2D, donde los jugadores controlan un equipo de gusanos, cada equipo consta de cuatro miembros. El objetivo es matar a todos los miembros de los equipos adversarios dentro de un límite de tiempo establecido por ronda. Cada turno, que dura de 45 a 100 segundos, un jugador puede mover un Gusano seleccionado y usar una o más de las armas y herramientas disponibles.

Originalmente el juego no presentaba gusanos, sino los Lemmings, del popular juego del mismo nombre. Andy Davidson estaba trabajando en un programa llamado "Jack the Ripper" para la computadora personal de Amiga, que le permitió rastrear los contenidos residuales de la RAM después de que las aplicaciones se hubieran ejecutado y salido. De esta manera, "arrancó" los gráficos de Lemmings y los usó mientras desarrollaba su versión de artillería. El nombre original del juego era Lemartillery.

**Algoritmo MiniMax**

Este algoritmo se usa mayormente para la implementación básica de inteligencia artificial en juegos basados para dos jugadores. El algoritmo te dice cuál es el mejor movimiento que un jugador debe elegir en cualquier estado del juego por lo cual, la entrada al algoritmo MiniMax son el estado del juego y de quién es el turno, la salida sería el mejor movimiento que puede jugar el jugador dado en la entrada.

En el desarrollo del algoritmo siempre suele usarse la recursividad para la búsqueda a través del árbol ya que este necesita trabajar por diferentes caminos del árbol, de aquí su nombre MiniMax. Mayormente suele usarse para juegos de dos jugadores, debido a que uno juega obteniendo los beneficios mínimos y el oponente por los beneficios máximos. Al ser ambos jugadores del juego opuestos entre sí, MIN seleccionará el valor minimizado y MAX seleccionará el valor maximizado. Se basa en el concepto de juego de suma cero donde en un juego de suma cero, el puntaje de utilidad total se divide entre los jugadores. Un aumento en el puntaje de un jugador resulta en una disminución en el puntaje de otro jugador. Entonces, el puntaje total es siempre cero. Para que un jugador gane, el otro tiene que perder

**Algoritmo Alpha-beta**

Este algoritmo es una mejora significativa del algoritmo de búsqueda MiniMax que elimina la necesidad de buscar grandes partes del árbol del juego aplicando una técnica de ramificación y unión, la cual se basa en la enumeración sistemática de soluciones candidatas mediante la búsqueda de espacio de estado y antes de enumerar las soluciones candidatas de una rama, la rama se compara con los límites estimados superior e inferior de la solución favorable, y ​​se descarta si no puede producir una solución mejor que la mejor encontrada hasta ahora por el algoritmo. Este algoritmo hace esto sin ningún potencial de pasar por alto un mejor movimiento. Si uno ya ha encontrado un movimiento bastante bueno y busca alternativas, una refutación es suficiente para evitarlo. No es necesario buscar refutaciones aún más fuertes. El algoritmo mantiene dos valores, alfa y beta. Representan el puntaje mínimo que el jugador maximizador tiene asegurado y, el puntaje máximo que el jugador minimizador tiene asegurado respectivamente.

* **Problemática y problemas**

1. **Problemática**

Nuestro objetivo es aplicar y luego comparar cuál de los algoritmos entre MiniMax y Alpha-beta es el más rápido, ágil y adecuado en cuanto a la resolución del videojuego Worms (1995) pero, al tener muchos componentes, cantidad de espacio y movimientos; los árboles que nos pueden surgir en el algoritmo pueden llegar a ser muy grandes, lo cual puede ocasionar que la capacidad de solución sea muy larga, lo cual podría perjudicar la experiencia del usuario.

1. **Problemas**

Con la problemática, se nos genera los siguientes problemas:

1. Tiempo muy largo de ejecución.
2. Deficiencia en búsqueda de solución.
3. Deficiencia en el almacenamiento de variables.

* **Matriz del problema**

|  |  |  |  |
| --- | --- | --- | --- |
| Problemas | Relación de entrada | Relación de salida | Orden jerárquico |
| 1 | ---- | 2,3 | Problema principal |
| 2 | 1,3 | 3 | Problema secundario 2 |
| 3 | 1,2 | 2 | Problema secundario 3 |

Tabla 2.1 Relación y orden jerárquico de los problemas.

* **Soluciones actuales**

En la serie de juegos Worms, se aplica la inteligencia artificial, es un término que se utiliza para indicar que los gusanos enemigos son controlados por CPU, solo hay esa forma para controlarlos. La IA utilizada en la serie Worms tiene múltiples niveles. El nivel de IA es el "cerebro" del equipo de CPU.

Estos están diseñados de la siguiente manera:

En los juegos de 3ra generación hay 7 niveles en total. A veces, aparecen burbujas de pensamiento sobre la cabeza de los gusanos AI, lo que indica que los gusanos planean sus movimientos. Sus ataques según el tipo de gusano suelen ser:

* **Estúpido:** Pelea muy débilmente.
* **Vengador:** Moderado y sediento en busca de venganza. Más probabilidades de dañar a los compañeros de equipo.
* **Imprudente**: usa el viento más que otros gusanos, pero no se preocupa demasiado por su propia seguridad.
* **Arrogante:** le encanta impresionar a los demás con ataques difíciles, arriesgados e innecesarios, le gusta ser visto como gusanos "tecno-estratego".
* **Fuego rápido:** sigue 4 principios dados por un líder desconocido, pero extremadamente suicida.
* **Darksiding:** puede ser cobarde, pero se mantiene 5 pasos por delante de los enemigos en todo momento, y excelente en el ajedrez, lo que también ayuda en las habilidades de batalla.
* **Estratégico:** soldados humildes que piensan, planean y desatan ataques devastadores, pero no funcionan sin ideas.

En los juegos 3D, hay 5 niveles de IA. En Worms 4: Mayhem y Worms: Ultimate Mayhem, los signos de interrogación parpadean sobre las cabezas de los gusanos de la CPU cuando están quietos. Sus ataques según el gusano son:

* **Estúpido:** la mayoría de estos gusanos no golpean a sus enemigos, por lo que a menudo eligen artillería de corto alcance, como Fire Punch.
* **Fácil:** ¡CPU Worms que pueden golpear a sus oponentes, pero eso no significa que sufras grandes pérdidas!
* **Medio:** nivel de habilidad promedio. Fácil de superar para profesionales, pero desafiante para nuevos jugadores.
* **Difícil:** ¡No deberías enfrentarte a este equipo a menos que tengas experiencia y quieras un partido desafiante!
* **Extremo:** estos enemigos son buenos para apuntar, apuntar y golpear a sus oponentes. También tienden a usar trucos en sus tiros.
* **Solución propuesta**

Se pretende solucionar el problema mediante el uso del algoritmo MiniMax y uno el de Alpha-Beta, con el propósito de usarlos para el control de movimientos y puntajes tanto de la maquina como el del jugador real. Con este método veremos como ambos resuelven y trabajan con los cálculos y resolución del juego al declarar un estado ganador, así también determinar cuál es más efectivo y rápido en este caso.

* **Herramientas de Software y Hardware**

**Herramientas de Software**

* **Windows 10**, sistema operativo del hardware. Windows 10 es un sistema operativo desarrollado por la empresa Microsoft que es el más utilizado actualmente. Windows 10 cuenta con una interfaz gráfica bastante agradable para el usuario.
* **Lenguaje JAVA** (Compiladores IntelliJ IDEA y NetBeans). JAVA es un lenguaje de programación orientado a objetos que permite el desarrollo de aplicaciones de escritorio y aplicaciones web. Cabe destacar que el lenguaje JAVA es el lenguaje más utilizado para el desarrollo de aplicaciones de escritorio actualmente.
* **Microsoft Word 365** (para la redacción del informe del proyecto). Word 365 es una herramienta del paquete de office 365 que permite el procesamiento de texto. Word 365 viene anexado al paquete de un año de licencia de herramientas de office (de ahí su nombre (365)).
* **Microsoft PowerPoint** (Para la sustentación del informe). PowerPoint es una herramienta creada por Microsoft para la elaboración de presentaciones digitales. PowerPoint permite crear presentaciones con texto, presentaciones con diapositivas y la inclusión de animaciones e imágenes.
* **Google Chrome** (Como navegador para realizar las investigaciones). Chrome es un buscador gratuito desarrollado por Google, considerado el navegador más utilizado en el mundo por su rapidez, estabilidad y seguridad. Chrome cuenta con un motor de renderizado que se basó en Webkit (una plataforma para el desarrollo de navegadores, en este caso con el lenguaje HTML), cuyo nombre es Blink, que permite el código base y orientar su desarrollo para contribuir a la web.

**Herramientas de Hardware**

* Asus Notebook, 2016, Intel Celeron, @1.6 Ghz, 1601 Mhz, 4 GB de Ram, 465 GB HDD
* Dell Inspiron 15 P66F001 Año:2016, i7 @2.7GHz,16 GB de Ram, 2TB HDD
* Acer Aspire 3 A315-51-31B4, Intel Core i3 de 6ta generación, 6 GB de Ram, 500 GB SSD
* Balboa, Intel Atom, @1.50 Ghz, 2 GB de Ram, 500 GB HDD
* Acer Predator con i7 - 7700hq, 2,81ghz, 24 GB de Ram, 1tb HDD ,250 SSD, GTX 1070
* i5 4690k, 4.1Ghz, 8GB de Ram, 1tb HDD, 250 SSD, GTX 970
* HP, 2018, Intel Core i3, @ 2.0GHz 2.0GHz, 4 GB de Ram, 450 HDD

**Resultados esperados**

Se espera que los algoritmos cumplan con el objetivo, en el caso de la IA buscar el camino optimo, en el caso de ambos que sepa determinar el mejor puntaje y camino según la toma de decisiones que se realizaron.

El alcance previsto para nuestro proyecto es su correcto funcionamiento, entendimiento y presentación a la hora de ser ejecutado, a la vez haciendo sentir al usuario que está compitiendo con un buen oponente y no una simple computadora, lo cual le exige a pesar de forma rápida.

**Objetivos**

|  |  |
| --- | --- |
| Problemas | Objetivos |
| PP. Tiempo muy largo de ejecución. | **OG.** Establecer una cantidad de condiciones apropiadas para funcionar de manera rápida y optima en ambos algoritmos. Se puede resolver en un periodo de 2 días. |
| PS1. Deficiencia en búsqueda de solución. | **OE1.** Establecer condiciones concretas y pasos a seguir entre los parámetros del juego y de los algoritmos para alcanzar la meta establecida. Se propone que la resolución de este problema se puede resolver en un periodo de tiempo de 2 días. |
| PS2. Deficiencia en el almacenamiento de variables. | **OE2.** Establecer un orden concreto y seguro dentro del programa que siga la estructura de los algoritmos según sus entradas y salidas. Se propone que la resolución de este problema se puede resolver en un periodo de tiempo de 2 días. |

Tabla 2.2 Objetivos específicos para la resolución de los problemas.

* **Justificación**

La importancia de este proyecto es que sirve como aporte para profundizar y expandir el área de la inteligencia artificial que se encarga de la implementación de algoritmos de búsqueda en los videojuegos; principalmente, a través del uso de algoritmos heurísticos para una mejor ampliación de estos y para el entretenimiento del usuario, todo ello sirve para cubrir más áreas de implementación, con mayor rendimiento en los videojuegos, tanto en computadoras como otros dispositivos tecnológicos; notando que existen muchas formas y maneras para que un videojuego mejore y sea más asequible.

Los videojuegos suelen desarrollarse para trabajar en modo gráfico, a su vez, de tal manera que el usuario tenga total control para llegar a la meta, para así, ir superando los conflictos que se le presente en el camino, ya sea la maquina u otros obstáculos preestablecidos. Sin embargo, la implementación de este videojuego en los algoritmos MiniMax y Alpha-beta, le da capacidad a la máquina para competir por ella misma sin darle chance al usuario de ganar o sobrepasar los obstáculos, llevándolo aprender y mejorar en el juego, pero no le garantiza llegar a la meta porque la IA al igual que el usuario aprende a tomar las mejores decisiones según crea conveniente.

Se puede proponer como novedad el hecho de que, posee un acercamiento diferente y más sencillo en la implementación de los videojuegos, en términos de su jugabilidad contra una IA y, que el usuario sienta que está contra un digno oponente al mismo tiempo que le ayuda a entrenar y a aprender más, incluso, a mejorar en el juego desarrollado.

Su apoyo a nivel social y nacional pretende proveer de más conocimientos en el desarrollo e implementación a la teoría de juegos, apoyándose de los algoritmos MiniMax y Alpha-Beta, así como reforzar la lógica en el momento de su usabilidad.

El algoritmo que se plantea, al tener una implementación diferente a los demás, hará la diferencia en aportar nuevos puntos de vistas, procedimientos y métodos para el mejor desarrollo y diseño de los videojuegos con el uso de algoritmos de IA. La mayoría de videojuegos, como su nombre lo dice, se codifican a base de modo gráfico, pero el rendimiento y rapidez de este en los dispositivos no siempre suele ser de lo más ideal ya que la capacidad de memoria para cada tipo de video juego varía, esto hace que no sean del todo asequibles ni fácil de poder jugar, por lo cual un acercamiento diferente en algoritmos más sencillos y recursivos que hacen el rendimiento tanto de espacio de memoria como usabilidad sea más eficiente, hace la diferencia.

Se contribuye a resolver el problema planteado brindando diferentes métodos para la resolución del mismo, con la capacidad de cumplir con lo que se exige de él y para lo que está hecho, creando un juego capaz de desarrollarse con fluidez, aplicando los algoritmos de búsqueda heurística a través del uso de los algoritmos MiniMax y Alpha-Beta, los cuales se utilizaron con el propósito de mejorar el rendimiento, eficiencia y usabilidad del videojuego, debido a que nos permite que el videojuego cumpla con las expectativas y condiciones establecidos en él, en cuanto a su jugabilidad se refiere, brindando un interacción mejorada.

**Capítulo III: El Marco Teórico**

# **Capítulo 3: Marco Teórico**

Índice del marco teórico

1. **Worms**
   1. **¿Qué es worms?**

Worms es un videojuego de estrategia en el que uno o dos jugadores se enfrentan a uno o más personajes dentro de un cierto período de tiempo para eliminar el personaje del oponente. En este caso, el personaje está representado por un gusano, en una pequeña isla que flota en una gran área de agua, con una atmósfera de dibujos animados, con una animación de dibujos animados espléndida y humorística y varias armas de ficción. Worms cuenta con muchas versiones en las que se mejoraba la jugabilidad y su interfaz gráfica. La última versión desarrollada fue en 2016, cuyo título fue “Worms W.M.D” (Wikipedia, La enciclopedia libre., 2020).

* 1. **Historia**

El juego fue creado originalmente por Andy Davidson, un artículo en la competencia de programación Blitz BASIC organizada por la revista "Amiga Format". La revista Amiga Format es una versión simplificada del lenguaje de programación que se ha introducido antes. La primera versión se introdujo en el mercado en 1995. Sugirió que controlemos un grupo de gusanos, usemos varios artilugios y armas, y matemos a los gusanos que componen el equipo contrario de varias maneras. Este método básico de jugabilidad se desarrolló en un entorno donde cada gusano puede interactuar con él, ya sea que estén cavando túneles, escalando o dinamizando ciertas partes para crear nuevas áreas para escalar o protegerse de los ataques (Zonared, s.f.).

* 1. **Jugabilidad**

En los años de desarrollo, la franquicia se ha perfeccionado en dos aspectos: el uso de 2D en la interfaz, siendo esta la más utilizada, contando con hasta cuatro generaciones de juego, como en entornos 3D. En el caso de los títulos basados ​​en el entorno dimensional, las escenas de batalla se generan aleatoriamente, por lo que debemos formular varias estrategias de acuerdo con la situación real, y nuestras tropas pueden morir debido al aumento repentino del nivel del agua o al colapso de ciertas áreas debido a algún elemento que no estaba contemplado en nuestros planes. La física juega un papel fundamental, porque las granadas y otras armas de alta energía no solo dañan a nuestro equipo debido a explosiones, sino que también las arrojan al medio ambiente, chocan con todos los elementos que se encuentran en el camino y pierden vitalidad hasta que ven Cuando desaparezcan para liberar una pequeña lápida, indicará dónde hemos perdido una unidad. El generador de mapas aleatorio proporciona fuentes ilimitadas de terreno colorido para varios temas. Un mapa de isla abierta como este permite a los jugadores usar ataques aéreos. El techo del mapa de la cueva es casi indestructible y no se puede pasar fácilmente (Zonared, s.f.).

Las armas disponibles en el juego van desde granadas temporizadas estándar y misiles autoguiados hasta ovejas explosivas y bombas de plátano altamente destructivas (que pueden involucrar armas en juegos de gorilas). Estas dos armas aparecerán en cada juego de Worms.



Figura 3.1. Worms. Fuente. (Ríos, 2012)

1. **Estructuras de datos**

Para comprender este proyecto primero debemos comprender cómo trabaja nuestro proyecto a nivel interno y lógico, por lo cual debemos definir ciertos conceptos con los que vamos a trabajar a lo largo de este trabajo.

* 1. **Pilas y colas**
     1. **Definición**

**Pila**

Es una lista secuencial o estructura de datos en la que el acceso a sus elementos es del tipo LIFO (del inglés, último en entrar, primero en salir, último en entrar), lo que permite el almacenamiento y la recuperación de datos. Debido a su simplicidad y disposición implícita de la estructura misma, se ha utilizado muchas veces en informática. El último elemento ingresado en la pila será el tope, y el primer elemento ingresado será el final de la pila.

Hay dos operaciones básicas para la gestión de datos: apilar (empujar), colocar objetos en la pila; y sus operaciones inversas, eliminar (o cancelar el apilamiento, reventar) y eliminar el último elemento apilado (Ecured Contributors, 2014).

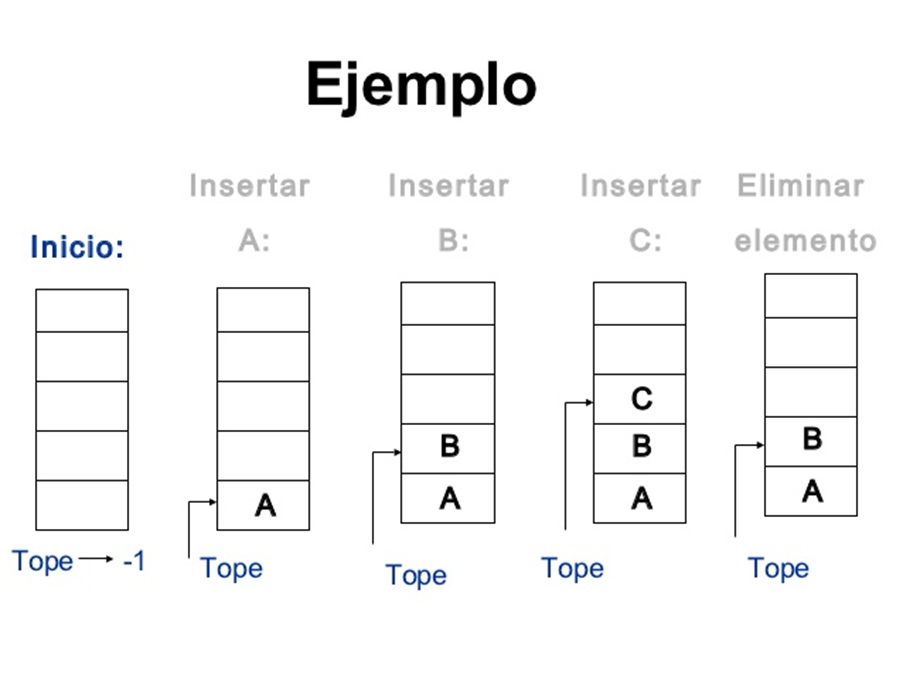


Figura 3.2. Representación de una pila. Fuente. (Hasif, 2016)

**Cola**

Una cola es una estructura de datos que se caracteriza por su estilo de ingreso de datos (Primero que entra, primero que sale). Las colas se utilizan en sistemas informáticos, transporte y operaciones de investigación (entre otras operaciones), donde los objetos, las personas o los eventos se tratan como datos, que la cola almacena y guarda para su posterior procesamiento. Este tipo de estructura de datos abstractos se implementa en un lenguaje orientado a objetos en forma de una lista vinculada mediante clases.

Las operaciones básicas que podemos realizar en una cola son la de insertar, eliminar, ver el frente, ver el último (Miguel, s.f.).

* + 1. **Tipos**

Solo existe actualmente un tipo de pila, sin embargo, se puede representar en la programación mediante arreglos o mediante punteros (esta última en caso de que se esté manejando nodos que tengan una dirección en memoria).

En las colas, existen 3 tipos de colas usadas actualmente, las cuales son:

* Cola lineal
* Cola circular
* Cola de prioridad

**Cola lineal**: Las colas lineales son teóricamente listas primero en entrar, primero en salir. Se llama lineal porque se asemeja a una línea recta, colocando elementos uno tras otro. Una colección homogénea que contiene elementos, donde se agregan nuevos elementos en un extremo y luego se eliminan del otro extremo (Gadget - Info, 2019). Un ejemplo típico de una cola lineal es el manejo de una clínica con sus pacientes… El primero que entra es el primero que sale.

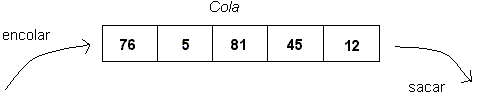


Figura 3.3. Cola lineal. Fuente. (Universidad de Chile, s.f.)

**Cola circular**: La cola circular es una variación de la cola lineal, que supera las limitaciones de la cola lineal. En la cola circular, si el último está ocupado y hay espacio libre, el nuevo elemento se agregará a la primera posición de la cola. Cuando se trata de colas lineales, solo se pueden insertar desde la parte trasera y quitar desde la parte delantera. En la cola completa después de realizar una serie de eliminaciones consecutivas en la cola, tal situación ocurre, incluso si la condición de desbordamiento (Anterior = max-1) todavía existe y no se pueden agregar nuevos elementos (Gadget - Info, 2019).

Hay ciertas reglas que debe cumplir una cola circular, las cuales son:

* La cola está vacía si Frente = Anterior.
* Si se inserta un nuevo elemento, la cola incrementa un valor, es decir, Anterior = Anterior + 1
* Frente debe ser igual al primer elemento
* Final es el último valor ingresado
* Si se elimina un elemento de la cola, ya que la cola extrae en FIFO (First in, First out), frente será igual al elemento anterior, es decir, Frente = Frente + 1

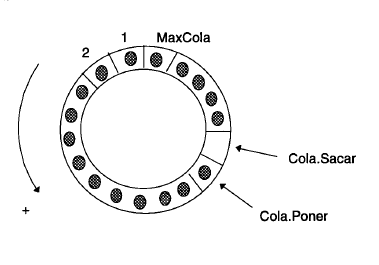


Figura 3.4. Cola Circular. Fuente. (Código Inicial, 2013)

**Cola de prioridad**: una cola de prioridad es un tipo de cola muy utilizado en los procesos de búsqueda en árboles, ya que permite guardar un valor de otra estructura de datos en esta. La utilidad que tiene la cola de prioridad en los algoritmos de búsqueda es que almacena los nodos visitados para evitar recorrer de nuevo a estos.

En este tipo de colas se pueden hacer todas las operaciones que se hacen en los tipos de colas anteriores (insertar, eliminar).

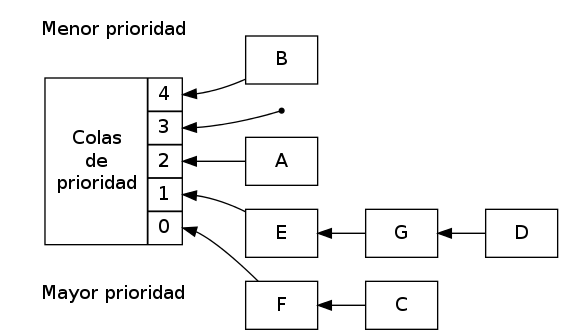


Figura 3.5. Cola de prioridad. Fuente. (Programacion facil estructura de datos, s.f.)

Ahora veamos la diferencia entre los 3 tipos de colas vistos anteriormente.

Tabla 3.1. Diferencia entre las colas lineales, circulares y de prioridad

|  |  |  |  |
| --- | --- | --- | --- |
| Diferencias | Cola Lineal | Cola Circular | Cola de prioridad |
| Inserción | Se ingresa el elemento al final de la cola | Se ingresa el elemento al final de la cola y se conecta con el primer elemento | Comienza con la creación de un hueco en la siguiente posición disponible d. Se añade el elemento con su correspondiente prioridad |
| Orden de ejecución | Se realiza en el orden First in, First out (FIFO) | El orden puede cambiar | Depende de la prioridad que se le dé |
| Utilidad | Sirve para manejar datos en la vida real que sea del primero que llega es el primero que sale (clínicas, Autorápido, banco) | Se utiliza en los sistemas operativos, en los que la cola mantiene la información que se lee de un archivo y que se escribe en él | Sirve para añadir un nodo visitado en una búsqueda en un árbol |

Fuente. Autoría propia

* + 1. **Implementación en Java**

**Pilas**

Para crear una pila en JAVA se necesita instanciar a la clase Stack (pila en español).

**Stack pila = new Stack();**

Para ingresar un valor a una pila, necesitamos escribir lo siguiente:

**pila.push(Integer.toString(x));**

Para sacar un elemento de la pila escribimos lo siguiente:

**pila.pop()**

**Colas**

Las colas en JAVA son muy fáciles de implementar, ya que es una estructura de datos definida (no es necesario crear un arreglo para su representación). Se implementa de la siguiente manera:

Queue<Integer> cola=new LinkedList(); //Es una cola lineal

Queue <Nodo> cola = new LinkedList<>(); //Es una cola de prioridad

int[] cola; cola=new int[tam]; inicio=0; fin=inicio; //Cola circular

* 1. **Árboles y grafos**
     1. **Definición**

**Árboles**

El árbol es la estructura de datos más utilizada, pero también es una de las estructuras de datos más complejas, su característica es que el árbol almacena sus nodos de manera jerárquica en lugar de hacerlo de forma lineal (como listas enlazadas, colas, pilas, etc.) (Blancarte, 2014).

Desde una perspectiva computacional, son estructuras de datos, lo que significa que cada elemento llamado nodo u hoja contiene un valor. Su investigación corresponde a la teoría de la base de datos: en esta terminología, los nodos que dependen de otros nodos se denominan nodos hijos. Cada hoja de trabajo puede tener como máximo un nodo secundario. Si no hay un nodo secundario, se llama nodo secundario.

Un árbol es una estructura de datos no lineal, donde cada nodo puede apuntar a uno o más nodos. A menudo se da una definición recursiva: un árbol es una estructura compuesta de datos y varios árboles (Blancarte, 2014).

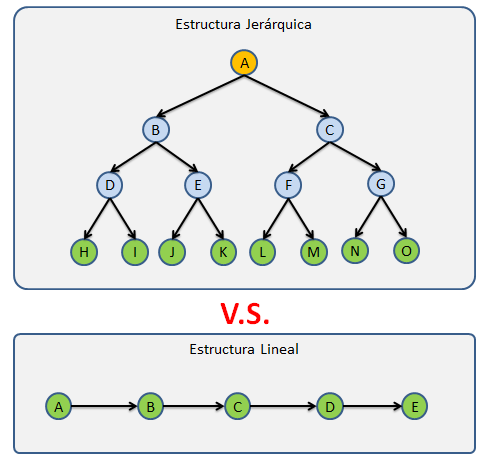


Figura 3.6. Árboles. Fuente. (Blancarte, 2014)

Datos a tomar en cuenta en los árboles:

* Nodo: elemento que contiene un árbol
* Nodo Raíz: es el primer elemento del árbol. Solo puede existir una raíz.
* Nodo Padre: nodo que contiene al menos un hijo
* Nodo Hijo: nodo que tiene al menos un padre (que se genera a partir de otro nodo)
* Nodo hermano: Son aquellos nodos que comparten un mismo padre
* Nodo hoja: último nodo del árbol (que no tiene más descendientes)
* Nodo Rama: todos aquellos nodos que no son nodo raíz
* Nivel: el nivel en un árbol está dado por el nodo de máximo nivel
* Altura: número máximo de niveles de un árbol

La altura de un árbol determina la eficiencia de las operaciones más definidas en el árbol. En algunos casos, podemos eliminar el subárbol del árbol para procesarlo por separado, de modo que el subárbol se convierta en un árbol independiente, también podemos eliminar el subárbol completo, agregarlos y otras operaciones (Blancarte, 2014).

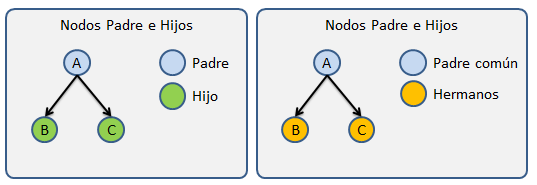


Figura 3.7. Muestra que es un nodo padre y un nodo hijo. Fuente. (Blancarte, 2014)

**Grafos**

Un grafo es una estructura de datos que consiste en un conjunto de nodos (también llamados vértices) y un conjunto de arcos (aristas) que establecen relaciones entre nodos (Vitriago, 2015).

**Elementos de un grafo:**

* Arista: es la entre dos vértices de un grafo
* Vértice: unidad de la que está formado un grafo

**Características de los grafos:**

* El número de vértices de un grafo se le llama grado del grafo
* Un vértice es adyacente con otro si existe un arco que los une
* Un camino es una secuencia de vértices unidos con aristas
* Si dos aristas convergen en el mismo vértice, se dice que son adyacentes.
* Si los puntos inicial y final son iguales, las dos aristas se llaman paralelas.
* Si dos vértices tienen una arista que los conecta, entonces son vértices adyacentes
* Si un vértice no tiene más adyacentes, entonces se dice que son vértices terminales. Fuente. (Vitriago, 2015).
  + 1. **Tipos**

**Tipos de árboles**

**Árboles Completos**

Se caracteriza porque todos sus nodos hojas terminan en el mismo nivel.

**Árboles Binarios**

Un árbol binario es un tipo de árbol que se caracteriza por que cada uno de los nodos padre tiene máximo 2 nodos hijos (Blancarte, 2014).

**Árbol Binario completo**

Un árbol binario se dice que es completo, si todos los nodos del árbol, excepto los del ultimo nivel tienen dos hijos.

**Árbol Binario de Búsqueda**

Son árboles binarios, donde para cada nodo, el valor de la clave raíz del subárbol izquierdo es de hecho menor que el valor de la clave del nodo, mientras que el valor de la clave raíz del subárbol derecho es mayor que el valor de la clave del nodo. La operación de ABB es similar a la operación realizada en un árbol binario general.

Tipos de grafos

**Grafo dirigido**

Se dice que un grafo es dirigido cuando las aristas tienen una dirección definida,



Figura 3.8. Representación de grafos dirigidos. Fuente. (Vitriago, 2015).

**Grafo no dirigido**

Un grafo es no dirigido cuando las aristas no tienen una dirección en concreto.

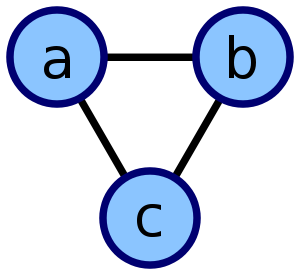


Figura 3.9. Grafo no dirigido. Fuente. (Vitriago, 2015)

* + 1. **Recorridos**

**Árboles**

Pre-orden: el recorrido comienza en la "raíz" y luego atraviesa cada subárbol en un orden predeterminado de izquierda a derecha.

In-orden: se recorre el árbol izquierdo, luego la raíz y por último el árbol derecho

Pos-orden: se recorre el árbol izquierdo, luego el árbol derecho y al final la raíz.

**Grafos**

Un grafo se puede recorrer de dos formas, las cuales son: profundidad y anchura

**Recorrido en anchura**

El recorrido de búsqueda en anchura es un resumen de la ruta jerárquica del árbol. Se trata de acceder al nodo inicial y luego acceder a todos los nodos lejos del arco de distancia de este, luego todos los nodos que están a dos radianes de distancia, y así sucesivamente, hasta que se alcancen todos los nodos accesibles desde el nodo inicial.

**Recorrido en profundidad**

Este método es un método básico al atravesar recursivamente un grafo. Los viajes de pre-orden y pos-orden de un árbol binario se basan en este viaje. Supongamos que una persona está en un sistema de cuevas interconectadas, o en Un laberinto, en una determinada intersección (o nodo), y luego pídale a esta persona que encuentre salida, se encuentra en un nodo. En esta búsqueda, puedes usar varias opciones.

* + 1. **Implementación en Java**

**Árbol**

Un árbol en JAVA se crea de la siguiente manera:

Nodo raiz = new Nodo(); //Se crea la raíz

Nodo nodo1 = New Nodo(); //Se crea el nodo

Los recorridos:

private static void preOrden(Nodo raiz) {

if (raiz != null) {

System.out.print(raiz.getDato() + " - ");

preOrden(raiz.getNodoIzquierdo());

preOrden(raiz.getNodoDerecho());

}

}

//Metodo Inorden

private static void inorden(Nodo raiz) {

if (raiz != null) {

inorden(raiz.getNodoIzquierdo());

System.out.print(raiz.getDato()+ " - ");

inorden(raiz.getNodoDerecho());

}

}

//Metodo PostOrden

private static void posOrden(Nodo raiz) {

if (raiz != null) {

posOrden(raiz.getNodoIzquierdo());

posOrden(raiz.getNodoDerecho());

System.out.print(raiz.getDato() + " - ");

}

}

**Grafos**

Los grafos en JAVA se pueden crear de la misma manera que los árboles, ya que los grafos son árboles. Otra manera de crear un grafo es con un arreglo que represente la matriz de adyacencia y que almacenemos en los pesos que ingresemos y así podríamos ver la relación entre los vértices, el camino mínimo para llegar de un vértice a otro, etc....En la siguiente imagen un ejemplo claro.

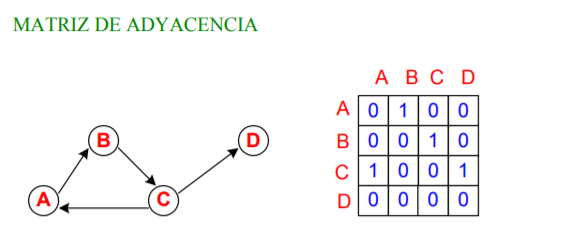


Figura 3.10. Matriz de adyacencia de un grafo (sin pesos). Fuente. (Lipschutz, s.f.)

Si se utilizan los pesos de las aristas entonces se representaría de la siguiente manera:

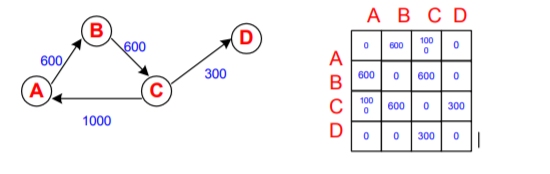


Figura 3.11. Matriz de adyacencia (con pesos). Fuente. (Lipschutz, s.f.)

* 1. **Objetos**

Un objeto en POO (Programación orientada a objetos) es un miembro de la clase que posee todos los atributos definidos en la clase dentro de la cual ha sido creado. Un objeto es capturar algo de la vida real y programarlo (objetivo de la POO). Para crear un objeto en JAVA solo falta escribir la siguiente instrucción:

**Nombre\_Clase objeto = new objeto ();**

1. **Teoría de juegos**
   1. **Definición**

La teoría de juegos es una rama de las matemáticas adecuada para la economía, la sociología, la biología y la psicología y analiza las interacciones entre las personas que toman decisiones en el marco de mecanismos formales de incentivos (juegos). En el juego, varios agentes intentan maximizar su efectividad eligiendo ciertos planes de acción. El beneficio final para cada persona depende del curso de acción elegido por otros. La teoría de juegos es una herramienta que ayuda a analizar problemas de optimización interactiva. La teoría de juegos tiene muchas aplicaciones en las ciencias sociales. La mayoría de las situaciones en la investigación de teoría de juegos implican conflictos de intereses, estrategias y dificultades. De particular preocupación es que cuando los agentes cooperan entre sí, se pueden obtener mejores resultados que cuando los agentes intentan maximizar su efectividad solamente (Anzil, 2006).

* 1. **Historia**

John von Neumann realizó la primera investigación científica sobre tácticas de juego en 1928, en un trabajo titulado “Sobre la teoría de juegos de salón” consideraba juegos de suma cero; aquellos donde lo que gana un jugador es lo que pierde el otro. La teoría de Neumann se llama "minimax", que se basa en obtener el mayor beneficio en el peor de los casos.

En 1950, John Nash hizo lo mismo para los juegos que no son de suma cero. Este científico digital saca una conclusión sobre el concepto de "Equilibrio de Nash", en el que los jugadores usan la mejor estrategia en consideración de otros oponentes. Un ejemplo de esto es el conocido dilema del prisionero. La mejor estrategia es no exponer al compañero y la sentencia (Cardescu web, 2016).

* 1. **La teoría de juegos en el campo computacional**
  2. **¿Cómo se relaciona la teoría de juegos con la inteligencia artificial?**

La economía, como un dominio experto para la toma de decisiones, porque implica pérdida o ganancia de rendimiento; proporciona una serie de teorías para la IA (teoría de la decisión: combinación de teoría de la probabilidad y teoría de la utilidad; teoría del juego adecuada para pequeñas economías; proceso de decisión de Markov -Proceso secuencial; etc.) Le permiten tomar "buenas decisiones". En la teoría de juegos, se intenta determinar la mejor estrategia analizando la interacción del juego con los mismos participantes. Dado un esquema de negociación específico que involucra agentes de negociación, las técnicas de teoría de juegos pueden aplicarse para resolver dos problemas clave: diseñar protocolos apropiados (que controlarán la interacción entre los agentes participantes) y diseñar listas de decisiones para estrategias (modelos). Usado durante la negociación (Universidad Pedagógica y Tecnológica de Colombia, 2014).

* + 1. **Inteligencia artificial: Definición y usos**

La inteligencia artificial es el campo científico de la informática, y se centra en crear programas y mecanismos que puedan mostrar lo que se considera un comportamiento inteligente. En otras palabras, la inteligencia artificial es el concepto de "las máquinas piensan como los humanos" ( SALESFORCE LATINOAMÉRICA, 2017).

Según la RAE, inteligencia artificial es la disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico (Real Academia Española, 2019).



Figura 3.12. Inteligencia Artificial y el ser humano… Fuente. (Pascual, 2019).

Actualmente, la inteligencia artificial automatiza el aprendizaje y el descubrimiento repetidos a través de los datos. Reemplaza la ejecución automática de tareas manuales y realiza frecuentes tareas computarizadas de gran capacidad de manera confiable y sin fatiga. A pesar de todo esto, la intervención humana en estos procesos aún sigue siendo crucial.

La Inteligencia Artificial añade inteligencia a un producto que ya existe, ya que es poco común que la inteligencia artificial se venda como un producto independiente. La automatización, las plataformas de diálogo, los robots y los teléfonos inteligentes se pueden combinar con grandes cantidades de datos para mejorar muchas tecnologías en hogares y lugares de trabajo, desde inteligencia de seguridad hasta análisis de inversiones.

Cabe destacar que la IA logra una increíble precisión por medio de las redes neuronales profundas. La capacitación de modelos de aprendizaje profundo requiere muchos datos porque aprenden directamente de los datos. Cuantos más datos pueda proporcionar, más precisos serán.

Toda la información obtenida de: (SAS Corporation, s.f.).

El futuro de la Inteligencia Artificial es bastante prometedor, ya que tras la pandemia del Covid-19 los seres humanos nos hemos dado cuenta de la importancia de la tecnología actualmente. Pero… ¿Cómo influye la inteligencia artificial en esto? Simplemente el uso de algoritmos productivos ha sido un factor de suma importancia en estos días, tanto que se crearon algoritmos predictivos que por probabilidad y recolección de datos predecían qué tanto iban a subir el número de casos del coronavirus en una región.

Mirando al futuro más cercano, e incluso al futuro lejano, el desarrollo de la IA también se enfrenta a grandes desafíos para proporcionar las herramientas necesarias para que la sociedad en el siglo XXI atraiga un progreso imparable. En este camino, los expertos en muchos campos han visto cómo las computadoras programadas con IA ayudarán a resolver la mayoría de las tareas futuras, a pesar del hecho de que los factores humanos parecen irremplazables en este proceso. Las personas y las máquinas trabajarán lado a lado (ITCL Noticias, 2019).

* + 1. **Algoritmos de búsqueda en inteligencia artificial**

El algoritmo de búsqueda es una rama muy importante de la inteligencia artificial, que incluye la búsqueda en el espacio de estado. Las búsquedas a ciegas (primero en amplitud, primero en profundidad) y las búsquedas informadas (como el algoritmo de A estrella, Hill Climbing) y muchas otras búsquedas se han estudiado en esta disciplina (Cenamor, 2019). Durante el proceso de creación o diseño de un programa con algoritmos de búsqueda se necesita seguir una serie de pasos para su correcta realización con los espacios de estados. Algunas de estas técnicas son:

1. Formulación de metas: Proceso que toma como base la situación de un momento dado, para tomar una determinación en relación con otros factores que afectan lo deseable de las diversas maneras de alcanzar una meta.
2. Formulación de problemas: Proceso que consiste en decidir qué acciones y estados habrán de considerarse. El diseño del programa de búsquedas se reduce a formular una meta y el problema que hay que resolver.
3. Búsqueda: Proceso en el que se evalúan las diversas secuencias de acciones posibles que conducen a estados cuyo valor se conoce, y luego decidir por la mejor.
4. Ejecución: La entrada es un problema, y la respuesta es una solución que adopta la forma de secuencia de acciones. Una vez encontrada una solución, se procede a ejecutar las acciones que ésta recomienda.

Obtenido de: (Béliz, 2020)

Los algoritmos de búsqueda en inteligencia artificial poseen una característica que son los elementos de búsqueda, estos configuran la definición de un problema y las acciones. Algunos de estos elementos son:

* Conjunto de estados: todas las configuraciones posibles en el dominio.
* Estados iniciales: estados desde los que partimos.
* Operadores: se aplican para pasar de un estado a otro.
* Costo de la ruta: Es una función mediante la que se asigna un costo a una ruta determinada.
* Espacio del Estado: Es el conjunto de todos los estados que pueden alcanzarse a partir del estado inicial mediante cualquier secuencia de acciones.
* Estados finales: las soluciones del problema.
* Solución: Es la salida producida por un algoritmo de búsqueda.

Obtenido de: (Béliz, 2020)

* + - 1. **Algoritmo de búsqueda a ciegas o sin información**

La búsqueda a ciegas es un tipo de búsqueda en inteligencia artificial que no utiliza información del problema para resolverlo. Para realizar este tipo de búsquedas requerimos de un árbol de búsqueda (no es necesario que sea binario) (Rossel, 2008).

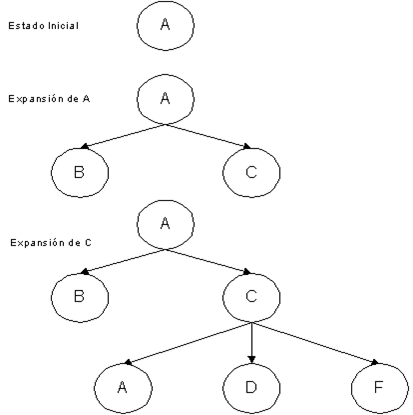


Figura 3.13. Representación de un árbol de búsqueda Fuente. (Rossel, 2008)

**Algoritmo Primero en anchura (BFS)**

El algoritmo de búsqueda a ciegas Primero en anchura es una estrategia de búsqueda de las más sencillas que hay. Se basa en expandir el nodo que se encuentra en igual profundidad en el árbol. Primero, la raíz se expande (estado inicial), luego todos los sucesores de la raíz generan un nuevo conjunto de nodos, estos nodos se expanden, etc. En esta estrategia se omiten los estados ya expandidos en una iteración anterior (Rossel, 2008).

Cabe señalar que uno de los problemas de esta estrategia es la capacidad que tiene la computadora de procesar información.

**Algoritmo Primero en profundidad (DFS)**

Es un tipo de búsqueda que expande el nodo más profundo del árbol y cuando ya no puede ser expandido vuelve hacia atrás para buscar otro nodo a expandir (Rossel, 2008). El algoritmo de primero en profundidad se implementa como una pila (last in first out). Los sucesores se insertan al principio de la pila. Cabe señalar que esta búsqueda tampoco es completa, ya que puede derivar en una rama infinita. El tiempo de ejecución crece de manera exponencial y el espacio de memoria necesario crece de forma lineal con el tamaño del problema (Martinez, 2016).

Este algoritmo es aplicable cuando:

* Existe más de una solución
* Existen soluciones a poca profundidad
* No se tiene mucho espacio en memoria

El algoritmo funciona de la siguiente manera:

* Se expande la raíz
* Se toma un nodo de los recién expandidos
* Se expande el nodo tomado
* Se toma un nodo del nodo expandido anterior

Este algoritmo usa la estrategia de Backtracking. Backtracking o búsqueda atrás es una estrategia que funciona para hacer una búsqueda de todos los estados posibles dentro de un espacio de búsqueda (Baier Aranda).

Las ramas del árbol de búsqueda pueden ser ilimitadas. En este caso, el algoritmo de primero en profundidad no tendrá ninguna desventaja, porque cuando se resuelve gradualmente, el algoritmo más cercano se cubre primero, y si hay una solución, lo encontrará (Rossel, 2008).

**Algoritmo Primero en profundidad iterativa (DFS)**

El algoritmo de búsqueda primero en profundidad iterativa, también llamado algoritmo BPI, es una estrategia que, tal como dice su nombre, se caracteriza porque se realizan un número de búsquedas limitadas y se incrementa el límite de profundidad por cada iteración (Villacis, 2013).

Es importante saber que la búsqueda con profundidad iterativa permite terminar la búsqueda en cualquier momento sin consecuencias nefastas.

La búsqueda primero en profundidad iterativa funciona de la siguiente manera:

* Se define una profundidad predefinida
* Se crea el árbol realizando la búsqueda en profundidad hasta el límite definido anteriormente.
* Si se encuentra la solución, el programa termina. Si no se encuentra solución entonces se establece un nuevo límite y se realiza todo este procedimiento de nuevo (podríamos considerarlo recursividad) (Calderón de la Cruz, 2014)

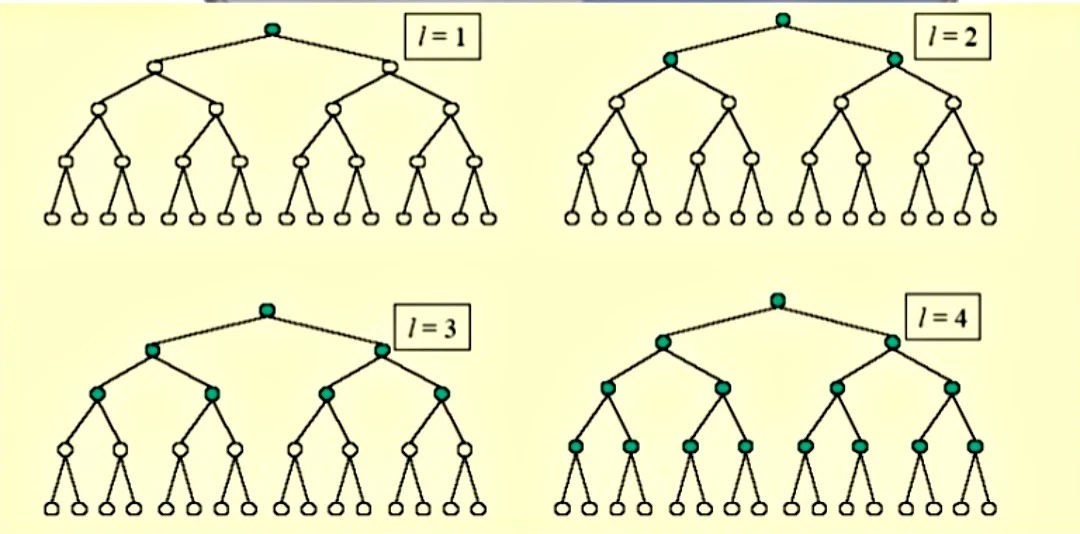


Figura 3.14. Búsqueda con profundidad iterativa Fuente: (Búsqueda por Amplitud Iterativa, 2017)

Tabla 3.2. Comparación entre los 3 algoritmos

|  |  |  |
| --- | --- | --- |
| Búsqueda | Algoritmo | Diferencias |
| Profundidad | * Se expande la raíz * Se toma un nodo de los recién expandidos * Se expande el nodo tomado * Se toma un nodo del nodo expandido anterior | Expande el nodo más profundo del árbol y cuando ya no puede ser expandido vuelve hacia atrás para buscar otro nodo a expandir |
| Anchura | Se basa en expandir el nodo que se encuentra en igual profundidad en el árbol. Primero, la raíz se expande (estado inicial), luego todos los sucesores de la raíz generan un nuevo conjunto de nodos, estos nodos se expanden, etc. | Expande el nodo que se encuentra en igual profundidad en el árbol. Pero este método puede no encontrar una solución ya que todo depende de la capacidad de procesamiento de la computadora. |
| Profundidad iterativa | * Se define una profundidad predefinida * Se crea el árbol realizando la búsqueda en profundidad hasta el límite definido anteriormente. * Si se encuentra la solución, el programa termina. Si no se encuentra solución entonces se establece un nuevo límite y se realiza todo este procedimiento de nuevo (podríamos considerarlo recursividad) | Se caracteriza porque se realizan un número de búsquedas limitadas y se incrementa el límite de profundidad por cada iteración. |

* + - 1. **Algoritmos de búsqueda heurística**

El método de búsqueda heurística tiene como objetivo reducir la cantidad de búsqueda requerida para encontrar una solución. Cuando el problema aparece en forma de un árbol de búsqueda, el método heurístico intenta reducir el tamaño del árbol cortando nodos prometedores (González). En la clase de IA utilizamos 3 algoritmos de búsqueda heurística, los cuales son:

* Búsqueda Heurística, Primero el mejor
* Búsqueda Heurística, Ascenso de la colina o hill climbing
* Búsqueda Heurística, A\* (A estrella)

**Algoritmo de Búsqueda, Primero el mejor**

El algoritmo de búsqueda, primero el mejor, es una estrategia cuya función es expandir el nodo que más cerca esté del objetivo. Evalúa los nodos utilizando una función heurística denotada de la siguiente manera:

f(n) = h(n)

En el algoritmo Primero el mejor, se crea una cola con prioridad que vaya almacenando los nodos visitados y su coste. Cabe destacar que en cada iteración se escoge el nodo más cercano a la solución, lo cual podría no darnos una solución clara (Ceccaroni, 2007).

Algoritmo:

1. Sea L una lista de nodos iniciales.

2. Sea N el nodo más cercano a la meta.

1. Si L está vacía falla.

3. Si N es la meta.

1. Regrese la trayectoria desde el nodo inicial al nodo N.

4. Si N no es meta.

1. Buscar los hijos de N, colocarlos en L, etiquetarlos con la trayectoria desde el nodo inicial.

2. Retornar al paso 2.

Fuente: (Béliz Osorio, Búsqueda Primero el mejor, 2020).

**Algoritmo de Búsqueda, Ascenso de la colina**

Es una técnica de mejoramiento iterativo, el cual comienza a partir de un punto en el espacio de búsqueda y si el nuevo punto es mejor este se transforma en el actual, si no se evalúa otro punto. El método termina cuando no hay mejorías, o cuando se alcanza un número predefinido de iteraciones. Los algoritmos de ascenso a colina son típicamente locales, ya que deciden qué hacer, mirando únicamente a las consecuencias de las acciones realizadas (López Takeyas).

Este algoritmo cuenta con dos elementos muy importantes, los cuáles son escalada simple y escalada por máxima pendiente.

La escalada simple consiste en dirigirse siempre a un estado mejor que el actual, pero no muestra reportes de estados ya recorridos con anterioridad.

La escalada por máxima pendiente se basa en buscar no solamente un estado mejor que el actual, sino el mejor de todos los estados posibles.

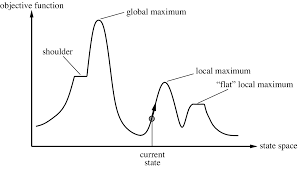


Figura 3.15. Búsqueda Heurística, Ascenso de la colina

Fuente: (Vargas Govea, 2012)

Espacio de estados:

* Máximo local: estado mejor que sus vecinos, pero no mejor que otros que están más adelante.
* Meseta: espacio de búsqueda en el cual todo un conjunto de estados tiene el mismo valor.
* Planicie: áreas en donde la función de evaluación es plana.
* Risco: es un tipo especial de un máximo local.

Como dato adicional, cabe destacar que se pueden evitar los máximos locales regresando a un estado anterior y explorar otra dirección (Béliz, 2020).

Algoritmo:

1. Sea L una lista de nodos iniciales, ordenados según su cercanía a la meta.

2. Tomar el primer nodo N.

1. Si L está vacía falla.

3. Si N es la meta.

1. Regrese la trayectoria desde el nodo inicial al nodo N.

4. Si N no es una meta.

1. Buscar los hijos de N, ordenarlos según su proximidad a la meta etiquetándolos con la trayectoria desde el nodo inicial, agregue los hijos al frente de L.

2. Elimine las primeras trayectorias de la lista

3. Regresar al paso 2.

**Algoritmo de Búsqueda, Búsqueda A\***

Se basa en escoger, en cada iteración, el mejor camino estimado (el primero que encuentre en la cola). Este algoritmo usa una cola de prioridad. Esta prioridad la constituye la siguiente ecuación heurística:

f(n) = g(n) + h(n)

Esta búsqueda tiene similitud con la búsqueda de primero el mejor (Ceccaroni, 2007).

En cada paso del algoritmo, el primer nodo abierto se expandirá. Si no es el nodo objetivo, calcule la f (n) de todos sus nodos secundarios, insértelo en abierto y luego pase el nodo evaluado para cerrar. El algoritmo combina búsqueda de primero en anchura y profundidad: h '(n) tiende a profundidad primero, mientras que g (n) tiende a ancho primero. De esta manera, mientras haya nodos más prometedores, la ruta de búsqueda cambiará. (EcuRed contributors, 2019).

* + 1. **Algoritmos de búsqueda en problemas de juegos**
       1. **¿Qué son?**

Los algoritmos de búsqueda en problemas de juegos son unos tipos de algoritmo que permite que dos jugadores (en este caso un humano y una computadora) disputen una partida de un videojuego en donde la computadora busca ser más efectiva que el ser humano y evitar que gane. Cabe destacar que este tipo de algoritmo cuenta con información y cada movimiento posible del humano se encuentra en un nodo de un árbol.

* + - 1. **Algoritmo Minimax**

En pocas palabras, el algoritmo Minimax consta en seleccionar el mejor movimiento de la computadora, asumiendo que el oponente elegirá el movimiento que puede dañarlo, elegirá la mejor opción, el algoritmo ejecutará un árbol de búsqueda para todos los movimientos posibles y luego atravesará todas las cosas en un estado dado (Es decir, basado en los cuadros ya llenos) árbol de soluciones de juego. Por lo tanto, cada vez que la IA se mueve, se ejecuta minimax (Plasencia Prado, 2016).

Identificamos a cada jugador como un jugador MAX y un jugador MIN. MAX será el jugador que comience el juego, asumimos que somos nosotros. Marcaremos como objetivo encontrar el conjunto de movimientos que proporcionen la victoria a MAX, independientemente de lo que haga el jugador MIN (Fruto, 2008).

Para que el juego se resuelva, debe haber una función de evaluación heurística, que devuelve un valor alto para indicar que el oponente está en buenas condiciones, mientras que un valor negativo indica que el oponente está en condiciones favorables. justo ahora Ya podemos ver cómo, para cada ejercicio, conoceremos su valor en función de su "bondad", y podremos identificar el mejor ejercicio.

Además de utilizar la función heurística anterior, también utilizaremos una estrategia DFS de profundidad limitada para explorar el conjunto de juego. Como es imposible explorar a fondo todas las jugadas, se realizó una búsqueda limitada en profundidad. Por ejemplo, esto significa que no todas las situaciones posibles se estudiarán hasta el final del juego, pero todas las situaciones se buscarán hasta 3 rondas. Aunque esto limitará lo que vemos en el espacio de búsqueda, puede inspirarnos a crear el script de terror definitivo.

Espacio de estados:

* Estado inicial: Es una configuración inicial del juego, es decir, un estado en el que se encuentre el juego.
* Operadores: Corresponden a las jugadas legales que se pueden hacer en el juego.
* Estado final: Determina cuando el juego se acabó.
* Función de utilidad: proporciona valores para la configuración final del juego. En juegos que pueden ganar, perder o empatar, el valor puede ser 1, 0 o -1.
* Función minimax: El algoritmo primero genera un árbol de soluciones completo a partir de un nodo dado. Para cada nodo final, buscamos la función de utilidad de estos. La operación que se realizará cuando regrese el algoritmo Minimax es decirle a la capa superior que llame de forma recursiva, que es el mejor nodo hoja logrado hasta ahora. Cada llamada recursiva debe saber quién debe jugar para analizar si la acción realizada pertenece a la IA o a otros jugadores, porque cuando es el turno de la IA, estamos interesados ​​en maximizar el resultado y cuando el turno del oponente es MINIMIZAR (Plasencia Prado, 2016).

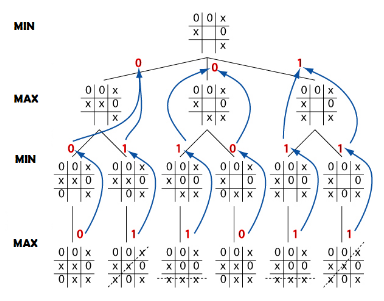


Figura 3.16. Algoritmo minimax. Fuente. (Plasencia Prado, 2016).

* + - 1. **Algoritmo Alfa-Beta**

La poda alfa beta es una técnica de búsqueda que reduce el número de nodos evaluados en el árbol del juego a través del algoritmo minimax. La técnica de poda alfa-beta intenta eliminar la mayor parte del árbol y aplicarlo a un árbol minimax estándar, por lo que el hecho de que la poda de esta rama no afecte la decisión final devolverá la misma acción que devuelve el árbol. La poda alfa-beta se denomina debido al uso de dos parámetros que describen los límites de los valores hacia atrás que aparecen a lo largo de cada ruta. Hasta ahora, ALFA es el valor de la mejor opción de MAX, por lo que esto significa elegir el valor más alto. BETA es, con mucho, la mejor opción para MIN, por lo que implicará elegir el valor más bajo (Millán Villalba, Pérez Cortés, & Ruiz Duarte, 2013).

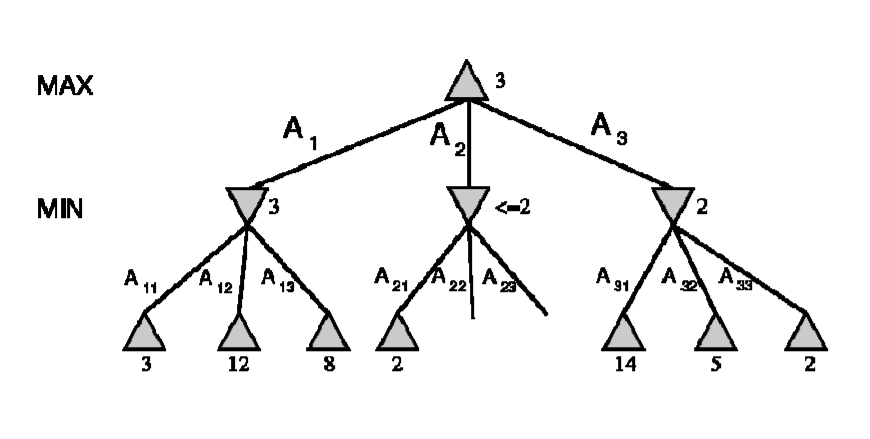


Figura 3.17. Algoritmo Alfa-Beta. Fuente. (Takeyas)

Características:

* Es una mejora del algoritmo minimax
* Se aplica en espacios de estados demasiado grandes como para analizar todos los nodos
* Interrumpe la búsqueda hasta cierto punto y aplica la evaluación heurística a las hojas (profundidad limitada).
* Si el valor del nodo MAX (alfa) es menor que el más alto hasta este momento, entonces se omite el nodo.
* Si el valor del nodo MIN (beta) es mayor que el nodo más bajo hasta el momento, entonces se omite nodo.
  + - 1. **La estrategia en teoría de juegos en la inteligencia artificial**

Cuando un jugador toma una decisión considerando las reacciones de otros jugadores, se dice que el jugador tiene una estrategia. Una estrategia es un plan de acción completo que se ejecuta mientras se juega. Está claro antes de que comience el juego, y estipula cada decisión que el agente debe tomar durante el juego (considerando la información disponible para el agente). La estrategia puede incluir movimiento aleatorio (Anzil, 2006).

Lo que realmente tenemos que hacer es encontrar el valor aproximado del mejor guion, guiarnos a través de la heurística y solo predecir el mejor estado (por ejemplo, con pérdidas), qué puede pasar después del paso equivocado.

Necesitaremos una representación del estado (válido para cualquier estado), otra forma del estado inicial (donde comienza el juego y quién comenzó el juego), otra forma del estado de victoria (a través de la estructura o atributos), y finalmente Defina movimientos efectivos del jugador, que determinarán los movimientos que el agente puede hacer desde el estado actual (Fruto, 2008)

1. **Otros algoritmos de búsqueda**
   1. **Algoritmo de Floyd-Warshall**

En informática, el algoritmo de Floyd-Warshall descrito por Bernard Roy en 1959 es un algoritmo de análisis de grafo utilizado para encontrar la ruta más pequeña entre los gráficos. El algoritmo puede encontrar la ruta entre todos los pares de vértices en una ejecución. Permite calcular la distancia mínima entre dos puntos de un grafo.

Pasos:

* Asignar el valor 0 al nodo origen
* Mediante un proceso iterativo se le asignará a cada nodo Xi un valor n igual a la longitud del camino más corto que exista desde el nodo origen al nodo Xj.

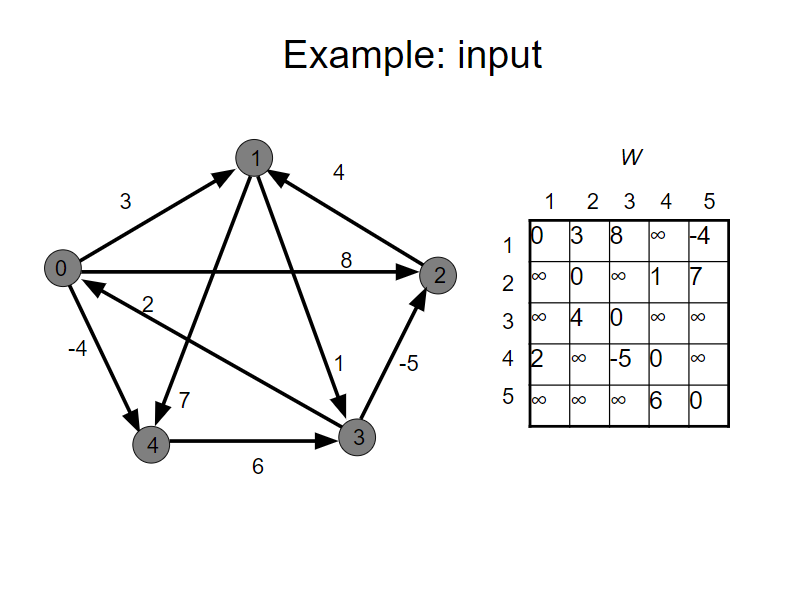


Figura 3.18. Algoritmo de Floyd-Warshall. Fuente. <https://github.com/epomp447/Floyd-Warshall-Algorithm-Java->

* 1. **Algoritmo de Dijkstra**

El algoritmo de Dijkstra (también conocido como algoritmo de ruta mínima) es un algoritmo utilizado para determinar la ruta más corta con pesos en cada vértice de un origen de vértice dado a los vértices restantes en el gráfico. Su nombre hace referencia a Edsger Dijkstra, quien lo describió por primera vez en 1959.

1.- Dado un V0, Dijkstra busca un conjunto D con las menores distancias de V0 al resto de vértices

2.- Al inicio, solo conocemos las distancias de los adyacentes D es inicializada a factor de peso para los adyacentes, Infinito ∞ para los no adyacentes

3.- D va a ser mejorado sucesivamente escogiendo el vértice Vk no elegido antes, que tenga la distancia más corta V0, Vk, probamos si pasando por Vk se puede obtener distancias más cortas de las que tenemos para cada Vértice restante del grafo.

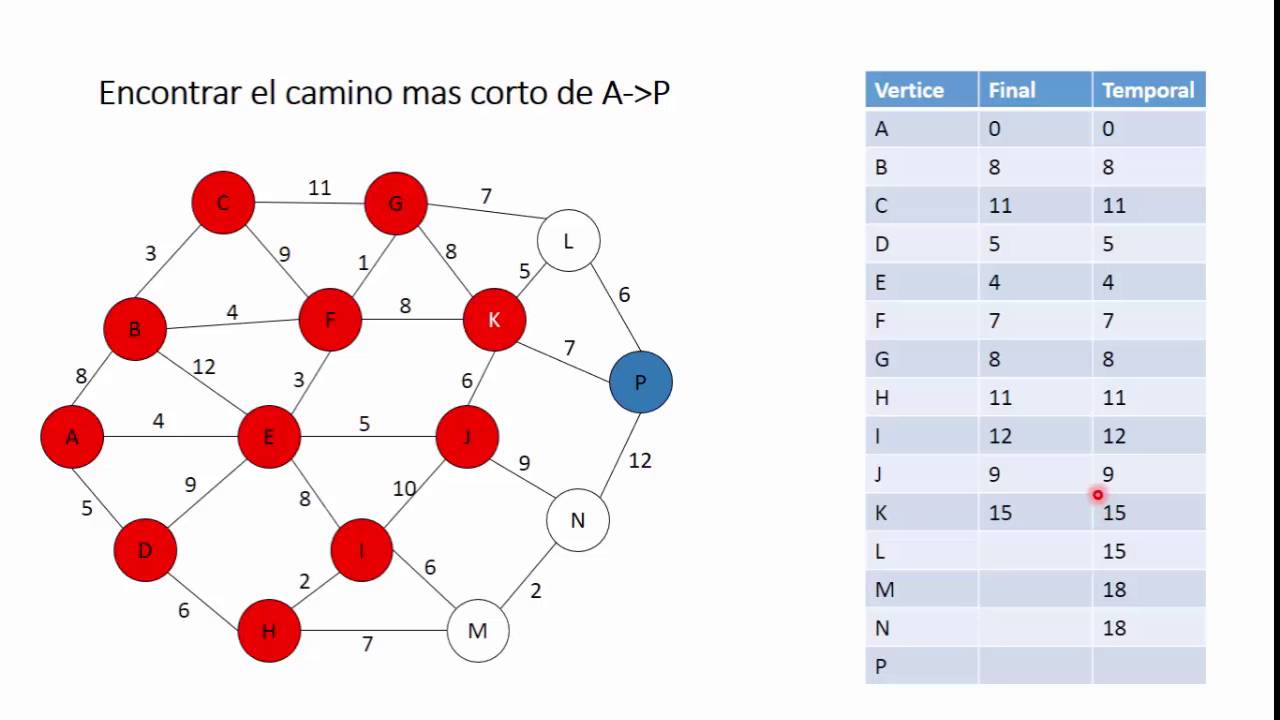


Figura 3.19. Algoritmo de Dijkstra. Fuente. <https://www.youtube.com/watch?v=4I7W5WUQQQI>

* 1. **Estrategias de inteligencia artificial**
     1. **Agentes Inteligentes**

Es una entidad que percibe su entorno a través de sensores y actúa de forma autónoma y razonada con la mejor acción posible sobre ese entorno mediante actuadores. Utiliza el bucle:

PPA (Percepción – Planificación – Actuación)

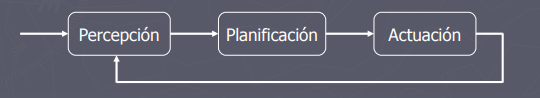


Figura 3.20. Agentes Inteligentes. Fuente. (Velasco Gutierrez, 2014)

Aplicaciones en los videojuegos:

* Comportamientos inteligentes
* Simulación de personalidades
* Exploración de mapas desconocidos
* Simulación de multitudes
  + 1. **Redes Neuronales**

Una Red de Neuronas Artificiales (RNA) es un sistema computacional que imita las capacidades de los sistemas biológicos utilizando muchos elementos simples interconectados.

**Características:**

* Clasifican patrones de entrada
* Necesitan entrenamiento supervisado con muchos ejemplos
* Son capaces de generalizar el reconocimiento de patrones
* Ante una entrada desconocida devuelve la clase más parecida
* Una vez entrenadas, funcionan en tiempo real.
  + 1. **Librerías en lenguajes de programación**

Existen muchos lenguajes que nos sirven como soporte para desarrollo de técnicas de inteligencia artificial, desde c hasta Java. Para aplicar ciertas funcionalidades “extras” contamos con lo siguiente:

* C++: cuenta con librerías para machine learning como lo sería Shark, que ofrece métodos de optimización lineal y no lineal, está basada en el aprendizaje de los algoritmos del núcleo, redes neurales y otras técnicas avanzadas de machine learning. Otra librería muy importante relacionada con este trabajo es Mlpack, cuya finalidad es ofrecer una rápida puesta en marcha de los algoritmos de machine learning, esto lo hace proveyendo a estos algoritmos una línea de código simple que facilita su integración en soluciones de mayor escala.
* Java: cuenta con una librería llamada Deeplearning4j, dedicada al Deep learning, está escrita para Java y Scala, ofrece un entorno para que los desarrolladores entrenen y elaboren modelos de IA.
* Python: cuenta con dos librerías relacionadas con el tema de este trabajo, las cuales son: Scikit-learn (enfocada totalmente al machine learning) y Tensorflow (aplicada a la investigación de redes neuronales a un nivel profundo)

Fuente: (Sancho Azcoitia, 2018)

1. **Lenguajes de programación**
   1. **Definición**

Es un lenguaje formal que permite a los programadores escribir un conjunto de comandos, acciones continuas, datos y algoritmos a través de una serie de instrucciones para crear programas que controlen el comportamiento físico y lógico de las máquinas.

Los programadores y las máquinas usan este lenguaje para comunicarse, de modo que los siguientes aspectos se pueden especificar con precisión:

* Qué datos debe ejecutar un software específico
* Cómo deben almacenarse o transmitirse dichos datos
* El software toma medidas de acuerdo con diferentes situaciones

Los lenguajes de programación son la base de todas las aplicaciones digitales que se usan todos los días, y se dividen en dos tipos principales: idiomas de bajo nivel e idiomas de alto nivel (Content, 2019).

* 1. **IDE’s**
     1. **¿Qué son?**

El IDE es un entorno de programación que se ha empaquetado como una aplicación, es decir, consta de un editor de código, compilador, depurador y generador de interfaz gráfica. El IDE puede ser una aplicación independiente o una parte de una aplicación existente.

Proporciona un marco amigable para la mayoría de los lenguajes de programación, como C ++, Python, Java, C #, Delphi, Visual Basic, etc. En algunos idiomas, el IDE puede actuar como un sistema de tiempo de ejecución, donde puede usar lenguajes de programación de forma interactiva sin tener que trabajar en archivos orientados a texto, como Smalltalk u Objective-C. El mismo IDE puede usarse con varios lenguajes de programación. En el caso de Eclipse, se puede agregar soporte de idioma adicional a través de complementos (EcuRed contributors, 2019).

Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación.

Algunas características típicas del IDE incluyen herramientas de depuración, llenado automático de ciertos elementos de código y la capacidad de detectar errores en el codo. Para proyectos que abarcan varios archivos, el IDE comprueba los archivos y ayuda a mantener la coherencia en todo el proyecto. El IDE puede facilitar la prueba de código e identificar varias partes del código para que pueda refactorizar ("ajustar" el código para que sea elegante y fácil de leer) (Luna, 2019).

* + 1. **IDE’s que soportan Java**

JAVA es un lenguaje de programación orientado a objetos creado en 1995 que actualmente pertenece a Oracle. Sin JAVA, muchas aplicaciones web no funcionarían.

Nació para convertirse en un lenguaje de programación estructurado simple que se puede ejecutar en varios sistemas operativos. En cuanto a su nombre, originalmente se llamaba Oak, pero como la marca ya estaba registrada, Java finalmente se seleccionó.

Java se usa para crear aplicaciones y procesos en varios dispositivos. Se basa en una programación orientada a objetivos, que le permite ejecutar el mismo programa en una variedad de sistemas operativos y ejecutar código de manera segura en sistemas remotos (Tokio School, s.f.).

* + - 1. **NetBeans**

Es un software que sirve como entorno de desarrollo que nos permite programar en ciertos lenguajes. Para este proyecto, NetBeans lo usaremos con el lenguaje JAVA. NetBeans es ideal para trabajar con el lenguaje de desarrollo JAVA (y todos sus derivados). La IDE de NetBeans es perfecta. Tiene un excelente balance entre una interfaz con múltiples opciones y el editor puede autocompletar nuestro código.

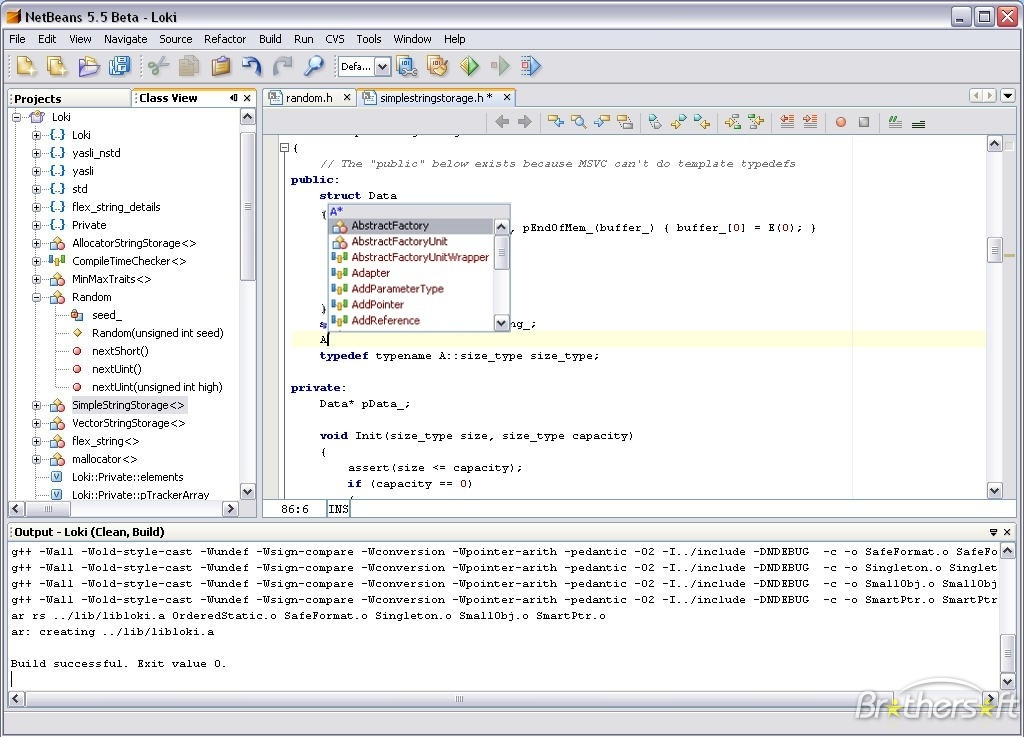


Figura 3.21. Entorno de NetBeans

Fuente: (Luis, s.f.)

* + - 1. **Eclipse**

Entorno de desarrollo integrado de código abierto multiplataforma para desarrollar proyectos. La plataforma se ha utilizado para desarrollar entornos de desarrollo integrados, como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) proporcionado como parte de Eclipse (también utilizado para desarrollar Eclipse). También se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azureus. En Eclipse, puede usar diferentes lenguajes de programación, como: Java, ANCI C, C ++, JSP, sh, perl, php, sed (EcuRed contributors, 2019).



Figura 3.22. Logo de Eclipse. Fuente. (Entre unos y ceros, 2013)

* + - 1. **IntelliJ IDEA**

Es un compilador desarrollado por la empresa JetBrains, diseñado para traducir lo escrito en lenguaje de programación (JAVA) a lenguaje de máquina. Ofrece una experiencia inteligente y ultrarrápida aportando sugerencias relevantes en cada contexto: finalización de código instantánea e inteligente, análisis del código al momento y herramientas de refactorización fiables. Este compilador cuenta con todo lo necesario para el desarrollo de aplicaciones en JAVA.

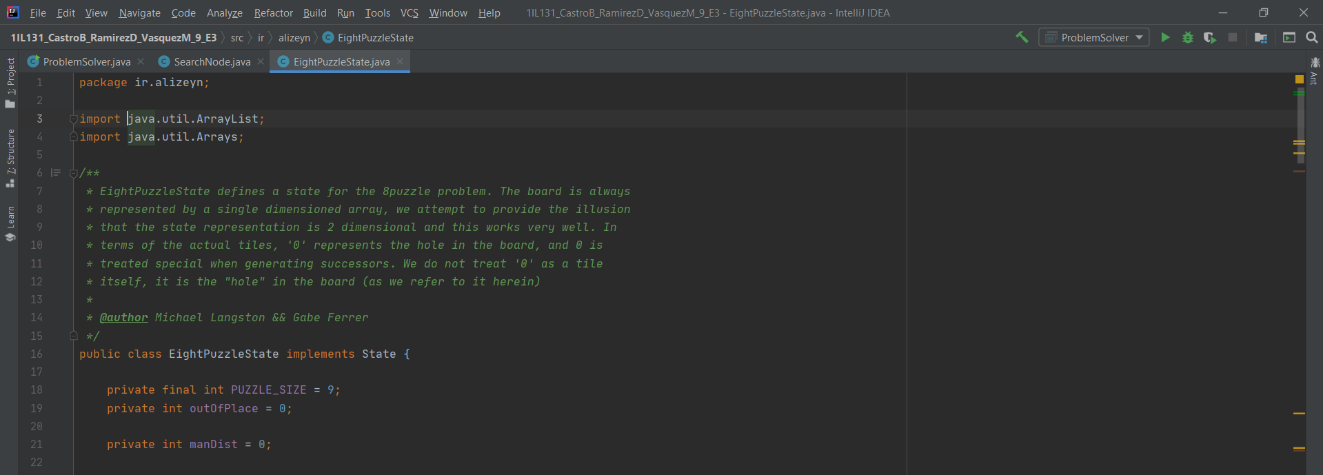


Figura 3.23. Entorno de IntelliJ IDEA. Fuente. Autoría propia

Para instalar alguno de estos IDE’s véase las referencias (Guía de instalación).

* + 1. PasteBin

Pastebin es una aplicación que permite a aquellos que la usan subir archivos de texto o ejemplos de código fuente. Esta herramienta sirve para exportar el código fuente de un programa para que otras personas puedan descargarlo. Pastebin es útil porque proporciona al usuario una forma simple y fácil de poner su código en línea para que cualquiera pueda verlo, posiblemente en un formato personalizado o un color de sintaxis para que sea fácil de leer. El usuario no necesita enviar el código a cada usuario que quiera ayudarlo. Un envío puede satisfacer a todos los posibles ayudantes, ahorrando tiempo y esfuerzo.

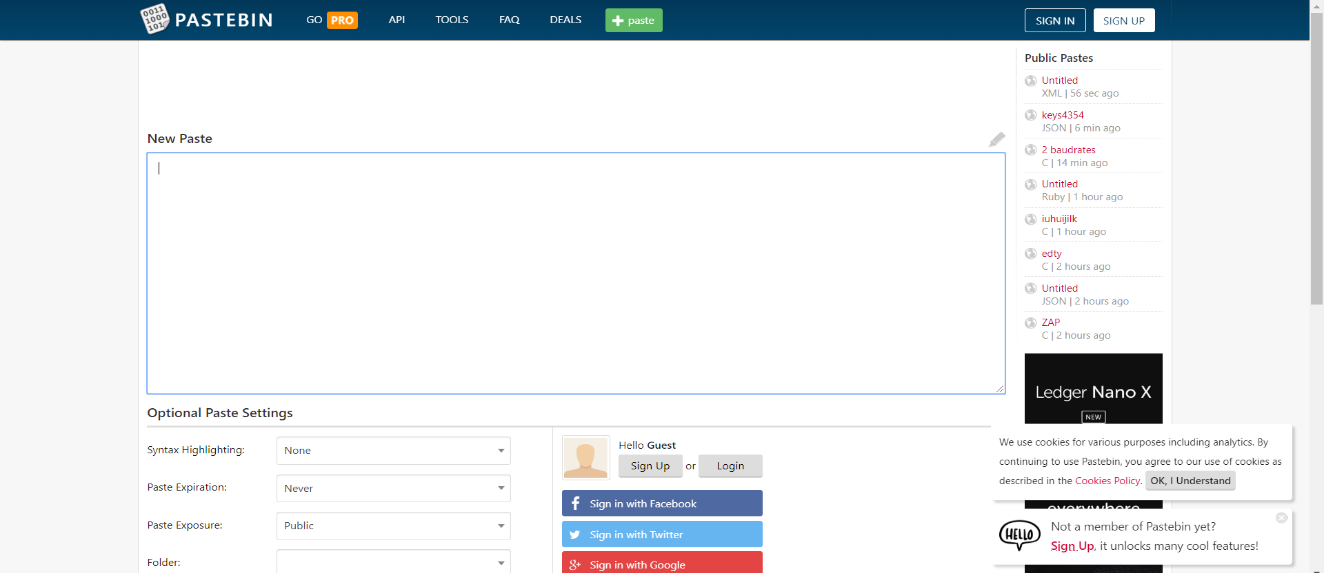


Figura 3.24. Pastebin. Fuente: Autoría propia

**Capítulo IV: El Marco Metodológico**

# **Capítulo 4: Marco Metodológico**

## **Diseño de la investigación**

### **Definición de investigación**

Debido a que se puede extraer una gran cantidad de nuevos conocimientos de un programa, este concepto de investigación es muy efectivo en el próximo trabajo. Estos nuevos conocimientos no se dirigen a la creación o elaboración del programa en sí, sino a la comparación de ambos algoritmos.

Para fines de investigación, el verbo se refiere a una acción que ejecuta una estrategia para descubrir algo. También permite mencionar conjuntos de actividades intelectuales y experimentales sistemáticas, el propósito es aumentar el conocimiento sobre un tema determinado (Porto & Gardey, 2012).

### **Tipo de investigación**

La investigación puede ser de los siguientes tipos:

1. Por la forma en que la investigación es usada: Básica o Aplicada.
2. Por el propósito del estudio: Exploratoria, Descriptiva, Explicativa o Correlacional.
3. Por la técnica de recolección de datos: Cualitativa, Cuantitativa o Mixta.
4. Por el tiempo en evaluación de la investigación: Transversal, Longitudinal o Estudio de Caso.

Para el desarrollo del proyecto, se decidió tener la siguiente estructura, como se muestra a continuación:

a) Por la forma en que la investigación es usada: “La investigación Fundamental o Básica, es la que tiene como objetivo esencial, ampliar, intensificar y aclarar los campos de la ciencia. La investigación Aplicada o Técnica tiende a la resolución de problemas o al desarrollo de ideas, dirigidas a conseguir innovaciones, mejoras de procesos o productos, etc.” (Cegarra, 2011)

El proyecto de comparación de algoritmos de búsqueda de problemas en videojuegos estratégicos está diseñado para proporcionar soluciones a los problemas generales y específicos detallados en esta sección de este documento, en esta sección, el significado del proyecto se explica de manera básica y detallada.

Al comparar dos algoritmos, uno de los propósitos es comprender qué algoritmo es mejor que el otro algoritmo, por lo que debe usarse en videojuegos estratégicos o cualquier programa basado en la toma de decisiones para ganar. Por lo tanto, este proyecto se considera una investigación de **tecnología aplicada**, es decir, la investigación realizada aquí no se conserva sólo en el texto, sino en los datos reales recopilados en el experimento.

b) Por el propósito del estudio: “Los estudios exploratorios se efectúan cuando es necesario examinar un tema o un problema de investigación poco estudiado, del cual se tienen muchas dudas o no se han abordado antes. Los estudios descriptivos buscan especificar las propiedades, las características y los aspectos importantes del fenómeno que se somete a análisis. Los estudios explicativos están dirigidos a encontrar las causas de los eventos, sucesos y fenómenos físicos o sociales. Los estudios correlacionales tienen como objetivo evaluar la relación que existe entre dos o más conceptos, categorías o variables. Tales correlaciones se expresan en Hipótesis sometidas a prueba.” (Gómez, 2006).

El objetivo de este proyecto es comparar los dos algoritmos de búsqueda previamente estudiados en diferentes partes del curso de inteligencia artificial.

A través de estos algoritmos, podemos determinar que esto no es investigación exploratoria, sino investigación experimental. La siguiente sección se llama el tipo de diseño. Por un lado, el tipo de investigación correlacional se centra en el objetivo principal del proyecto, que es la relación y la diferencia entre los dos algoritmos de búsqueda. Por lo tanto, este proyecto se considera una **investigación correlacional**.

c) Por la técnica de recolección de datos: “Por un lado la investigación cuantitativa es un método científico que conlleva la formulación hipotética, reflexión objetiva, recopilación de datos, análisis de datos y aceptación o rechazo de las hipótesis. Sin embargo, la investigación cualitativa a la investigación que produce datos descriptivos: las propias palabras de las personas y la conducta observable. Las investigaciones mixtas representan la combinación entre los enfoques cualitativos y cuantitativos”. (Antonovica, España)

Dado que se ha llevado a cabo la recopilación y el análisis de datos cualitativos y cuantitativos, se determina que este proyecto corresponde a la **investigación mixta**.

d) Por el tiempo en evaluación de la investigación: “Las investigaciones transversales o seccionales obtienen la información del objeto de estudio (población o muestra) una única vez en un momento dado. La investigación longitudinal compara datos obtenidos en diferentes oportunidades o momentos de una misma población, con el propósito de evaluar cambios. Por último, una investigación de Estudio de Casos analiza una unidad específica de un universo poblacional”. (Bernal, 2006)

Para la información proporcionada en este proyecto, se han recopilado datos de la misma población en diferentes momentos o en diferentes momentos para evaluar y determinar los cambios. Con lo cual este trabajo es **longitudinal**.

### **Tipos de diseño**

Debido a que el propósito del estudio es comparar los dos algoritmos, MiniMax y Alpha-Beta, se utiliza un diseño experimental. Se puede decir que el diseño experimental es necesario para comparar variables (como los algoritmos de búsqueda).

El proyecto necesita algunos requisitos para que este pueda ser expresado como experimental, tales requisitos son los siguientes según (Sampieri, 2010):

1. Que se pueda manipular.
2. Que se pueda medir el efecto que la variable independiente tiene en la variable dependiente.
3. Que se tenga control del experimento.

El proyecto recreado usando los algoritmos investigados tiene los requisitos anteriores y puede usarse como un prototipo para el programa creado por el grupo de trabajo (es decir, el juego Worms).

Del mismo modo, se pueden medir variables como el tiempo de compilación de cada algoritmo y su efectividad. Además, se puede controlar completamente el experimento porque es un programa de computadora.

El experimento consta de dos grupos o variables, que se utilizarán como algoritmos de búsqueda y se probarán en el desarrollo del programa de videojuegos de estrategia Worms, donde habrá diferentes variables que harán que un algoritmo sea más cómodo o mejor que otro.

## **Selección de la muestra**

### **Población**

Según Tamayo (2012) indica que el total es la suma de los fenómenos de investigación, incluida la suma de las unidades de análisis que componen el fenómeno, y que la totalidad debe cuantificarse para un estudio específico de un grupo de N entidades que contienen características específicas. Esto se denomina la población del fenómeno completo que se utiliza para constituir la investigación



Figura 4.1. Gameplay de Worms 2: Armageddon. Fuente: (Rice, 2013)

La población necesaria para el desarrollo de un programa como el juego Worms en su totalidad son todas las funcionalidades, armas, movimientos y posibilidades del Worms original.

### **Muestra**

Tamayo y Tamayo (2006), define la muestra como: "el conjunto de operaciones que se realizan para estudiar la distribución de determinados caracteres en totalidad de una población universo, o colectivo partiendo de la observación de una fracción de la población considerada".

En el caso de Palella y Martins (2008), definen la muestra como: "…una parte o el subconjunto de la población dentro de la cual deben poseer características que reproducen de la manera más exacta posible”.

La muestra en el caso del proyecto de la simulación del videojuego Worms es el uso de unas herramientas básicas (Granada, Arco, Hacha), cada una con su respectivo daño y estilo de impacto. Así mismo se limitó la movilidad por turno a 5 espacios o casillas, ya que lo que se busca es observar el comportamiento de los algoritmos en el juego y no el juego en sí.

Un rasgo que tiene la muestra es que se tenía previsto que para la partida en el juego esta no durara más de 1 minuto, y en caso de ser por victorias, el máximo serían 3.

Muestra probabilística “Se eligen en función de las características de la investigación, no dependen de la probabilidad”, como su contra parte tenemos a la muestra no probabilística o dirigida “Se eligen en forma aleatoria. Todas tienen la misma posibilidad de ser elegidas” (Sampieri, 2010).

Como se expresa en la sección de análisis del problema, el objetivo del proyecto es determinar la eficiencia de los algoritmos de búsqueda siendo comparados en la simulación del videojuego Worms.

## **Instrumentos de medición**

Los instrumentos de medición necesarios en el proyecto permiten que el entorno de desarrollo del sistema de programación (por ejemplo, el tiempo de compilación) se muestre en la consola. Los entornos de desarrollo en el lenguaje de programación JAVA que se utilizara para la recopilación de datos del programa de Worms son NetBeans y Intellij.



Figura 4.2. Logo de IntelliJ IDEA. Fuente: Autoría propia

Al generar hipótesis basadas en el análisis de resultados, la recopilación de datos en proyectos destinados a analizar y comprender algoritmos es crucial.

La recopilación de datos implica el desarrollo de un plan de procedimiento detallado, que conduce a la recopilación de datos para verificar la validez de los supuestos. Las técnicas de recolección de datos según (Arias, 2006) “Son las distintas formas o maneras de obtener información”.

(Universidad Rafael Belloso Chacín, s.f.)

Los datos que se recopilarán antes y después de la ejecución del programa son el resultado obtenido al probar ambos algoritmos enfrentados en el juego en diferentes ocasiones o partidas.

La herramienta en la cual serán evaluados los datos obtenidos en el programa de Worms es Microsoft Excel, la cual cuenta con diversas herramientas para el análisis de datos como son el cálculo, herramientas gráficas, tablas calculares y un lenguaje de programación macro llamado Visual Basic para aplicaciones. De las cuales las herramientas graficas en combinación con las tablas de cálculo nos aportaran información sobre cuál de los algoritmos de búsqueda es más eficiente en su tarea para tomar las mejores decisiones en función de la situación presente en la partida y realizar las acciones necesarias para conseguir la victoria.



Figura 4.3. Ejemplo de las herramientas graficas de Excel presentada por Microsoft. Fuente. (Grupo Marmor, 2019)

Por otra parte, los participantes o creadores del programa que simula Worms se pondrán en contacto por medio de las herramientas de comunicación WhatsApp y Microsoft Teams que ofrecen diferentes opciones para compartir datos por medio de la web.

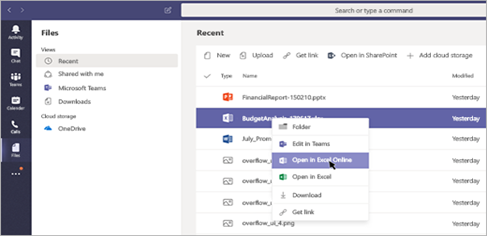


Figura 4.4. Ejemplo de cómo usar las herramientas para compartir datos en Microsoft Teams. Fuente: Autoría propia

**Capítulo V: Desarrollo de aplicaciones**

# **Capítulo 5: Desarrollo de las Aplicaciones**

## **Análisis del Problema**

Se requiere hacer un análisis comparativo entre el algoritmo de MiniMax y AlfaBeta aplicado al juego de estrategia de WORMS (1995).

Para la implementación de los algoritmos al juego, se necesitará un grafo o árbol el cual será recorrido por ambos algoritmos. Este árbol contendrá más de una solución para el estado inicial, sin embargo, se desea encontrar la mejor de ellas,

de modo que, solo se requiere obtener el siguiente nodo que lleva al mejor resultado; por lo tanto, no es necesario almacenar todo el recorrido del árbol.

Para el recorrido del árbol se tendrán cientos de miles de estados por recorrer y, para que el juego logre una fluidez de jugabilidad, el árbol recorrido debe ser limitado a una profundidad adecuada, es por ello, que se debe tener en cuenta esta información a la hora diseñar el algoritmo y de esta manera, se pueda mejorar el tiempo en el cual se encuentran los resultados.

La función heurística, para los algoritmos de MiniMax y Alpha-Beta, necesita arrojar tres posibles resultados. Positivo, que favorece a la IA; negativo, que favorece al oponente y cero, que no favorece a ninguno de los dos.

Por otra parte, para resolver el problema no es necesario recordar el pasado, porque el algoritmo se encarga de decidir los movimientos siguientes según las probabilidades de victoria que vayan surgiendo mientras se desarrolla la partida. De igual forma, el algoritmo no necesita aprender nuevas estrategias, ya que todas las posibles estrategias estarán definidas en el recorrido del árbol.

Teniendo en cuenta los puntos mencionados, se procede a desarrollar el análisis planteado.

* **Estado inicial:**

El estado inicial es un estado cualquiera de la partida, la cual está representada por la información de sus jugadores; cada jugador tiene su propia combinación de los atributos: vida, posición, turno y munición de cada arma. En el estado inicial ambos jugadores deben tener la misma vida y munición.

* **Operadores:**
  + Mirar el turno, si es mi turno.
  + Verificar todos los movimientos posibles.
  + Verificar todas las armas con los movimientos posibles.
  + Escoger la mejor combinación entre arma y movimiento que me favorezca.
  + atacar.
* **Meta:**

La meta será alcanzar un estado donde resulte beneficiada la Inteligencia Artificial.

* **Función de coste**:

La función de coste está determinada por la cantidad de estados recorridos para tomar una decisión.

g(Nodo) = estados recorridos por el Nodo

* **Función de coste heurística:**

La función de coste heurística será la diferencia de la salud entre ambos jugadores:

h(Nodo) = salud IA – salud jugador

**Espacio de estados:**

El juego dispone de 5 posiciones posibles y 3 armas disponibles; multiplicado, nos da una cantidad de 15 movimientos disponibles por nivel. Debido a que el juego solo cuenta con 6 municiones en total por todas las armas y, el árbol crece exponencialmente; el espacio de estados esperado es de:

150+151+152+153+154+155+156 = 12,204,241

* **Solución:**

La solución, para ambos algoritmos debe ser, buscar la mejor solución.

## **Diseño**

**Objetos:** jugadores con sus atributos: inventario, posición y salud.

**Estados**: Nodo que tiene la siguiente información.

**Representación del estado:**

Nodo {

int pos jugador

int pos IA

int vida jugador

int vida IA

int[] munición jugador

int[] munición IA

int turno

int peso

Lista hijos

}

**Estado Inicial:** Es estado inicial será la siguiente información para el Nodo:

pos jugador = elegida por el usuario

pos IA = elegida por la IA

vida jugador = 100

vida IA = 100

armas jugador = 1, 2, 3

armas IA = 1, 2, 3

**Estado Final:** El estado final debe ser cuando el juego termina; este termina por dos razones:

1. Cuando la vida de uno de los dos jugadores es menor o igual a cero.
2. Cuando se acaba la munición de los jugadores.

Un ejemplo de estado final donde gana la IA porque la vida del otro jugador es cero.

pos jugador = 1

pos IA = 2

vida jugador = 0

vida IA = 30

armas jugador = 0, 1, 0

armas IA = 0, 0, 1

**Estructuras a utilizar:**

Listas, para almacenar los nodos descendientes de otro nodo.

Árboles, para representar la conexión entre los estados del juego.

Arreglos, para guardar la munición de cada jugador.

y estructuras primitivas donde sea necesario.

**Diseño de clases:**

**Clase “Nodo”, sus propiedades y sus funciones:**

Propiedades

public int posp; guarda la posición del jugador

public int posIA; guarda la posición de la IA

public int vidap; guarda la salud del jugador

public int vidaIA; guarda la salud de la IA

public int[] weaponIA; guarda la munición de la IA

public int[] weaponP; guarda la munición del jugador

public int turn; i ndica de quién es el turno en un estado

public int peso; almacena el peso de un estado

Métodos

Imprimir() Imprime en consola la información de un nodo actual.

Ganador() Retorna un número que representa al jugador que ganó.

Clonar() Clona el nodo actual y lo devuelve en una nueva posición de memoria.

Evaluar() Retorna la diferencia de vida entre ambos jugadores.

GenerarHijo() guarda en la lista de hijos todas las combinaciones posibles entre movimiento y arma para efectuar el siguiente movimiento.

Game\_over() determina si el juego se acabó.

GenerarHijosinmovimiento() guarda en la lista de hijos todos los disparos posibles desde la posición actual del nodo.

**Clase “Algoritmodebusqueda”:**

Propiedades

int estadosrecorridos cuenta los estados recorridos por un nodo en el algoritmo antes de llegar a la respuesta.

static long inicio almacena el tiempo en que empieza a buscar.

static long fin almacena el tiempo en que termina de buscar.

Nodo hijo almacena el mejor estado que favorece a la IA para retornarlo.

int algoritmo recibe el algoritmo de búsqueda deseado por el usuario.

Métodos

Buscar() selecciona el algoritmo de búsqueda deseado por el usuario.

Minimax() Búsqueda heurística en profundidad que retorna el mejor movimiento para la IA.

Aldabeta() Búsqueda heurística en profundidad que retorna el mejor movimiento para la IA.

A continuación, se explicará de forma más detallada los algoritmos de Minimax() y Alfabeta()

**MiniMax()** : El algoritmo de Minimax recorrerá el árbol hasta el fondo hasta encontrar el fin de un estado sin hijos, luego según el turno del jugador escoge el nodo máximo para el jugador MAX o el nodo mínimo para el jugador MIN.

Ejemplo gráfico.

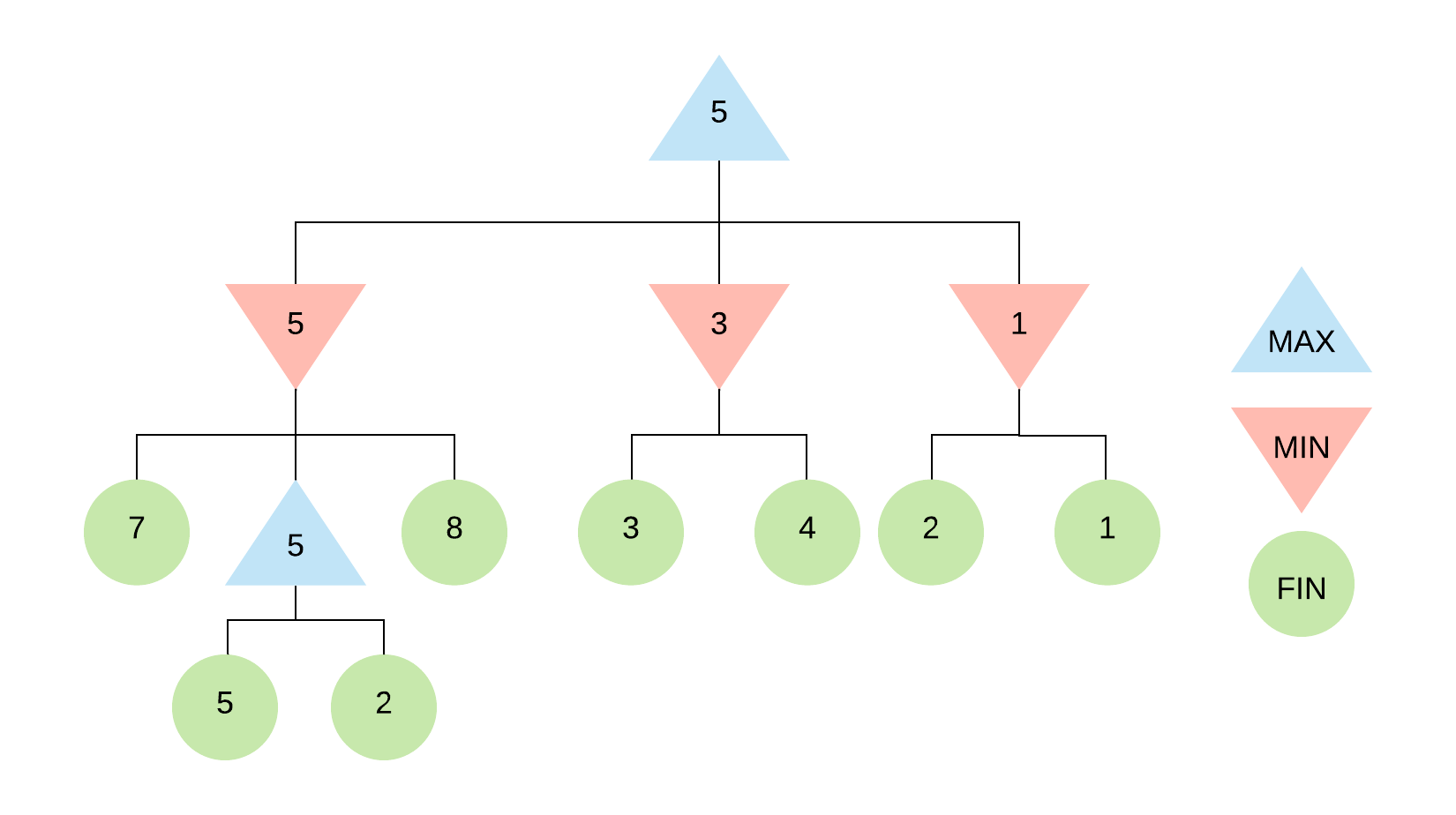


Figura 5.1 Recorrido de un Árbol con el algoritmo de MiniMax

**Alfabeta()**: En este algoritmo cada nodo se marca con dos etiquetas, alfa y beta. Alfa contendrá el mejor valor contenido por la función de evaluación en cualquiera de los descendientes de ese nodo para el jugador MAX, que es la IA.

Beta contiene el peor valor obtenido por la función de evaluación para todos los descendientes del jugador MIN, que es el humano.

Siempre que se encuentre un nodo en el que alfa es mayor o igual a beta se podrá obviar esa rama del árbol, en la figura 5.2 se puede observar cómo ocurre esta situación en el nodo “2” de la parte izquierda de la figura.

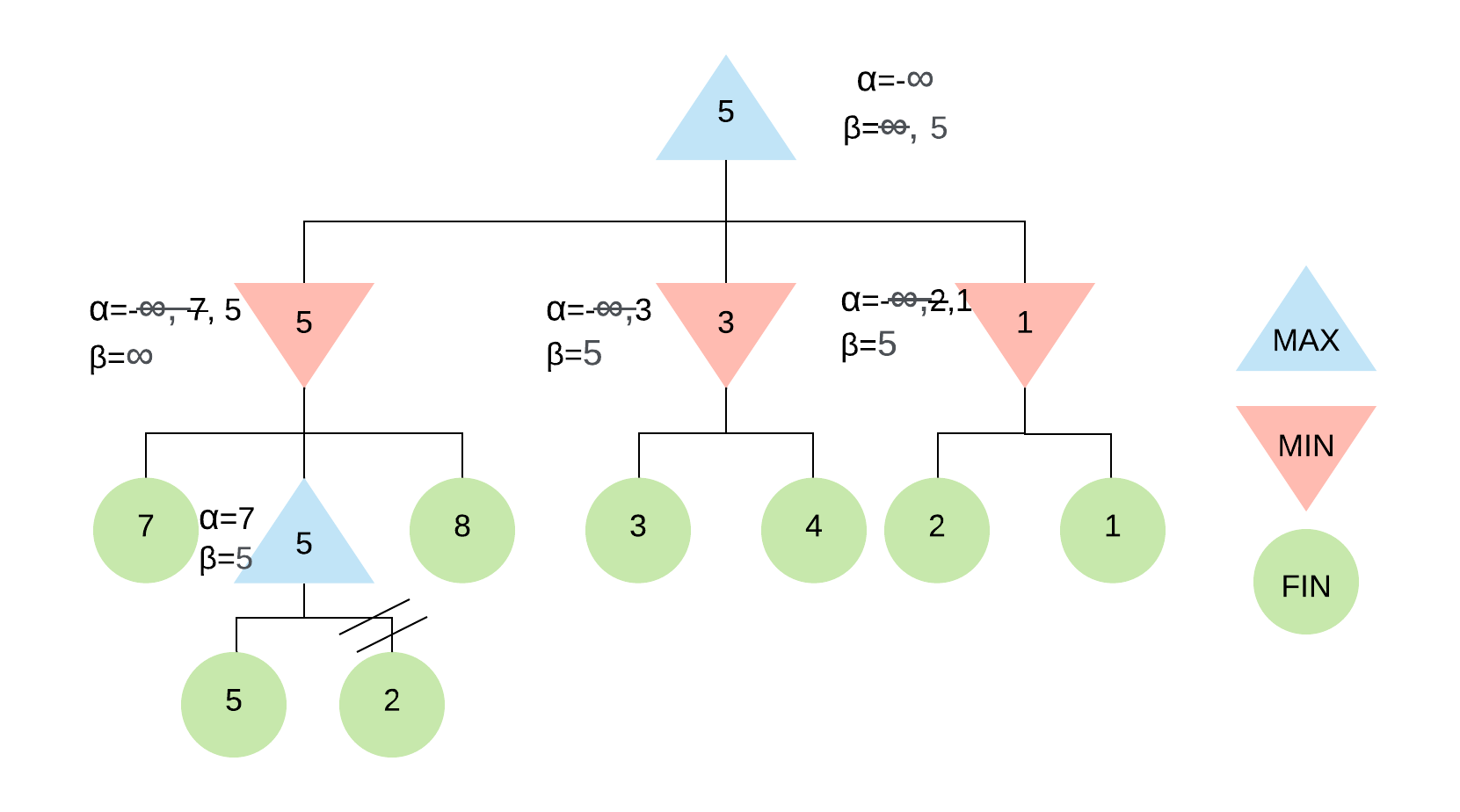


Figura 5.2 Recorrido de un Árbol con el algoritmo de Alpha-Beta

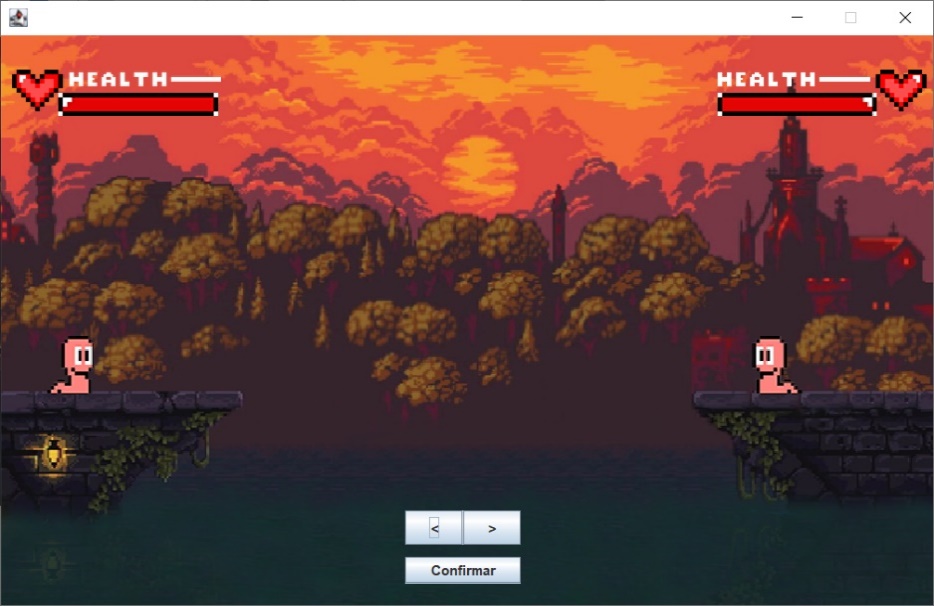
**Diseño de la interfaz**

Las condiciones que tendrá la interfaz serán:

El usuario debe ingresar el algoritmo y el nivel de dificultad antes de comenzar el juego. Para ello se implementa el diseño de la figura 5.4.

Figura 5.4 Interfaz del programa

Luego, el jugador puede realizar un movimiento antes de seleccionar su arma. El movimiento del jugador se hará a través de botones que indican la dirección. En la figura 5.5 de puede observar el diseño y la ubicación de dichos botones en la parte inferior de la pantalla.

Figura 5.5 Interfaz del programa

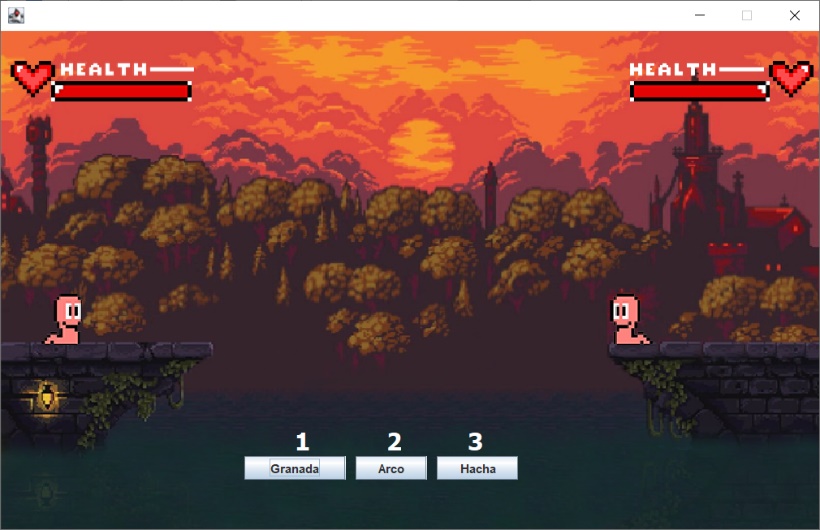
Por último, el inventario será desplegado arriba de cada botón de arma para saber la cantidad de munición disponible por cada arma. Ver ejemplo de la figura 5.6.

Cada jugador y su posición estará representado por la caricatura de un gusano.

El nivel de salud de cada jugador estará representado por la barra roja presente en la parte superior de la ventana.

La implementación de este diseño de interfaz supone que el usuario tenga una agradable experiencia al momento de utilizar el juego.

Figura 5.6 Interfaz del programa



## **Implementación en el lenguaje seleccionado**

**Partes del código más significativas:**

Algoritmo de búsqueda MiniMax

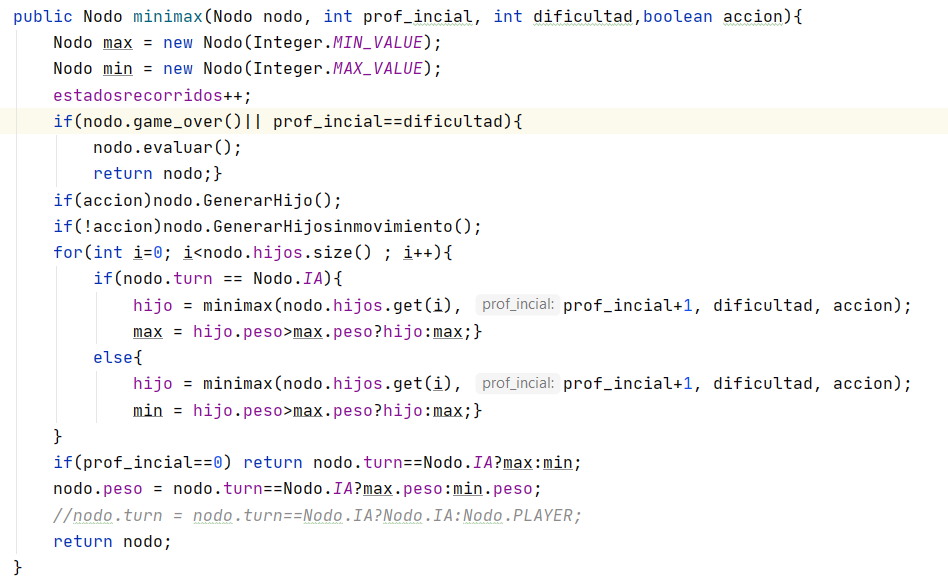


Figura 5.7 Algoritmo MiniMax en Java

El algoritmo MiniMax presentado en la figura 5.7 es recursivo, empieza recibiendo los siguientes datos: un nodo, una profundidad de cero, una dificultad que sería el límite del recorrido del árbol y una acción que puede ser moverse o disparar. Luego, inicializa dos variables de tipo Nodo, que representarán el máximo y el mínimo. Por consiguiente, evalúa si el juego se acabó o se llegó al límite preestablecido en la dificultad, si no, evalúa si el estado actual es MAX o MIN, y llama la función de MiniMax para cada uno de los hijos de este nodo, escogiendo el máximo o el mínimo según el turno del jugador. Luego retorna el mejor o peor hijo encontrado.

Algoritmo de búsqueda Alpha-Beta

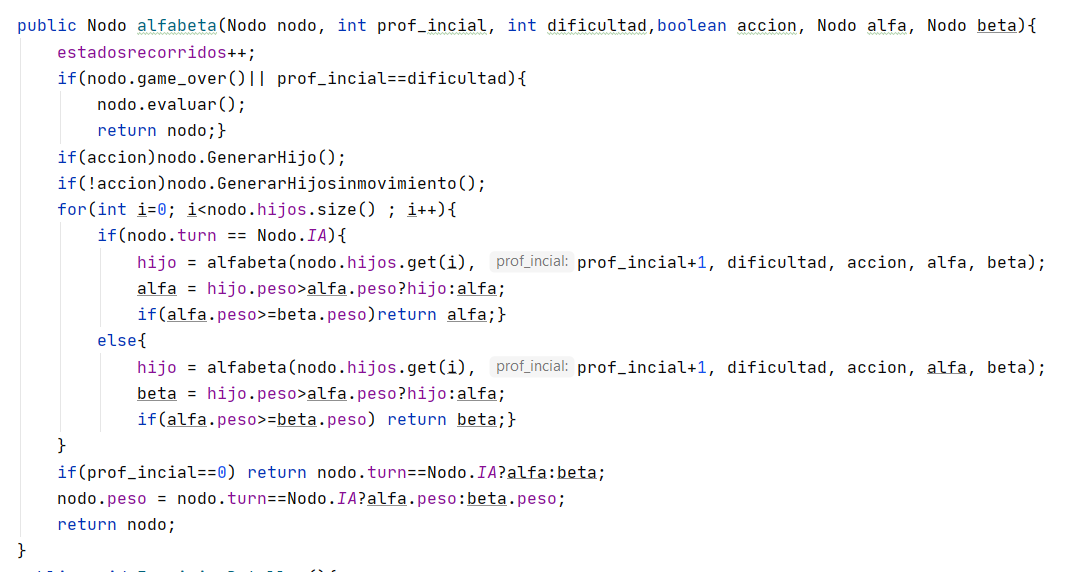


Figura 5.8 Algoritmo de Alpha-Beta

El algoritmo de la figura 5.8 al igual que el de la 5.7 es recursivo, y si se observa con detenimiento, se puede apreciar su gran similitud, esto sucede debido a que este algoritmo es la mejora del anterior. Empieza recibiendo un nodo, una profundidad de cero, una dificultad que sería el límite del recorrido del árbol, y una acción que puede ser moverse o disparar. Luego en vez de inicializar dos variables llamadas MAX y MIN, las recibe como parámetro y las llama Alfa y Beta. A continuación, evalúa si el juego finalizó o se llegó al límite preestablecido en la dificultad, si no, evalúa si el estado actual es MAX o MIN, y llama a la función de AlfaBeta para cada uno de los hijos de este nodo, escogiendo el máximo y asignándolo a Alfa o el mínimo y asignándolo a Beta, según el turno del jugador. Si en algún momento alfa es mayor o igual a Beta, retorna el nodo actual, que es el equivalente a cortar una rama del árbol. Luego retorna el mejor o peor hijo encontrado.

## **Pruebas**

Se procede a ejecutar el programa tres veces, poniendo a prueba los dos algoritmos de la figura 5.7 y 5.8 para ver cuál es el comportamiento de cada uno. Se puede apreciar cómo de la figura 5.9 a la 5.11 el algoritmo MiniMax le toma más tiempo elegir su movimiento, pero es más certero generalmente que el de AlfaBeta. Para estas pruebas, el primer movimiento es aleatorio; la desventaja de esto, es que el tiempo para ambos al inicio es de cero milisegundos y se pierde esa primera comparación, pero así se crean ventajas y desventajas iniciales.

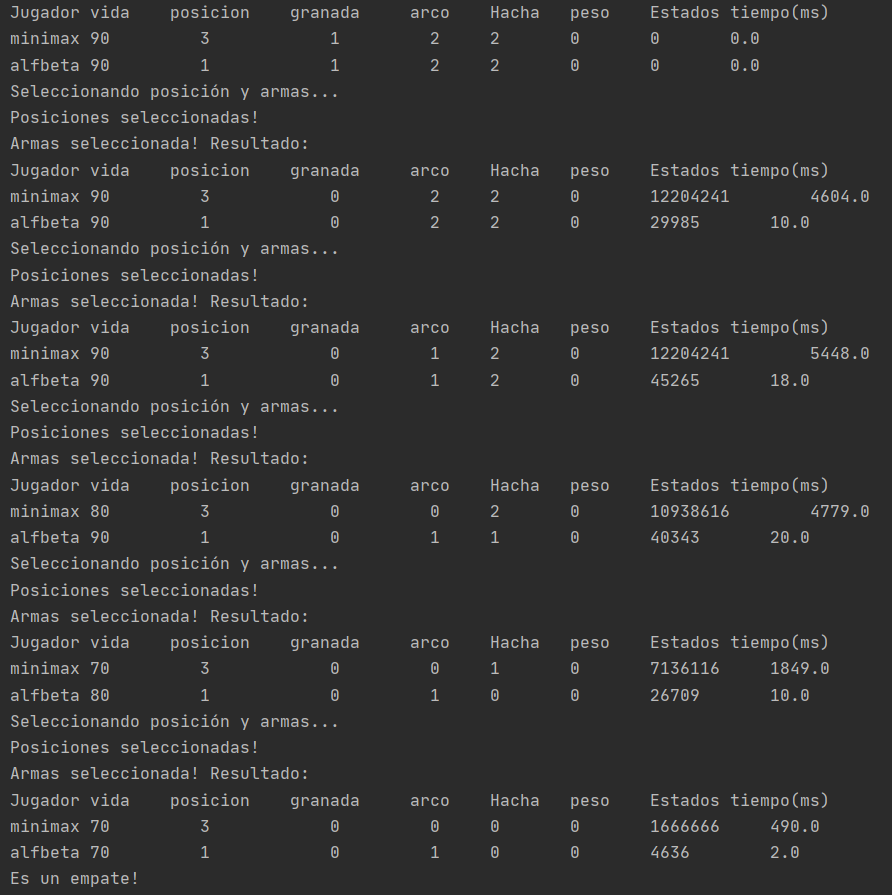
Figura 5.9 MiniMax vs AlfaBeta con una profundidad de 6 niveles

Figura 5.10 MiniMax vs AlfaBeta con una profundidad de 3 niveles

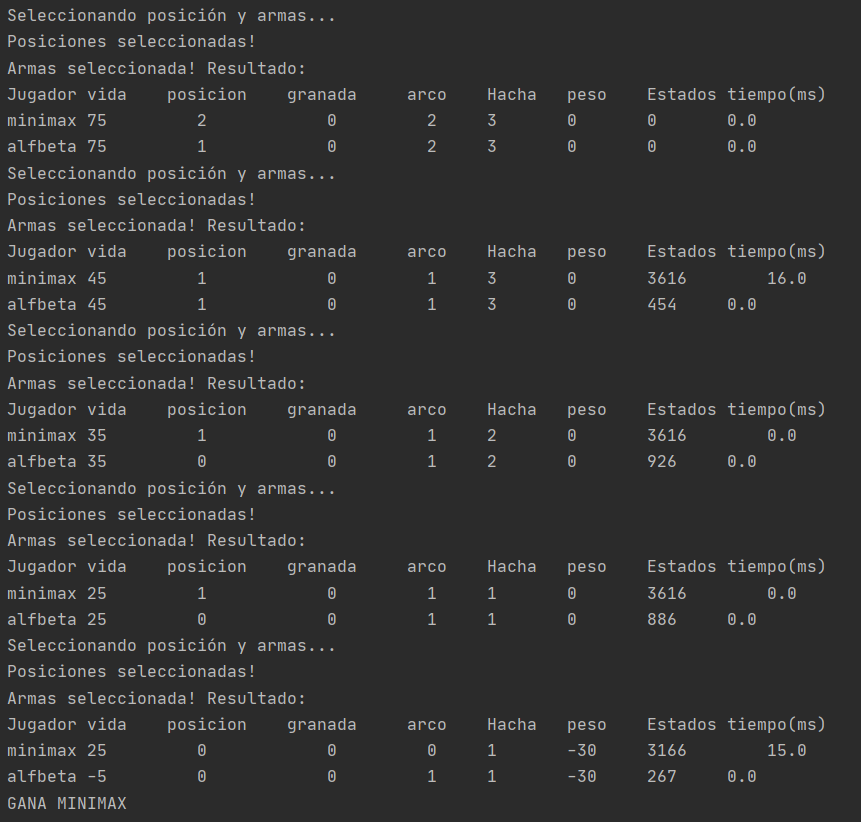


Figura 5.11 MiniMax vs AlfaBeta con una profundidad de 1 nivel



**Capítulo VI: Presentación y análisis de resultados**

# **Capítulo 6: Presentación y análisis de resultados**

## **Recolección y medición de datos**

Recolectamos los datos a través de un fragmento de código que fue creado con el propósito de recolectar los datos directamente del programa, ya que esta cuenta una variable que contiene el tiempo de ejecución del algoritmo y los estados que se van generando a lo largo del juego.

Para medir el tiempo de ejecución del algoritmo utilizamos dos variables que almacena el tiempo en el que inicia el algoritmo y el tiempo en que este finaliza al restar estos dos valores podremos obtener el tiempo de ejecución del código en milisegundos. Luego para calcular la cantidad de estados se creó otra variable que guarda esa cantidad por movimientos hasta que se termina la ejecución del programa.

Los datos fueros obtenidos de 300 pruebas de ejecución de la cual se sacaron respectivamente tiempo de ejecución del programa, estados recorridos y el resultado el juego.

## **Análisis de los datos (Modelo matemático)**

Hicimos uso de la medida de tendencia central para analizar los datos obtenidos. Obtuvimos que los dos algoritmos se ejecutan en tiempos con una diferencia abismal. De las 300 ejecuciones tenemos que el algoritmo Alfabeta supero al Minimax en 129 partidas, siendo derrotado en 61 y empatando en 110 con su rival.

Con los datos obtenidos en las partidas tenemos que el tiempo promedio del Algoritmo Alfabeta para ejecutar sus movimientos es de 2.6 ms en comparación con los 67.8 ms en promedio que tarda el Minimax.

El algoritmo Alfabeta muestra ser mucho mejor para resolver este tipo de problemas.

Tabla 6.1. Análisis de los datos

|  |  |  |
| --- | --- | --- |
| **PARTIDAS SIMULADAS** | | |
| **300** | | |
| **RESULTADOS** | | |
| **EMPATE** | MINIMAX | ALFABETA |
| **110** | 61 | 129 |

Tabla 6.2. Análisis Estadístico de los datos

|  |  |  |
| --- | --- | --- |
| **Análisis** | **Estados Rec. Minimax** | **Estados Rec. Alfabeta** |
| **moda** | 813616 | 26532 |
| **media** | 621976 | 14414.72933 |
| **mediana** | 813616 | 11672 |
|  |  |  |

Tabla 6.3. Análisis Estadístico de los datos

|  |  |  |
| --- | --- | --- |
| **Análisis** | **Tiempo (ms). Minimax** | **Tiempo (ms). Alfabeta** |
| **moda** | 13 | 1 |
| **media** | 67.81266667 | 2.365333333 |
| **mediana** | 69 | 2 |

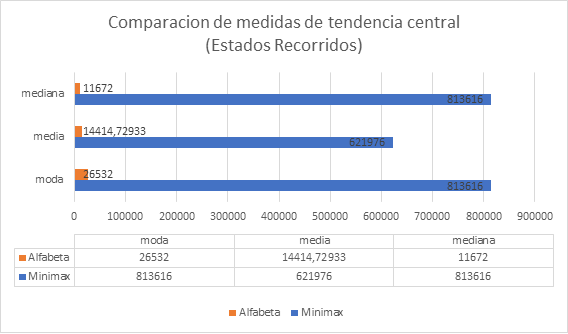


Figura 6.1. Comparación de medidas de tendencia central

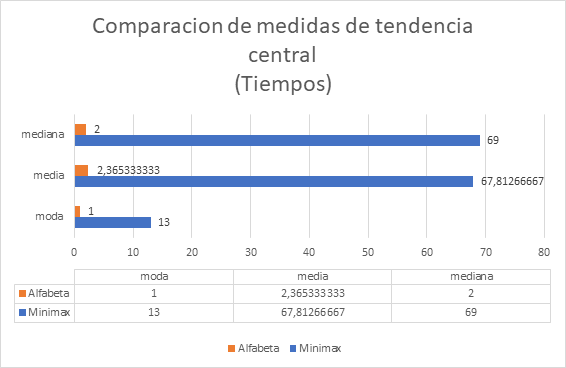


Figura 6.2. Comparación de medidas de tendencia central

## **Análisis de estadística descriptiva**

En las gráficas de tiempos de ejecución por movimientos podemos apreciar la enorme diferencia que existe entre los algoritmos de Minimax y Alfabeta. El algoritmo Alfabeta demuestra ser más rápido a la hora de encontrar una solución al problema que se le plantea en el juego por ende el algoritmo Alfabeta como se puede ver en la gráfica de Estados recorridos genera menos de estos estados lo que lo hace más eficiente, recorriendo en promedio 14414.7 a diferencia del Minimax que recorren en promedio 621976 Estados.

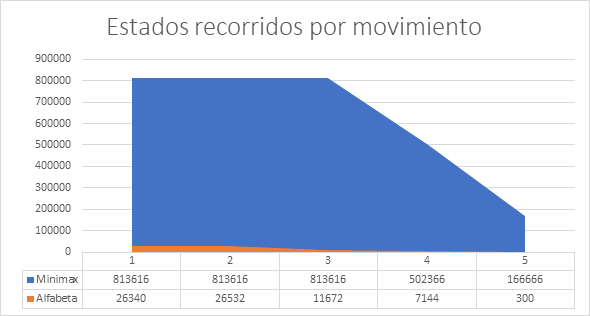


Figura 6.3. Comparación de Estados recorridos en ejecución de una partida

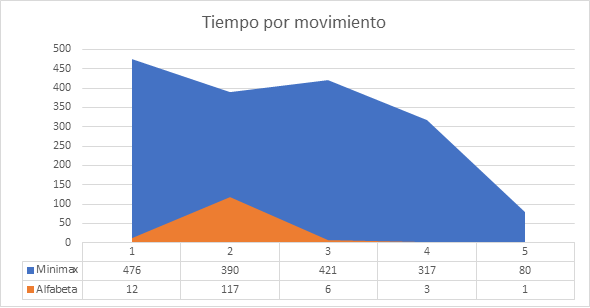


Figura 6.4. Comparación de tiempo en ejecución de una partida

## **Análisis de la confiabilidad de los resultados**

Haciendo Uso de las fórmulas de varianza y de desviación estándar no podemos hacer una idea de que tan confiables puede ser el rendimiento de los algoritmos.

Tabla 6.4. Confiabilidad de los resultados

|  |  |  |
| --- | --- | --- |
| **Análisis** | **Estados Rec. Minimax** | **Estados Rec. Alfabeta** |
| **Varianza** | 66358283400 | 110146529 |
| **desviación estándar** | 257601.0159 | 10495.0716 |

|  |  |  |
| --- | --- | --- |
| **Análisis** | **Tiempo (ms). Minimax** | **Tiempo (ms). Alfabeta** |
| **Varianza** | 1588.99357 | 28.4051982 |
| **desviación estándar** | 39.8621822 | 5.32965273 |

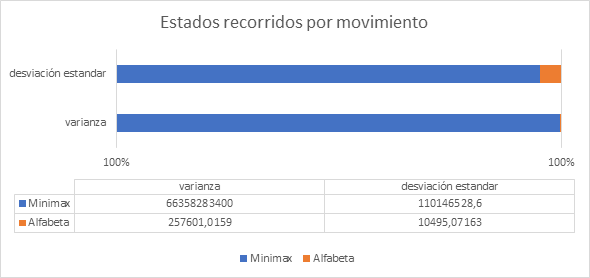


Figura 6.5. Comparación de confiabilidad

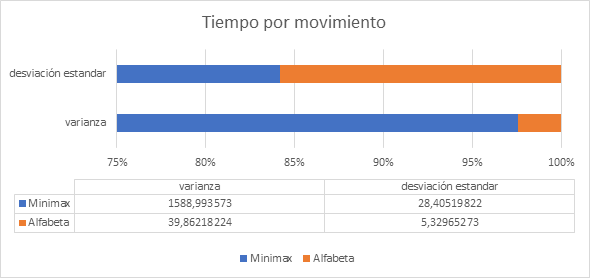


Figura 6.6. Comparación de confiabilidad

En la Figura 6.6 se pueda apreciar que la varianza y la desviación estándar de el algoritmo Minimax son mayores a los del algoritmo en Alfabeta, esto nos indica que el tiempo y los estados recorridos tienden a estar dentro de un margen de variación mayor que los del algoritmo Alfabeta, Lo que revela que el algoritmo Alfabeta es más estable que el Minimax al tener tiempos más corto

# **Conclusiones**

**Conclusiones**

En función a las investigaciones y pruebas aquí desarrolladas se obtuvo como resultado que el algoritmo Alfa – Beta es muchísimo mejor que el algoritmo MiniMax ya que demora mucho menos en hacer el recorrido en los nodos con lo cual es más efectivo en la toma de decisiones, y también nos sorprendió que sea el que menos dispersión posee.

La implementación de estos algoritmos en otros videojuegos de estrategia, lo podemos apreciar en otros de esta área como Mario + Rabbits Kingdom Battle, Starcraft o Age of Empires. Su funcionalidad es controlar la capacidad de decidir o pensar para escoger mecanismos de los llamados “NPCs”, en los videojuegos de manera casi instantánea.

De las pruebas o experimentos realizados a ambos algoritmos, se apreció como el Alfa – Beta venció en todos los aspectos al MiniMax, es por eso por lo que, para la implementación de toma de decisiones en juegos de estrategia, el algoritmo de Alfa – Beta es de los más reconocidos y muy recomendable su aplicación. Es por ello que, en juegos por turnos para dos jugadores, se recomienda con toda seguridad la aplicación de este algoritmo para el desarrollo de la Inteligencia Artificial.

Es así como podemos concluir que sí se cumplió la hipótesis planteada al inicio de este documento, ya que se esperaba que el algoritmo de Alfa – Beta fuera más efectivo que el MiniMax; como se desarrolló previamente, este algoritmo recorre los espacios de una manera mucho más reducida y efectiva que el MiniMax.

Gracias a la realización del proyecto 1 con anterioridad, se contaba con una base sobre la cuál trabajar; esto nos sirvió de guía para resolver el desarrollo de este proyecto 2 de una forma más precisa y clara. En un principio, el hacer este documento fue una experiencia algo acelerada ya que se nos asignó este proyecto para una semana después del primero; sin embargo, fue realizado de forma mucho más fluida ya que pasamos por la experiencia del primer proyecto y teníamos nuestras divisiones claras.

Hemos podido apreciar como la Inteligencia Artificial está siendo importante en la época actual en la que estamos. La aplicación de los algoritmos de búsqueda aprendidos en clases en algo tan común como un texto predictivo o un videojuego de estrategia, el cual nos permitió ver la gran cantidad de cosas que se pueden hacer.

Una de las complicaciones que se presentó a lo largo de la realización de este trabajo fue la creación de los estados y la representación del videojuego Worms, lo bueno es que, a diferencia del anterior proyecto, se logró manejar la recursividad de Java de manera correcta para que el programa no se quedara sin memoria en la aplicación de Worms, con lo que se pudo llevar a cabo un juego fluido sin mayores dificultades.

Uno de los hallazgos que se tuvo como interesante fue que los algoritmos Minimax y Alfa-Beta se utilizan mucho en el ámbito de los videojuegos, un caso muy específico es el “modo carrera” de FIFA, en donde la IA debe ser efectiva dependiendo del nivel que se le asigne.

En otros videojuegos en donde se aprecian algoritmos para la toma de decisiones de forma eficiente en función de las situaciones, son aquellos de tipo RPG, donde en las batallas o enfrentamientos la inteligencia artificial enemiga del juego toma decisiones, como atacar, curarse, realizar algún movimiento especial, etc. Uno muy característico es Mario y Luigi: Viaje al Centro de Bowser, así como los juegos de la franquicia Mario Party.

Con el desarrollo llevado a cabo en ambos proyectos podemos afirmar y estar seguros de que, a través de los avances de la Inteligencia Artificial y su buena implementación, la calidad de vida del ser humano mejorará y permitirá seguir descubriendo constantemente muchas posibilidades.

# **Referencias**

SALESFORCE LATINOAMÉRICA. (22 de junio de 2017). *¿Qué es la inteligencia artificial?* Obtenido de SalesForce Blog: https://www.salesforce.com/mx/blog/2017/6/Que-es-la-inteligencia-artificial.html

Anzil, F. (8 de septiembre de 2006). *Teoría de Juegos*. Obtenido de EconoLink: https://www.econlink.com.ar/definicion/teoriadejuegos.shtml

Baier Aranda, J. (s.f.). *Backtracking.* Obtenido de jabaier sitios: http://jabaier.sitios.ing.uc.cl/iic2552/backtracking.pdf

Béliz Osorio, N. (2020). *Ascenso de la colina.* Obtenido de eCampus UTP: https://ecampus.utp.ac.pa/moodle/pluginfile.php/228082/mod\_resource/content/6/undefined/02\_Mod2\_IA\_2020\_busqueda.colina.pdf

Béliz Osorio, N. (2020). *Búsqueda Primero el mejor.* Obtenido de eCampus UTP: https://ecampus.utp.ac.pa/moodle/pluginfile.php/228021/mod\_resource/content/4/02\_Mod2\_IA\_2020\_busqueda\_1roMejor.pdf

Béliz, N. (2020). *8452:Inteligencia Artificial 1IL-131 (Béliz Osorio).* Obtenido de eCampus UTP: https://ecampus.utp.ac.pa/moodle/pluginfile.php/220531/mod\_resource/content/1/02\_Mod2\_1\_IA\_2020\_busqueda.key.pdf

Blancarte, O. (22 de agosto de 2014). *Estructura de datos – Árboles*. Obtenido de Oscar Blancarte, Software architect: https://www.oscarblancarteblog.com/2014/08/22/estructura-de-datos-arboles/

*Búsqueda por Amplitud Iterativa.* (1 de marzo de 2017). Obtenido de Inteligencia Artificial, trabajos, actividades entre otros: https://inteligenciartificialjjmc.wordpress.com/2017/03/01/busqueda-por-amplitud-iterativa/

Calderón de la Cruz, I. (3 de octubre de 2014). *Busqueda por profundidad iterativa.* Obtenido de Slideshare: https://es.slideshare.net/icalderond/busqueda-por-profundidad-iterativa

Cardescu web. (12 de mayo de 2016). *Juegos de guerra: Inteligencia Artificial y Teoría de juegos*. Obtenido de https://cardescu.es/juegos-de-guerra-parte-i-inteligencia-articifial/

Ceccaroni, L. (abril de 2007). *Inteligencia Artificial. Búsqueda informada y exploración.* Obtenido de https://www.cs.upc.edu/~luigi/II/IA-2007-fall/2b-busqueda-informada-y-exploracion-(es).pdf

Cenamor, I. (2 de mayo de 2019). *Inteligencia Artificial: Algoritmos de Búsquedas- Isa Star*. Obtenido de Isabel Cenamor: https://icenamor.github.io/posts/2019/5/blog-post-3/#:~:text=Los%20algoritmos%20de%20b%C3%BAsqueda%20es,como%20A\*%20entre%20muchas%20otras.

Código Inicial. (27 de octubre de 2013). *Cola Circular*. Obtenido de Código Inicial : http://codigoinicial.blogspot.com/2013/10/cola-circular.html

Content, R. R. (20 de abril de 2019). *¿Qué es un lenguaje de programación y qué tipos existen?* Obtenido de https://rockcontent.com/es/blog/que-es-un-lenguaje-de-programacion/#:~:text=%C2%BFQu%C3%A9%20es%20un%20lenguaje%20de%20programaci%C3%B3n%3F,y%20l%C3%B3gico%20de%20una%20m%C3%A1quina.

Ecured Contributors. (25 de abril de 2014). *Pilas*. Obtenido de Ecured: https://www.ecured.cu/Pila\_(Estructura\_de\_datos)

EcuRed contributors. (15 de julio de 2019). *Algoritmo de Búsqueda Heurística A\*.* Obtenido de EcuRed: https://www.ecured.cu/index.php?title=Algoritmo\_de\_B%C3%BAsqueda\_Heur%C3%ADstica\_A\*&oldid=3458260

EcuRed contributors. (14 de agosto de 2019). *IDE de Programación*. Obtenido de EcuRed: https://www.ecured.cu/IDE\_de\_Programaci%C3%B3n

Entre unos y ceros. (12 de agosto de 2013). *Cómo instalar Eclipse IDE en Ubuntu 12.04*. Obtenido de https://entreunosyceros.net/como-instalar-eclipse-ide-ubuntu-1204/

Fruto, P. A. (15 de junio de 2008). *IA – Algoritmos de Juegos .* Obtenido de Universidad politécnica de Cataluña: https://www.cs.upc.edu/~bejar/ia/material/trabajos/Algoritmos\_Juegos.pdf

Gadget - Info. (2019). *Diferencia entre la cola lineal y la cola circular*. Obtenido de Gadget-info.com: https://es.gadget-info.com/difference-between-linear-queue

González, C. G. (s.f.). *Búsqueda heurística.* Obtenido de Monografías.com: https://www.monografias.com/trabajos75/busqueda-heuristica/busqueda-heuristica.shtml

Hasif. (19 de enero de 2016). *Pilas en c ejemplos*. Obtenido de http://pumpstubos.blogspot.com/2016/01/pilas-en-c-ejemplos.html

ITCL Noticias. (4 de septiembre de 2019). *Presente y futuro de la Inteligencia Artificial*. Obtenido de ITCL: https://itcl.es/blog/presente-y-futuro-de-la-inteligencia-artificial/

Lipschutz, S. (s.f.). *“GRAFOS”*. Obtenido de INSTITUTO TECNOLÓGICO DE NUEVO: http://www.itnuevolaredo.edu.mx/takeyas/apuntes/Estructura%20de%20Datos/Apuntes/grafos/Apuntes\_Grafos.pdf

López Takeyas, B. (s.f.). *Hill Climbing.* Obtenido de http://www.itnuevolaredo.edu.mx/takeyas/Apuntes/Inteligencia%20Artificial/Apuntes/IA/Hill-Climbing.pdf

Luis. (s.f.). *SOPORTE TECNICO*. Obtenido de WordPress: https://luisthebigboss.wordpress.com/ide/

Luna, E. (2019). *¿Qué son los IDEs y los editores de texto?* Obtenido de Platzi: https://platzi.com/blog/que-son-los-ides-y-los-editores-de-texto/

Martinez, D. (28 de mayo de 2016). *PRIMERO EN PROFUNDIDAD.* Obtenido de APRENDE Y PROGRAMA: https://aprendeyprogramablog.wordpress.com/2016/05/28/primero-en-profundidad/

Miguel. (s.f.). *que son las colas en estructuras de datos*. Obtenido de Blogger: https://sites.google.com/site/miguelestructura/que-son-las-colas-en-la-estructura-de-datos

Millán Villalba, M. C., Pérez Cortés, A. L., & Ruiz Duarte, J. M. (23 de enero de 2013). *Minimax y Poda Alfa-beta.* Obtenido de Prezi: https://prezi.com/okhoygagvjz6/minimax-y-poda-alfa-beta/

Pascual, J. A. (2019). *Inteligencia artificial: qué es, cómo funciona y para qué se está utilizando*. Obtenido de https://computerhoy.com/reportajes/tecnologia/inteligencia-artificial-469917

Plasencia Prado, C. E. (2016). *El algoritmo Minimax y su aplicación en un juego.* Obtenido de DevCode: https://devcode.la/tutoriales/algoritmo-minimax/#:~:text=El%20algoritmo%20de%20minimax%20en,recorre%20todo%20el%20%C3%A1rbol%20de

Programacion facil estructura de datos. (s.f.). *Colas en JAVA*. Obtenido de Programacion facil estructura de datos.blogspot: https://www.google.com/url?sa=i&url=http%3A%2F%2Fprogramacionfacilestructuradedatos.blogspot.com%2Fp%2Fcolas-en-java.html&psig=AOvVaw2mCeRZ8DcGmvEsShjTGxY4&ust=1592311770401000&source=images&cd=vfe&ved=0CA0QjhxqFwoTCICy3uPtg-oCFQAAAAAdAAAAABAT

Real Academia Española. (2019). *Inteligencia*. Obtenido de Real Academia Española: https://dle.rae.es/inteligencia

Rice, J. (5 de abril de 2013). *Worms 2*. Obtenido de Android Police: https://www.androidpolice.com/2013/04/05/new-game-worms-2-armageddon-invades-canada-and-only-canada-with-classic-ballistic-gameplay/

Ríos, T. (8 de agosto de 2012). *Diario de desarrollo de 'End of Nations': héroes y mercenariosAnunciado oficialmente 'GRID 2'*. Obtenido de Juegos.es: http://juegos.es/aventura-plataformas/worms-abre-la-beta-cerrada-en-su-juego-en-el-facebook

Rossel, G. (s.f.). *Inteligencia Computacional, Inteligencia Artificial Explicad.* Obtenido de SmartComputing: http://smartcomputing.gerardorossel.org/busqueda-a-ciegas.aspx

Sancho Azcoitia, S. (24 de abril de 2018). *Plataformas y librerías para comenzar en el mundo del Machine Learning*. Obtenido de Think Big: https://empresas.blogthinkbig.com/plataformas-y-librerias-machine-learning/

SAS Corporation. (s.f.). *Inteligencia artificial, Qué es y por qué es importante*. Obtenido de SAS: https://www.sas.com/es\_cl/insights/analytics/what-is-artificial-intelligence.html

Takeyas, B. L. (s.f.). *Poda Alfa-Beta .* Obtenido de Instituto Técnico Nuevo Laredo: http://www.itnuevolaredo.edu.mx/takeyas/Apuntes/Inteligencia%20Artificial/Apuntes/IA/Alfa-Beta.pdf

Universidad de Chile. (s.f.). *Tipos de datos abstractos*. Obtenido de https://users.dcc.uchile.cl/~bebustos/apuntes/cc30a/TDA/

Universidad Pedagógica y Tecnológica de Colombia. (2014). *Teoría de juegos en Inteligencia Artificial.* Obtenido de Studocu: https://www.studocu.com/co/document/universidad-pedagogica-y-tecnologica-de-colombia/inteligencia-computacional/apuntes/teoria-de-juegos-ia-na/5073011/view

Vargas Govea, B. (21 de agosto de 2012). *Búsqueda local: hill-climbing.* Obtenido de http://blancavg.com/tc3023ic/ic5.pdf

Velasco Gutierrez, J. D. (2014). Aplicación de algoritmos de búsqueda en los videojuegos. 6.

Villacis, J. (26 de noviembre de 2013). *Búsqueda por profundidad iterativa.* Obtenido de Prezi: https://prezi.com/awxsbnc\_3dbt/busqueda-por-profundidad-iterativa/

Vitriago, M. (26 de enero de 2015). *Grafos (Estructura de Datos)*. Obtenido de grafosestructuradedatos.blogspot: http://grafosestructuradedatos.blogspot.com/

Wikipedia, La enciclopedia libre. (27 de mayo de 2020). *Worms (serie)*. Obtenido de Wikipedia: https://es.wikipedia.org/w/index.php?title=Worms\_(serie)&oldid=126420952

Zonared. (s.f.). *Saga 'Worms'*. Obtenido de Zonared: https://www.zonared.com/juegos/sagas/worms/

# **Anexos**

# **Pruebas realizadas al programa**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **ANÁLISIS POR MOVIMIENTO** | | | |
|  | **MINIMAX** | | **ALFABETA** | |
|  | Estados Rec. | Tiempo (ms) | Estados Rec. | Tiempo (ms) |
| 1 | 813616 | 476 | 26340 | 12 |
| 2 | 813616 | 390 | 26532 | 117 |
| 3 | 813616 | 421 | 11672 | 6 |
| 4 | 502366 | 317 | 7144 | 3 |
| 5 | 166666 | 80 | 300 | 1 |
| 6 | 813616 | 315 | 26340 | 14 |
| 7 | 813616 | 161 | 26532 | 3 |
| 8 | 813616 | 199 | 11672 | 3 |
| 9 | 502366 | 124 | 7144 | 1 |
| 10 | 166666 | 15 | 300 | 1 |
| 11 | 813616 | 109 | 26340 | 3 |
| 12 | 813616 | 145 | 26532 | 4 |
| 13 | 813616 | 65 | 11672 | 2 |
| 14 | 502366 | 243 | 7144 | 1 |
| 15 | 166666 | 15 | 300 | 0 |
| 16 | 813616 | 71 | 26340 | 3 |
| 17 | 813616 | 243 | 26532 | 7 |
| 18 | 813616 | 417 | 11672 | 2 |
| 19 | 502366 | 59 | 7144 | 2 |
| 20 | 166666 | 112 | 300 | 0 |
| 21 | 813616 | 78 | 26783 | 5 |
| 22 | 813616 | 142 | 26532 | 3 |
| 23 | 813616 | 77 | 11672 | 2 |
| 24 | 502366 | 132 | 7144 | 3 |
| 25 | 166666 | 38 | 300 | 0 |
| 26 | 813616 | 106 | 26783 | 3 |
| 27 | 813616 | 193 | 26532 | 3 |
| 28 | 813616 | 90 | 11672 | 3 |
| 29 | 502366 | 43 | 7144 | 1 |
| 30 | 166666 | 98 | 300 | 0 |
| 31 | 813616 | 66 | 26340 | 4 |
| 32 | 813616 | 101 | 26532 | 3 |
| 33 | 813616 | 140 | 11672 | 2 |
| 34 | 502366 | 47 | 7144 | 1 |
| 35 | 166666 | 14 | 300 | 0 |
| 36 | 813616 | 66 | 26340 | 3 |
| 37 | 813616 | 85 | 26532 | 5 |
| 38 | 813616 | 102 | 11672 | 3 |
| 39 | 502366 | 65 | 7144 | 2 |
| 40 | 166666 | 18 | 300 | 0 |
| 41 | 813616 | 123 | 26340 | 4 |
| 42 | 813616 | 151 | 26532 | 4 |
| 43 | 813616 | 80 | 11672 | 1 |
| 44 | 502366 | 62 | 7144 | 1 |
| 45 | 166666 | 17 | 300 | 0 |
| 46 | 813616 | 87 | 26340 | 3 |
| 47 | 813616 | 81 | 26532 | 118 |
| 48 | 813616 | 118 | 11672 | 3 |
| 49 | 502366 | 54 | 7144 | 1 |
| 50 | 166666 | 20 | 300 | 0 |
| 51 | 813616 | 129 | 26340 | 4 |
| 52 | 813616 | 81 | 26532 | 3 |
| 53 | 813616 | 81 | 11672 | 2 |
| 54 | 502366 | 41 | 7144 | 1 |
| 55 | 166666 | 16 | 300 | 0 |
| 56 | 813616 | 93 | 26783 | 3 |
| 57 | 813616 | 79 | 26532 | 4 |
| 58 | 813616 | 89 | 11672 | 2 |
| 59 | 502366 | 105 | 7144 | 1 |
| 60 | 166666 | 16 | 300 | 0 |
| 61 | 813616 | 75 | 26340 | 4 |
| 62 | 813616 | 96 | 26532 | 5 |
| 63 | 813616 | 99 | 11672 | 2 |
| 64 | 502366 | 51 | 7144 | 1 |
| 65 | 166666 | 16 | 300 | 0 |
| 66 | 813616 | 77 | 26340 | 4 |
| 67 | 813616 | 99 | 26532 | 8 |
| 68 | 813616 | 104 | 11672 | 2 |
| 69 | 502366 | 39 | 7144 | 1 |
| 70 | 166666 | 19 | 300 | 0 |
| 71 | 813616 | 81 | 26340 | 5 |
| 72 | 813616 | 93 | 26532 | 3 |
| 73 | 813616 | 174 | 11672 | 1 |
| 74 | 502366 | 46 | 7144 | 1 |
| 75 | 166666 | 13 | 300 | 1 |
| 76 | 813616 | 70 | 26340 | 3 |
| 77 | 813616 | 68 | 26532 | 5 |
| 78 | 813616 | 89 | 11672 | 2 |
| 79 | 502366 | 49 | 7144 | 1 |
| 80 | 166666 | 13 | 300 | 0 |
| 81 | 813616 | 150 | 26340 | 5 |
| 82 | 813616 | 80 | 26532 | 3 |
| 83 | 813616 | 76 | 11672 | 2 |
| 84 | 502366 | 49 | 7144 | 2 |
| 85 | 166666 | 16 | 300 | 0 |
| 86 | 813616 | 88 | 26340 | 3 |

Esta tabla fue la que se usó para hacer el análisis estadístico, pero consideramos que no era necesario ponerla en ese capítulo, por lo cual decidimos ponerlo en los anexos.

# **Guía de instalación de NetBeans**

Para instalar NetBeans sin ningún inconveniente seleccione el siguiente link: <https://www.locurainformaticadigital.com/2019/07/21/descargar-e-instalar-netbeans-11-windows/>

Enlace de descarga del JDK: <https://www.oracle.com/java/technologies/javase/jdk14-archive-downloads.html>

Selecciona el sistema operativo de su computadora y si quiere descargar en archivo comprimido o el .exe

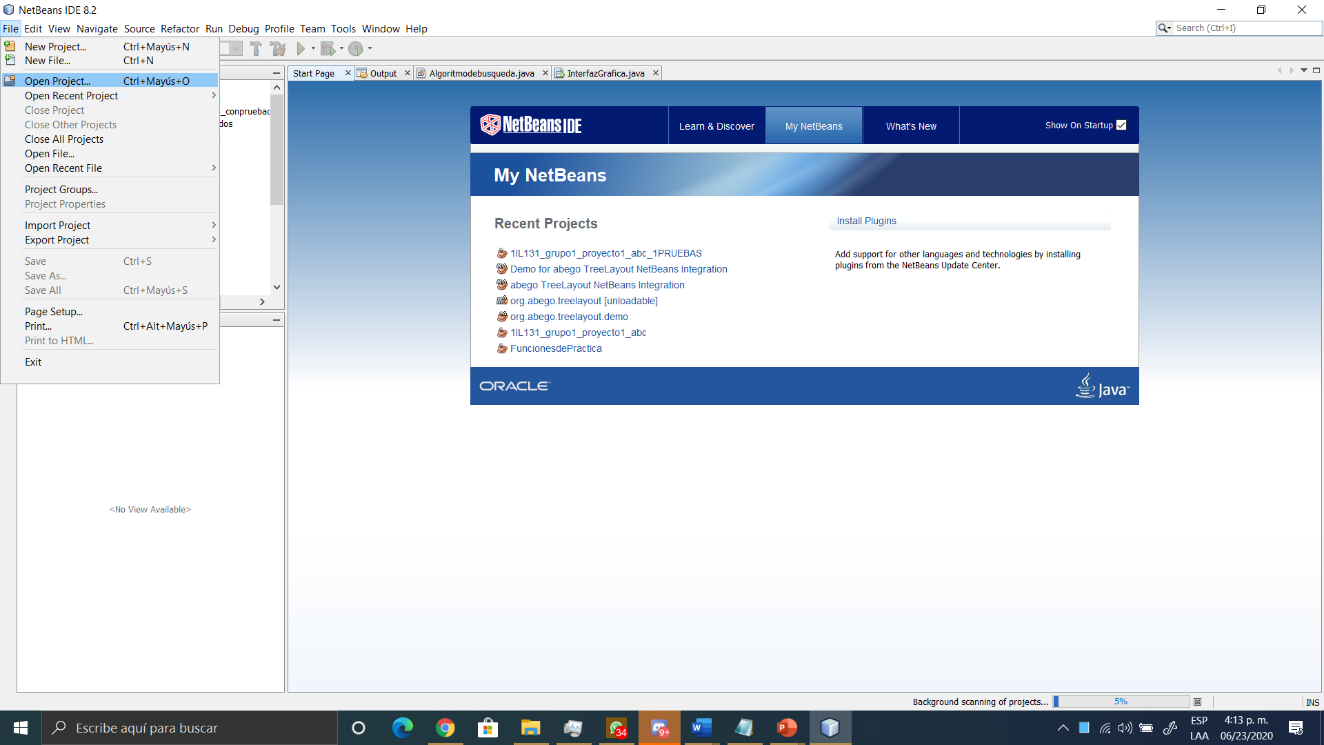
Enlace de descarga de NetBeans: <https://netbeans.apache.org/download/index.html>

Descargar la versión Apache NetBeans 12 LTS (NB 12.0).

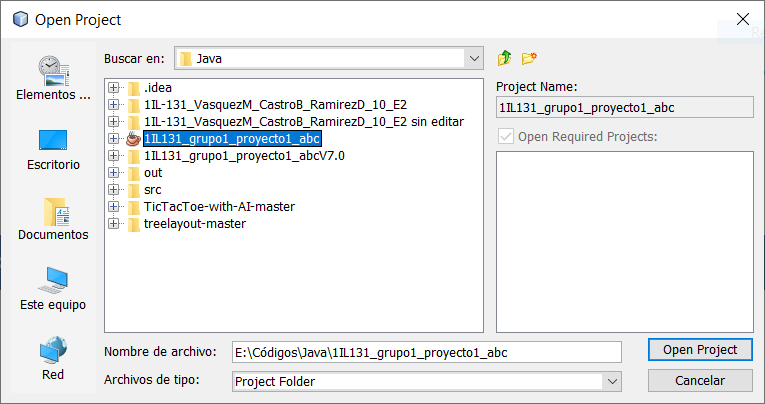
Guía de uso de NetBeans aquí: <https://netbeans.org/kb/>

Cómo abrir un proyecto en la herramienta NetBeans.

Luego de abrir el programa, ir a la pestaña File y seleccionar la opción Open Project.



Una vez abierta la siguiente ventana, se selecciona la carpeta que contiene el proyecto y se le da clic al botón Open Project.



Realizado este paso se agregará nuestro proyecto en la ventana de izquierda llamada “Projects”, debemos abrir el que se acaba de importar y buscar la carpeta “Source Package” la cual contiene los archivos ejecutables. Para ejecutarlo debemos abrir el archivo que contenga el símbolo de triángulo verde que significa que contiene el “Main”. Y dar clic en el botón de ejecutar.