INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO
SUBDIRECCIÓN ACADÉMICA

No. TT0316 Serie Amarilla 13 de Mayo del 2002.
Trabajo Terminal

Integrantes:

Carlos Eduardo Del Castillo Torrontegui
mercifulbein@hotmail.com
Calle Amsterdam 80, Int. 402.
Col. Hipódromo Condesa.
C. P. 06100
Tel. 52-86-31-93

Omar Sharif Valladolid García
true_crow@hotmail.com
Calle 319 # 531
Col. Nueva Atzacoalco
C. P. 07420
Tel. 57-53-70-39

Alberto Rodrigo Veru Bustamante
kidveru@hotmail.com
Calle Violeta 94 Int. H103
Col. Buenavista
C. P. 06350
Tel. 57-05-65-34

RESUMEN:

El objetivo del proyecto es realizar un juego de estrategia en tiempo real, el cual se caracteriza por utilizar recursos proveídos por el sistema para crear a un pequeño ejercito y guiarlo a la victoria a través de las estrategias que el jugador utilice para lograrlo. Este juego será capaz de aprender jugadas hechas por el usuario y así utilizarlas en su contra, aumentando con esto la dificultad del juego.

Palabras Clave: Juegos, Estrategia, Tiempo Real, Aprendizaje.

Director de Trabajo Terminal

M. en C. Victor Márquez García
Vo. Bo.

Carlos Eduardo Del Castillo Torrontegui
Omar Sharif Valladolid García
Alberto Rodrigo Veru Bustamante

INDICE GENERAL

CONTENIDO **PAGINA**

INTRODUCCIÓN

Hace ya más de 15 años que el entretenimiento por computadoras se ha esparcido y desarrollado por más y más compañías, entre ellas Atari, IBM, Nintendo entre otras. Pero en los últimos años se ha empezado a desarrollar un tipo de juego en especial que es de tipo estratégico, el cual consiste en cumplir algún objetivo específico dado por el juego mismo, utilizando los elementos que el juego proporciona. Posteriormente se agregaron otros elementos más interesantes a éste tipo de juegos, como son estrategias militares y escenarios más grandes generados en tiempo real, esto es, al momento de que el juego es cargado en el sistema, para esto es necesario el uso de técnicas de hilos o iteraciones multiprocesos, las cuales permite ligar todos los objetos entre sí dentro de la memoria o realizar las operaciones propias de cada uno de los elementos del juego.

El tipo de juego estratégico militar se ha venido desarrollando cada vez más en los últimos 6 años de manera secuencial, ya que la idea principal de éste tipo de juegos es la de un escenario de un tamaño mayor al de la pantalla, esto provoca que no todos los elementos puedan ser apreciados al mismo tiempo, y por lo mismo es necesario usar técnicas de programación diferentes a las que se solían utilizar con otro tipo de juegos. Algunos ejemplos de éste tipo de juegos, sus características principales y la compañía que desarrollo el juego se muestran a continuación en orden cronológico:

- **Sim City (MAXIS 1988):** Este fue uno de los primeros juegos de estrategia tal y como se describe anteriormente, aunque no fue un juego de tipo militar sino más bien de planeación, ya que el objetivo en éste juego era hacer prosperar a una ciudad creada por el usuario en los aspectos económicos, políticos y sociales, además de que no se competía directamente contra un enemigo, sino contra situaciones generadas por el juego que debían ser resueltas.
- **Popolous (Aklaim 1991):** Otro juego de estrategia el cual tampoco empleaba el genero militar, sino que fue un juego similar a Sim City pero con una temática distinta, además de que innovó su modo de vista, el cual era de tres cuartos, la cual permitía una visión más amplia de la situación del juego, muy usada en la actualidad.
- **Warcraft (Blizzard 1995):** Sin duda uno de los juegos que innovó la industria de los juegos de estrategia, ya que implemento el modo de estrategia militar, el cual consiste en eliminar a uno o más jugadores controlados por la computadora, utilizando un ejercito creado por el usuario, además de utilizar diferentes tipos de unidades o personajes con características diferentes, como velocidad, fuerza, armadura, etc. Este juego también implementó el modo de juego vía módem, el cual permitía que un usuario jugará contra otro utilizando la red telefónica, una red interna o una conexión directa por cable, pero se restringía solamente a dos jugadores. Este juego mantuvo el tipo de vista aérea directa.
- **Warcraft II (Blizzard 1996):** Muy parecido a su predecesor, pero una de las novedades de éste juego fue la utilización de algoritmos muy básicos de inteligencia artificial, lo cual hacía que los personajes en el juego se comportarán siempre diferente dependiendo del desarrollo del juego, además de que mantuvo la capacidad de jugar vía modem al igual que su predecesor, pero aumentando la posibilidad de jugar hasta 8 usuarios simultáneamente, posteriormente se lanzó una nueva versión, la cual puede ser jugada vía internet, mediante un portal diseñado para tal propósito. En éste juego hubo una mejora considerable de la calidad de los gráficos y sonido.
- **Pax Imperia (THQ Inc. 1997):** Un juego con gráficos y sonido muy revolucionarios y un motor de combate impulsado por inteligencia artificial y redes neuronales, lo que hizo a éste juego casi "pensar", ya que era posible hacer alianzas con la computadora, y ésta, analizaba la situación del usuario y tomaba una decisión dependiendo de la situación, además de una temática muy apta para lo anterior. Podía ser jugado vía Internet, pero en este caso, hasta por 16 jugadores simultáneamente. En éste juego se retoma el uso de la vista en tres cuartos. A partir de éste juego en adelante, todos los juegos nombrados han tenido la capacidad de ser jugados vía Internet.

- **Age of Empires (Microsoft 1997):** Un juego con una temática muy atractiva para los usuarios, de tipo medieval, pero tratando temas históricos reales y una vista de tres cuartos muy bien implementada, además del uso de inteligencia artificial para la respuesta de la computadora a los ataques del usuario. Otro juego que mantuvo su jugabilidad mediante Internet hasta para 8 jugadores. Este fue el juego que lanzó con más fuerza a Microsoft en uno de los mercados que no cubría con tanta fuerza, el de los videojuegos.
- **Starcraft (Blizzard 1998):** Este juego innovó en cuanto a temática se refiere, ya que utilizó una temática futurista, que se desarrollaba en el espacio, no sólo con personajes humanos, sino también con personajes extraterrestres, además fue de los primeros en implementar características evolutivas a la jugabilidad de la computadora, esto es, que la computadora "aprendiera" nuevas jugadas a partir de las hechas por el usuario, lo cual le dio una gran dificultad al jugar contra la computadora. Utilizó también la vista de tres cuartos.
- **Age of Empires II (Microsoft 1999):** Uno de los juegos de estrategia más recientes y mejor elaborados, ya que no sólo contempla aspectos militares, sino económicos, sociales y políticos, esto es, se retomaron ciertos aspectos de juegos antiguos, pero con la utilización de inteligencia artificial, esto ya no se torno tan aleatorio, sino que el juego reaccionaba de acuerdo a factores del desarrollo del juego, además de que mediante IA se manejaba con mayor realidad el comportamiento de los ejércitos enemigos y su aumento de dificultad a través del tiempo.

Los anteriores son sólo algunos de los juegos más sobresalientes que salieron al mercado, con éstos ejemplos podemos darnos cuenta de cómo se han agregado más elementos a los juegos para mejorar su calidad y dificultad, además de que las temáticas utilizadas en cada uno de los juegos los impulsa a su mejor aceptación.

Más concretamente hablando del sistema presentado en este reporte, consiste de un Juego de Estrategia como los elementos mencionados anteriormente, pero con la implementación de algoritmos de inteligencia artificial basados en los movimientos del usuario, además de la capacidad de generar los escenarios en tiempo real, con lo que se varia constantemente la composición de los mapas.

1. ANÁLISIS DEL SISTEMA

1.1. Descripción del Problema:

En la actualidad existen muchos juegos de estrategia en el mercado, muchos de ellos con gráficos excelentes y temáticas entretenidas, pero el problema más grande que tienen estos juegos es que el grado de dificultad siempre es el mismo, por lo que el usuario tiende a dejar de usar estos juegos porque no ofrecen más reto del que tiene programado.

El objetivo de este sistema fue resolver el problema de la dificultad de los juegos agregando capacidades de inteligencia artificial, imitando el comportamiento del usuario, para así hacer más real la experiencia al jugar, y ofrecer diferentes grados de dificultad a los diferentes usuarios que utilicen el sistema.

Otra de las características de los juegos actuales es que cuentan con un número limitado de mapas y escenarios, excepto algunos pocos que utilizan la generación de mapas en tiempo real, esta cualidad fue agregada al juego para proporcionar un factor más de constante cambio en la jugabilidad de nuestro juego.

1.2. Descripción del funcionamiento a nivel cero:

Para el correcto análisis del desarrollo del sistema, fue necesaria la realización de diagramas conceptuales y de nivel cero, con el cual se aprecia el funcionamiento básico del sistema.

Inicialmente, al usuario se le presentará la pantalla de un juego recién iniciado, con un mapa generado que contendrá obstáculos u objetos estáticos, además de una sola construcción para el usuario y una sola construcción para la computadora y con la que empezará a hacer crecer su ejercito. La entrada principal del sistema son los movimientos del usuario, esto es, el movimiento de uno o varios elementos dinámicos (no obstáculos) que realice el usuario, estos movimientos en conjunto pueden conformar jugadas. Ya dentro del funcionamiento del sistema, se pasarán las situaciones del juego generadas por el usuario a una matriz de Mapa de juego, la cual será actualizada constantemente, también las situaciones serán enviadas a los mecanismos de inteligencia artificial, donde se generarán las respuestas a los movimientos realizados por el usuario.

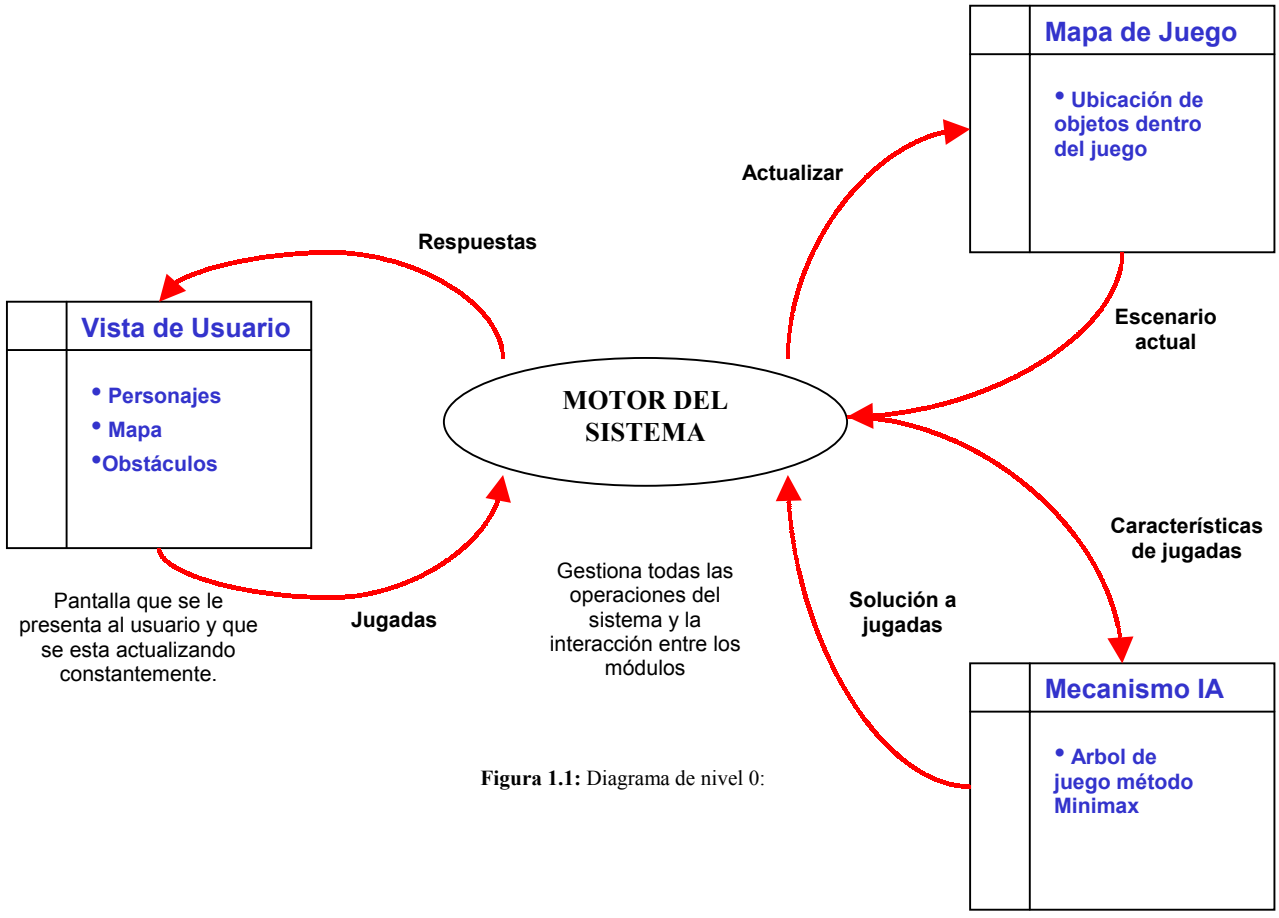


Figura 1.1: Diagrama de nivel 0:

1.3. Análisis general del sistema por módulos (Diagrama de Nivel 1)

Para un desarrollo consistente y una ejecución eficiente, fue necesaria la separación del sistema en módulos específicamente diseñados para una función concreta, además para la programación y codificación por partes para una depuración más fácil y rápida. A continuación se muestra una descripción por pasos de los procesos que realiza el sistema en su funcionamiento.

Descripción General de Funcionamiento:

1. El usuario inicia la ejecución del programa, por lo tanto inicia un nuevo juego.

2. El módulo generador de mapas se activa e inicializa la matriz de mapa de juego y la matriz de objetos estáticos.

3. El Motor recibe la matriz de mapa de juego y se inicializan los arreglos de objetos dinámicos y estáticos, todos estos arreglos se envían al módulo de Mapa de juego.

4. El módulo de mapa de juego crea e inicializa objetos de inicio en la matriz de objetos dinámicos, y los coloca en la matriz de mapa de juego.

5. En el módulo de mapa de juego se actualizan las posiciones de cada elemento en la matriz del mapa de juego según los arreglos de objetos dinámicos, y se agregan o se destruyen elementos creados o eliminados en los arreglos de objetos.

6. Se envían las matrices actualizadas de mapa de juego y los arreglos de objetos dinámicos al motor del juego, el cual reenvía la matriz de mapa de juego al módulo de rendering.

7. El módulo de rendering actualiza la matriz de pantalla y visualiza los objetos.

8. Ya visualizados los objetos, se envían las matrices de mapa de juego y los arreglos de objetos dinámicos al módulo de detección de colisiones.

9. En este módulo, se asignan las situaciones para cada objeto de la matriz de objetos dinámicos, según la posición de cada objeto y los demás objetos que los rodean, según la matriz de mapa de juego.

10. Se envía la matriz de objetos dinámicos al módulo de Inteligencia Artificial, donde se actualizan las características de cada objeto y se guardan los patrones más significativos en el archivo de conocimientos.

11. Con la matriz de objetos dinámicos actualizada, se regresa al paso 5, para realizar otra iteración.

Los módulos a desarrollar que se generaron a partir de este análisis fueron en total cinco, pero fuera de este funcionamiento general, se encuentra procesos de graficación, mensajes del sistema, manejo de sonidos entre otros que no conciernen al funcionamiento interno del sistema, sino que son procesos propios de una apliación que trabaja en la plataforma Windows. Los módulos que se desarrollaron se muestran en la siguiente sección de este reporte y son los siguientes:

- Módulo Generador de Mapas
- Módulo de Mapa de Juego
- Módulo de Rendering
- Módulo de Detección de Colisiones
- Módulo de Inteligencia Artificial

Para una visualización más concreta del funcionamiento del sistema por módulos, ver la figura 2, Diagrama de nivel 1.

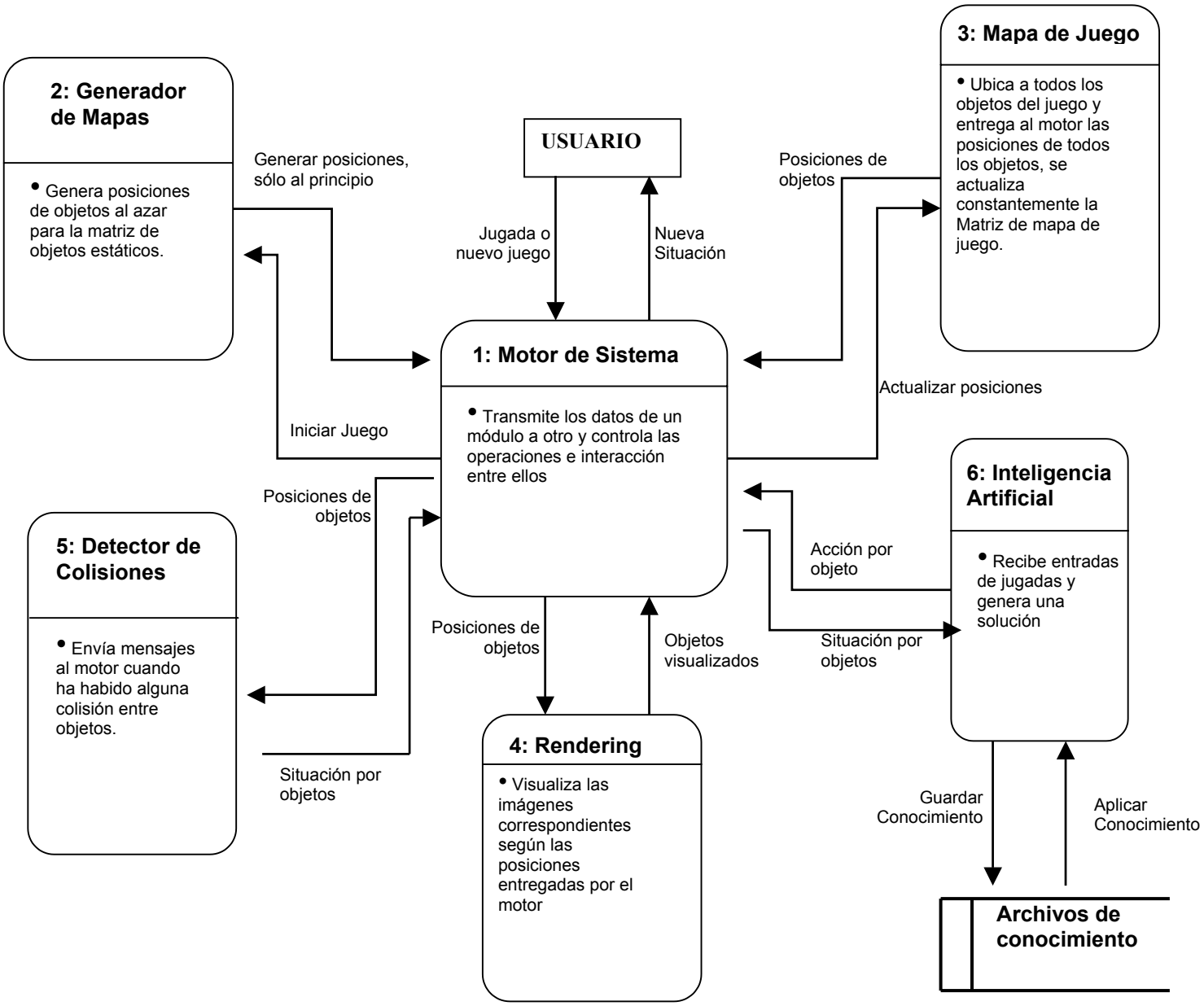


Figura 1.2: Diagrama de nivel 1

2. DISEÑO DEL SISTEMA

En esta sección se muestra el funcionamiento individual de cada uno de los módulos del sistema.

2.1. Módulo Generador de Mapas:

Objetivo: Genera la matriz del Mapa del juego y la Matriz de objetos estáticos.

Requerimientos: Ninguno.

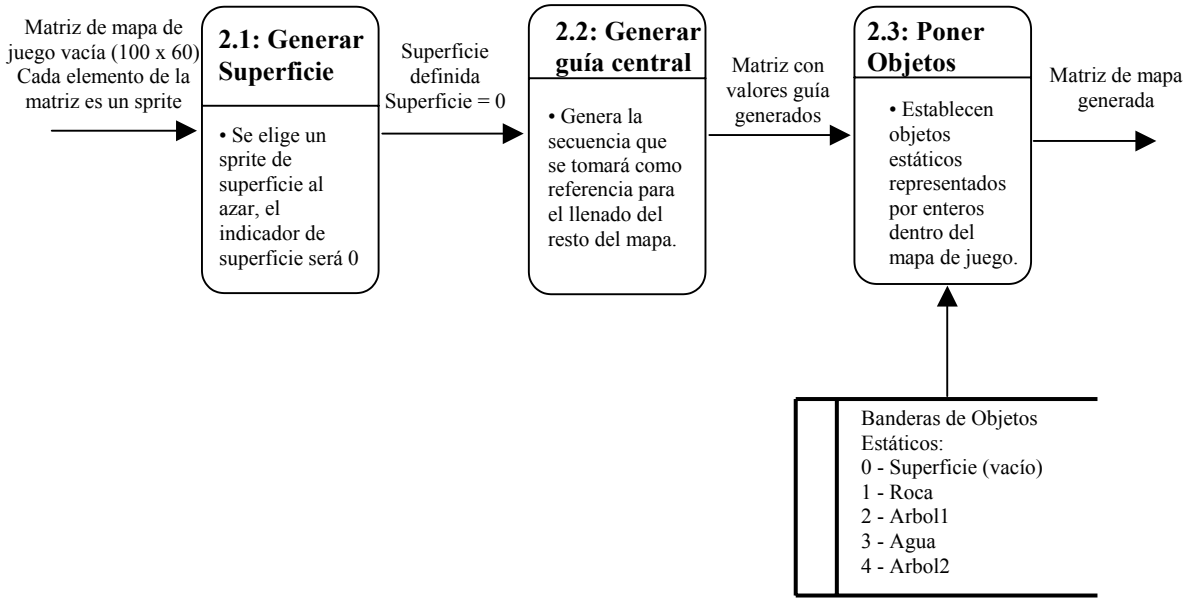


Figura 2.1: Diagrama del Módulo Generador de Mapas

Descripción del funcionamiento del módulo Generador de Mapas:

1. Este módulo entra al principio y una sola vez durante un juego.
2. Escoge una superficie al azar, a la cual se le asigna como bandera el cero.
3. Para colocar los objetos que van dentro del mapa, se utiliza una función gaussina, que sirve para generar la lógica de superficie de los escenarios.
4. Para colocar objetos que representan agua sólo se sigue una secuencia.

Datos de salida: Matriz de Mapa de Juego

2.2. Módulo de Mapa de Juego

Objetivo: Generar e inicializar la matriz de objetos dinámicos, actualiza la matriz del mapa de juego, genera una ruta para el desplazamiento de cada objeto y actualiza las características de cada objeto.

Requerimientos: Matriz de Mapa de juego, Matriz de objetos dinámicos y características de los objetos dinámicos.

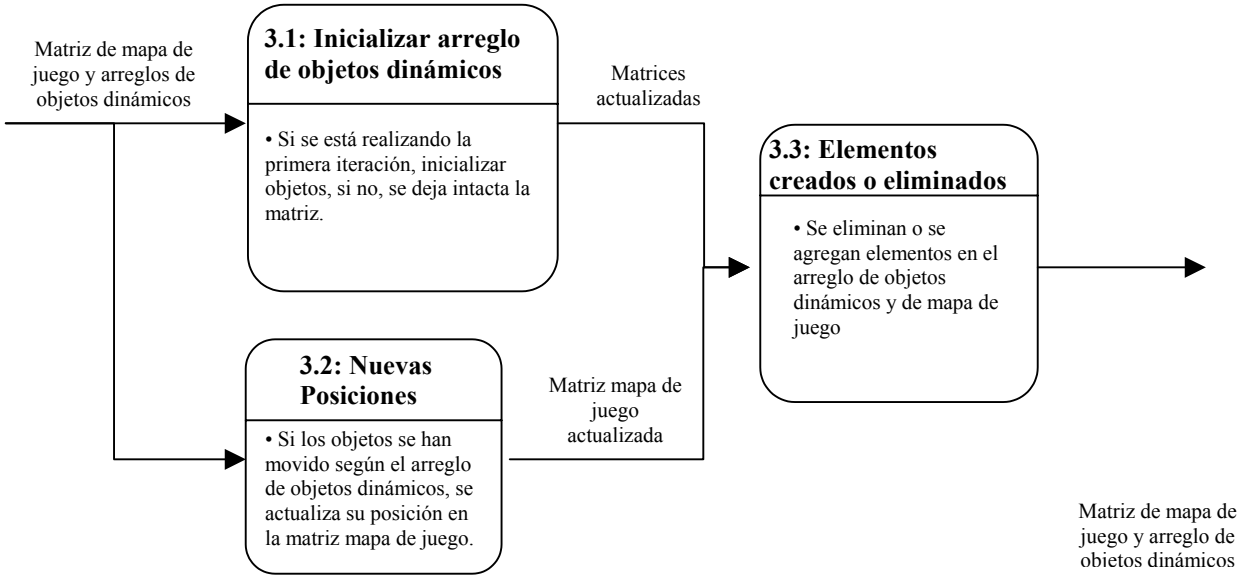


Figura 2.2: Diagrama de Módulo de Mapa de Juego

Descripción del funcionamiento del módulo mapa de Juego:

1. Verifica el estado y la posición en la que se encuentran cada uno de los objetos dinámicos del sistema. De la misma forma verifica la creación o eliminación de objetos dinámicos.

2. Actualiza las matrices de objetos y de mapa de juego.

3. Si algún objeto dinámico se ha desplazado, este módulo genera la ruta de desplazamiento que el objeto debe de seguir.

Datos de salida: Matriz de Mapa de juego
Arreglos de objetos dinámicos

2.3. Módulo de Rendering:

Objetivo: Visualiza gráficamente los objetos en pantalla.

Requerimientos: Matriz de mapa de juego

Datos de salida: Ninguno

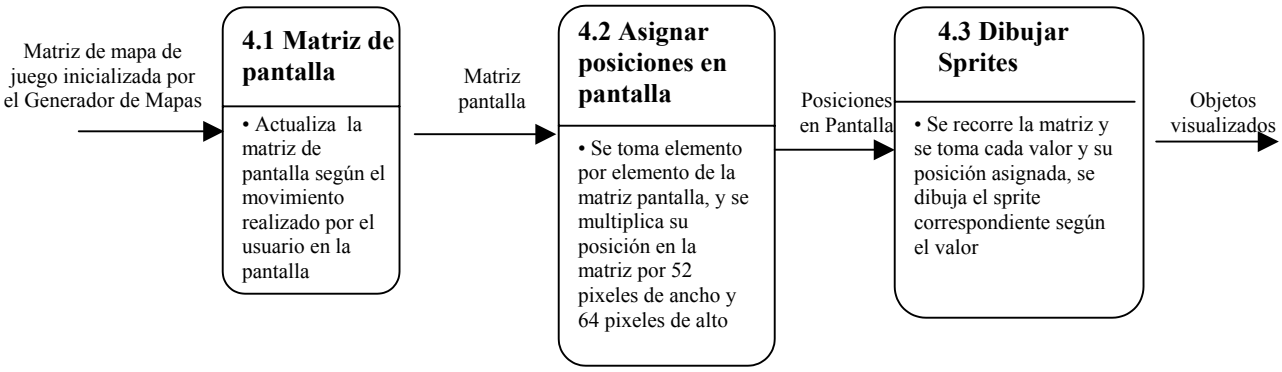


Figura 2.3: Diagrama de Módulo de Rendering

Descripción del funcionamiento del módulo Rendering:

1. Realiza la visualización de las imágenes en pantalla de acuerdo a la posición del mapa y lo entregado por el motor en las coordenadas indicadas.

2. Actualiza todas las imágenes ya existentes en visualización.

2.4. Módulo de Detección de Colisiones:

Objetivo: Este módulo realiza la detección de la interacción entre los objetos, es decir, nos indica el momento en el que interactúan dos ó más elementos del juego. También se encarga de obtener las características de la jugada que se desarrolla en ese momentos y de los objetos que en ella intervienen, con el fin de realizar una acción adecuada a la situación que se presente.

Requerimientos: Para la detección de las colisiones se requiere la posición actual del objeto en cuestión, así como su índice del arreglo de objetos, para identificar que objeto es.

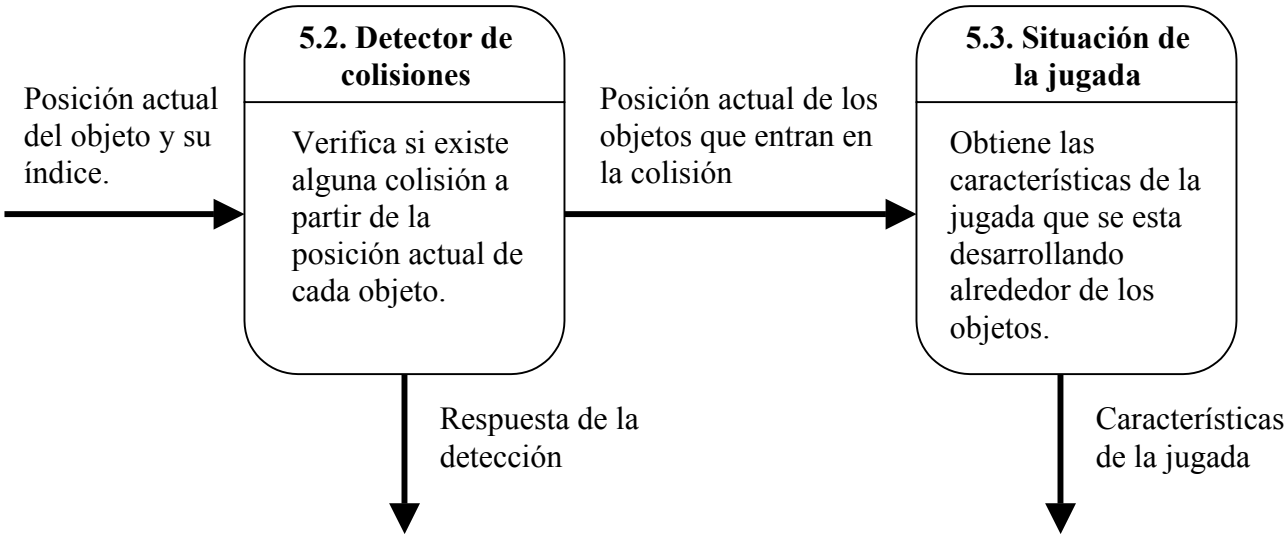


Figura 2.4: Diagrama de Módulo de Detección de Colisiones

Descripción General del Funcionamiento del Módulo Detector de Colisiones:

1. Las entradas de este módulo son las posiciones de todos los objetos dinámicos que están dentro del Mapa de juego, es decir la posición actual del objeto junto con su índice.

2. Para la detección de una colisión, se emplea una función de visión para el objeto dinámico, esta función genera un rango de visión para el objeto, que le da la capacidad de identificar a otro objeto que este lejos o cerca de él, con este rango se empieza a verificar la existencia de alguna posible colisión con los demás objetos que entren dentro de éste.

3. Sí detecta una colisión con uno o más objetos, analiza la situación del objeto y calcula el número de objetos aliados y el número de objetos enemigos (así como su fuerza y energía de cada uno), que interaccionan en la colisión. Estos datos son mandados a la función de evaluación, la cual generará la acción a realizar a partir de estos datos.

Las posibles acciones que se pueden generar son:

a) Atacar: En caso de que el estado del objeto que sé esta analizando sea mejor que el del contrincante. También si la orden que se le dio ese objeto fue directamente la de atacar.

b) Huir: En caso de que el estado del objeto a analizar este en desventaja con respecto su contrincante. También si la orden que se le dio ese objeto fue directamente la de huir.

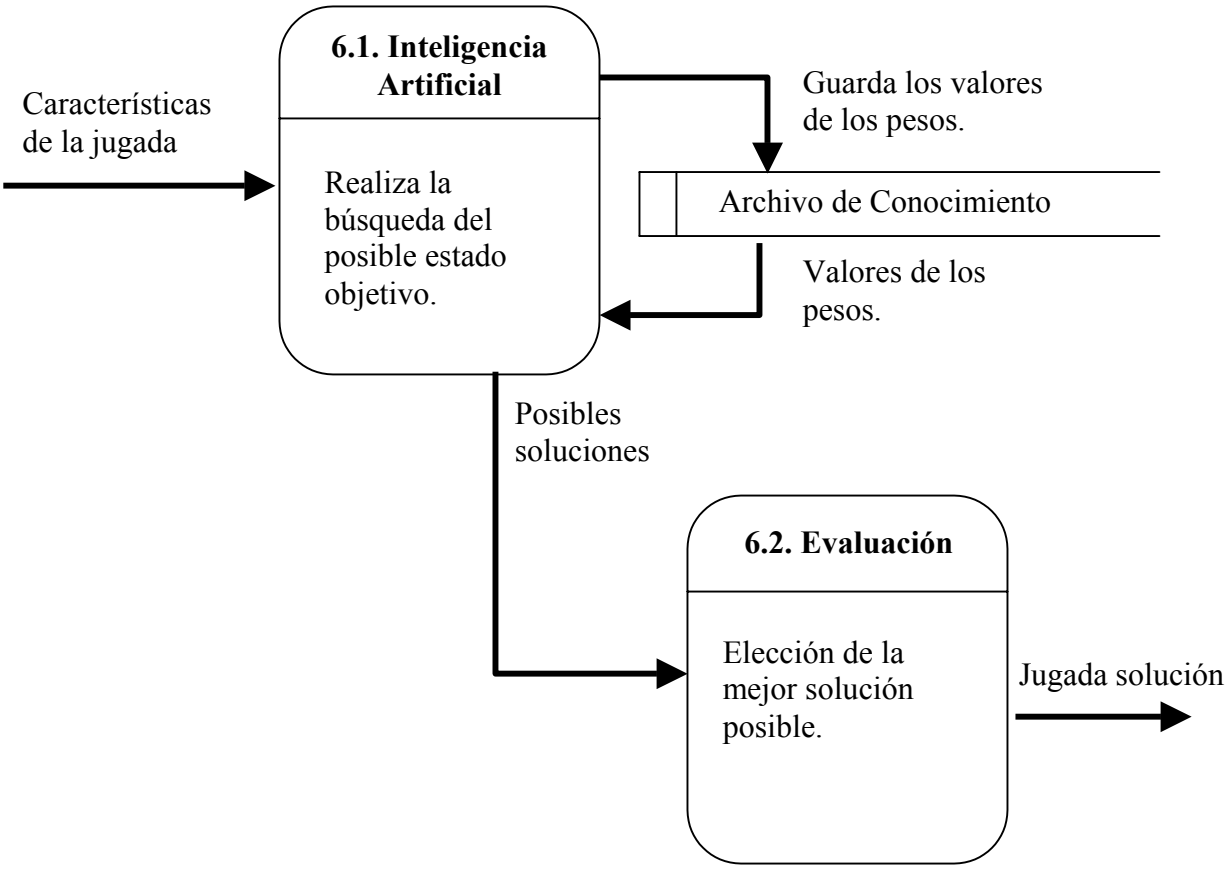
c) Ignorar: En caso d que el objeto se haya encontrado n otro objeto aliado y este sigue una ruta en específico.

Este módulo se va estar ejecutando siempre que sé este desplazando un objeto cualquiera.

2.5. Iteligencia Artificial:

Objetivo: Este módulo se encarga de realizar el aprendizaje del juego y además de proporcionar una solución de acuerdo a la situación de la jugada en la que se encuentran los objetos.

Requerimientos: Para lograr el aprendizaje, se necesitan las características de la jugada en acción, algunas de estas características son: tipos de objetos, la energía que tienen los objetos, el número de enemigos y el número de aliados que se encuentran alrededor de los objetos.



Fidura 2.5: Diagrama de Módulo de Inteligencia Artificial

Descripción General del Funcionamiento del Módulo Inteligencia Artificial :

1. Se toman las características del juego y éstas entran al submódulo de Inteligencia Artificial como la posición inicial del árbol de juego.

2. Se realiza el procedimiento de búsqueda minimax con la implementación de la poda alfa-beta.

3. Dentro del procedimiento de búsqueda se evaluarán los posibles movimientos mediante una función de evaluación. La cual contempla las características de las jugadas.

4. El aprendizaje se realiza mediante el incremento del valor de los pesos de aquellos componentes que han sugerido movimientos que nos llevarán a l victoria, mientras que decrece el valor de los pesos de aquellos componentes que han sugerido movimientos que han conducido a derrotas.

5. Al modificarse el valor de los pesos, estos se guardan en el archivo de conocimiento cada cierto tiempo.

6. El submódulo de inteligencia artificial arrojará una lista de los posibles movimientos que se pueden realizar a partir del estado inicial.

7. El submódulo de evaluación escogerá el mejor movimiento dentro de la lista de movimientos que recibe del submódulo de inteligencia artificial

8. El submódulo de evaluación nos da la jugada solución que se debe seguir para la situación actual.

3. IMPLEMENTACIÓN

3.1. Herramientas Utilizadas:

Durante la etapa de análisis del sistema, se tomaron decisiones sobre que herramientas se utilizarían para la implementación del juego, y se hizo un pequeño análisis de requerimientos para conocer qué herramientas disponibles eran las más aptas para cumplir con las necesidades del desarrollo. Una de las necesidades más importantes fue el uso de gráficos en 2D en modo de pantalla completa así como el uso de sonidos y del ratón, por esto se decidió desarrollar el juego bajo ambiente Windows, ya que este sistema operativo cumple con todas las necesidades de los juegos actuales además de ser la plataforma más utilizada comercialmente. También se pensó en la necesidad de programar algoritmos de inteligencia artificial por medio de un lenguaje ya conocido por los desarrolladores (C++), para evitar la pérdida de tiempo en el aprendizaje de nuevos lenguajes, pero al mismo tiempo utilizar una herramienta más poderosa que utilizara la codificación en dicho lenguaje y que pudiera utilizar todos los recursos que ofrece el sistema operativo Windows. A continuación se enlistan las herramientas utilizadas en el desarrollo del sistema:

- Microsoft Visual C++ 6.0:

Herramienta de programación en C++ para el sistema operativo Windows.

- Microsoft DirectX 8.0 SDK:

Conjunto de librerías con capacidad de uso de recursos de Windows (sonido, dispositivos externos, gráficos en 2D y 3D, etc.) para Microsoft Visual C++ 6.0. Las interfaces de DirectX utilizadas en el desarrollo del sistema fueron DirectDraw y DirectMusic.

- MetaCreations Poser 4:

Herramienta de modelación de personas y animales en 3D. Esta herramienta se utilizó para el modelado de los personajes del juego, esta modelación se realizó en 3D, pero se hizo una conversión a mapas de bits en 2D utilizando Corel Draw 10.

- Borland C++ 3.1:

Herramienta de programación que se utilizó para el desarrollo de algoritmos de inteligencia artificial.

- 3D Studio Max 4.0:

Herramienta de modelado en 3D que se utilizó para el modelado de objetos estáticos como árboles, rocas, etc.

- Corel Draw 10

3.2. Paradigma de programación:

Para decidir entre que paradigma de programación se utilizaría se tomó en cuenta el análisis realizado, y se decidió que sería necesario el uso de estructuras de datos para cada uno de los entes del sistema, pero si se tomaban estructuras de datos, el sistema perdería rapidez al momento de la ejecución, es por eso que se opto por el uso de clases y objetos para el manejo de todos los elementos dentro del sistema. Es por lo anterior que se eligió el paradigma de Programación Orientada a Objetos y a partir de esto se realizó el diseño del sistema sobre la base de objetos y clases.

Por todo lo anterior es por lo que se eligió la herramienta de programación Visual C++ 6.0 mencionada en la sección anterior, ya que la naturaleza de C++ es orientada a objetos, y con Visual C++ se permite, además del uso del paradigma orientado a objetos, el uso de todos los recursos del sistema operativo Windows.

3.3. Técnicas de Inteligencia artificial

La idea de que las computadoras pudieran jugar ha existido al menos desde que existen las computadoras. Desde entonces algunas personas se han dedicado a investigar la forma de lograrlo, las generaciones siguientes han continuado con esta idea y también se han empeñado en lograr que las computadoras aprendan y jueguen con inteligencia.

Existen dos razones para que los juegos pareciesen un buen dominio de exploración de la inteligencia de una máquina:

- Proporcionan una tarea estructurada en la que es muy fácil medir el éxito o el fracaso.
- Obviamente no necesitan grandes cantidades de conocimiento. Se pensó que se podrían resolver por búsqueda directa a partir del estado inicial hasta la posición ganadora.

La primera de estas dos razones aún sigue siendo válida y explica el continuado interés en el área de los juegos por computadora. Desgraciadamente, la segunda no es cierta para aquellos juegos que no sean muy simples. Por ejemplo, considere el juego del ajedrez.

- El factor de ramificación en media es más o menos 35.
- En una partida media, cada jugador realiza 50 movimientos.
- Por tanto, para examinar el árbol del juego completamente, se tendrían que examinar 35^{100} posiciones.

En los juegos, al igual que en otros dominios de problemas, existen diversas técnicas para la solución del problema como la técnica de búsqueda u otras técnicas más directas. En nuestro caso utilizaremos la técnica de búsqueda, debido a la semejanza de nuestro juego con el juego del ajedrez.

Así resulta evidente que un programa que realice una simple búsqueda exhaustiva en el árbol del juego, no podrá seleccionar ni siquiera su primer movimiento durante el tiempo de vida de su oponente. Es necesario algún tipo de búsqueda heurística. Además, que para mejorar la efectividad se necesita:

- Mejorar el procedimiento de generación de forma que sólo se generen movimientos buenos (generación de movimientos plausibles).
- Mejorar el procedimiento de prueba para que sólo se reconozcan y exploren en primer lugar los mejores movimientos (o caminos).

La semejanza de nuestro juego con el ajedrez, esta principalmente en que cada jugador puede realizar una cantidad considerada de movimientos por juego, y por tanto para examinar todo el juego se tendrían que examinar una cantidad considerada de posiciones. Por otra parte en los dos juegos todo depende del tipo de piezas, la posición de las piezas, los movimientos que puede realizar cada pieza y de la estrategia que se utiliza para lograr vencer al contrincante tomando en cuenta lo anterior. Específicamente en el juego de Amapola, todo depende del tipo de personaje, la posición de los personajes, las acciones que puedan realizar los personajes y de igual forma depende de la estrategia que se utiliza para lograr vencer al contrincante. Además en el juego de ajedrez todos los posibles movimientos se contemplan dentro de un tablero con cierto número de casillas. Para Amapola existe un mapa (en ajedrez tablero) con cierto número de sectores. El mapa es de 60x100 casillas, es decir, de 6 000 casillas, donde puede estar un solo personaje en cada casilla. Cada sector se compone de 10x10 casillas dando un resultado de 60 sectores y en cada sector caben 100 personajes (Ver Figura 3.1).

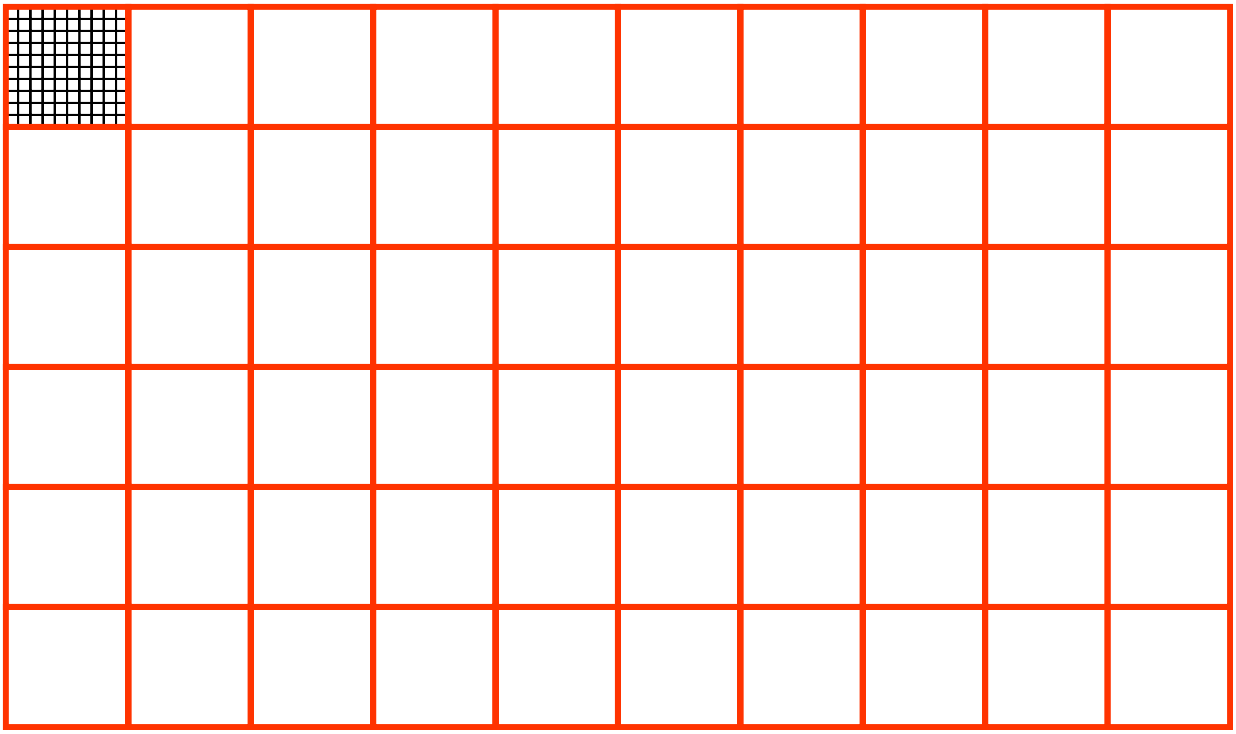


Figura 3.1: Mapa del juego y la división de los sectores.

La forma ideal de usar un procedimiento de búsqueda para encontrar una solución a un problema, es generar movimientos a través del espacio del problema hasta que se encuentra un estado objetivo. En el contexto de programas de juegos, un estado objetivo es aquél en el cual ganamos. Desgraciadamente, para juegos interesantes como el ajedrez al igual que el nuestro, normalmente no es posible buscar hasta encontrar un estado objetivo, ni siquiera disponiendo de un buen generador de movimientos plausibles. La profundidad del árbol (o grafo) resultante y su factor de ramificación son demasiado grandes. Con la cantidad de tiempo disponible, sólo es posible buscar menos de diez movimientos en el árbol (llamados capas). Así pues, para elegir el mejor movimiento, deben compararse las posiciones del tablero resultantes para determinar cuál es la más ventajosa. Esto se lleva a cabo con una función de evaluación estática, que usa toda la información disponible para evaluar posiciones individuales del tablero estimando la probabilidad de que conduzcan eventualmente a la victoria. Su función es similar a la función heurística h´ en el algoritmo A*: en ausencia de una información completa, elige la posición más prometedora. Naturalmente, la función de evaluación estática puede aplicarse simplemente a las posiciones generadas por los movimientos propuestos. Pero, puesto que es difícil producir una función como ésta que sea realmente precisa, es mejor aplicarla tantos niveles hacia abajo en el árbol de juego como lo permita el tiempo disponible.

Gran parte del trabajo realizado en programas de juegos se ha efectuado en el desarrollo de buenas funciones de evaluación estática. Una de las funciones de evaluación estática que se han propuesto es la de Turing que se basa en la ventaja de piezas, sumar simplemente los valores de las piezas de la máquina (M), los valores de las piezas del contrincante (U) y calcular entonces el coeficiente U/M. Existe otra aproximación más sofisticada, en donde la función de evaluación estática era una combinación lineal de diversas funciones simples, cada una de las cuales parecía ser importante. Estas funciones son llamadas funciones de Samuel, las cuales incluían, además de la obvia ventaja de piezas, otros aspectos tales como la capacidad de avance, el control del centro, movilidad, etc. Estos factores e combinan asignando a cada uno de ellos un peso apropiado y sumando a continuación todos los términos obtenidos. Así pues, la función de evaluación estática tenía la forma:

$$c1 * ventajapiezas + c2 * avance + c3 * controlcentro .....$$

Para nuestro caso la función de evaluación estática se ve de la siguiente forma:

$$c1 * fuerza \; + \; c2 * energía \; + \; c3 * ventajapiezas$$

Hay incluso algunos términos no lineales que reflejaban combinaciones de estos factores. Pero no se conocen los pesos correctos que deben asignar a cada uno de los componentes. Por lo tanto, se usa un mecanismo de aprendizaje en el cual se incrementan los pesos de aquellos componentes que han sugerido movimientos que llevaran a la victoria, mientras que los pesos de aquellos que han conducido a derrotas decrecen. De esta forma es como cambian el valor de los pesos y se realiza el aprendizaje.

A continuación se describen las funciones heurísticas que consideramos para el desarrollo de nuestro juego:

Funciones Heurísticas.

Para la evaluación de las jugadas, se tiene que tomar en cuenta una función de evaluación, la cual es nuestra heurística para el aprendizaje de las jugadas y la discriminación de resultados.

La función de evaluación, por lo tanto es nuestra función heurística.

La función heurística que se utilizó fue la siguiente:

$$FH1 = \left(\sum_{i=1}^{N} EU_i - \sum_{j=1}^{M} EM_j \right) + \left(\sum_{i=1}^{N} FU_i - \sum_{j=1}^{M} FM_j \right) + \left(N - M \right)$$

con:

EU = Energía del Usuario.
EM = Energía de la Máquina.
FU = Fuerza del Usuario.
FM = Fuerza de la Máquina.

N = Número de unidades de ataque del Usuario en el sector analizado.
M = Número de unidades de ataque de la Máquina en el sector analizado.

Ésta función evalúa el sector de la siguiente forma:

Cada objeto que se encuentra en el sector analizado se va acumulando en las variables N y M, tanto del usuario como de la máquina.

Al momento de que se analiza el objeto, se va tomando la energía de cada unidad de ataque y se va sumando con la siguiente, siempre y cuando pertenezcan al mismo grupo del usuario o de la máquina, al igual pasa con la fuerza.

De esta forma se en la función de evaluación se toman esos datos y se procede a realizar una resta de las energías, las fuerzas y el número de unidades de ataque, de esta forma se determina quién esta mejor armado que el otro.

Ésta función, nos indica quién puede estar mejor armado, pero en que grado sabemos que esta correcta, es por eso que se procedió a realizar otra función de evaluación (FH=Función Heurística).

La Función Heurística es la siguiente:

$$FH2 = \sum_{i=1}^{N}\left(EU_i * FU_i\right) - \sum_{j=1}^{M}\left(EM_j * FM_j\right)$$

con:

EU = Energía del Usuario.
FU = Fuerza del Usuario.

EM = Energía de la Máquina.
FM = Fuerza de la Máquina.

N = Número de unidades de ataque del Usuario en el sector analizado.
M = Número de unidades de ataque de la Máquina en el sector analizado.

Ésta función a diferencia de la anterior, toma la energía de cada unidad de ataque del usuario ó de la máquina y la multiplica por su fuerza, este resultado se va acumulando para cada tipo de objeto (usuario ó máquina).

Con esto se toma el valor real de cada objeto, consiguiendo con esto otros valores distintos.

Como tercera Función Heurística se tomo la primera función, pero modificada, la cual queda de la siguiente forma:

$$FH3 = \left(w_1 \sum_{i=1}^{N} EU_i - w_2 \sum_{j=1}^{M} EM_j \right) + \left(w_3 \sum_{i=1}^{N} FU_i - w_4 \sum_{j=1}^{M} FM_j \right) + \left(w_5 N - w_6 M \right)$$

con:

w_1, w_2, w_3, w_4, w_5, w_6 = Pesos de aprendizaje.

Esta función se difiere de la primera en que se tienen seis pesos, los cuales se van modificando conforme se presenten las entradas a la función.

Debido a la naturaleza de nuestro juego (juegos bipersonales), es decir, conforme los valores se van propagando hacia atrás, deben hacerse suposiciones diferentes en aquellos niveles en que el programa elige el movimiento y en los niveles alternativos en donde la elección la realiza el oponente. Para esto el método más usado es el procedimiento *Minimax*.

Búsqueda Minimax

Este es un procedimiento de búsqueda en profundidad, de profundidad limitada.

La idea de este algoritmo consiste en comenzar en la posición actual y usar el generador de movimientos plausibles para generar un conjunto de posiciones sucesivas posibles. Entonces se puede aplicar la función de evaluación estática a esas posiciones y escoger simplemente la mejor. Después de hacer esto, puede llevarse hacia atrás ese valor hasta la posición de partida para representar nuestra evaluación de la misma. La posición de partida es exactamente tan buena para nosotros como la posición generada por el mejor movimiento que podamos hacer a continuación. Aquí se va a suponer que la función de evaluación estática devuelve valores elevados para indicar buenas situaciones para nosotros, de manera que nuestra meta es maximizar el valor de la función de evaluación estática de la siguiente posición de l tablero.

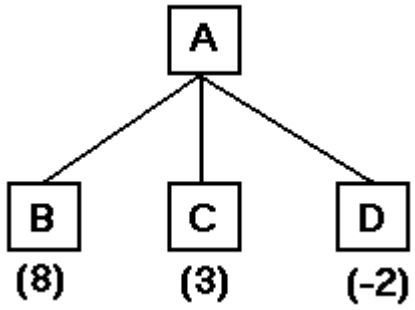


Figura 3.2: Búsqueda de una sola capa

En la figura 3.2 se muestra un ejemplo de esta operación. Supone una función de evaluación estática que devuelve valores entre –10 y 10, donde 10 indica una victoria para nosotros, -10 un a victoria para el oponente y 0 un empate. Puesto que nuestra meta es maximizar el valor de la función heurística escogemos movernos a B. Al llevar hacia atrás el valor de B hasta A, podemos concluir que el valor de A es 8, puesto que sabemos que se puede llegar a una posición con un valor de 8.

Puesto que sabemos que la función de evaluación estática no es completamente precisa, nos gustaría llevar la búsqueda más allá de una sola capa. Esto podría ser muy importante ya que después de nuestro movimiento, podría parecer que nuestra posición es muy buena pero, si mirásemos un movimiento más allá, veríamos que una de nuestras piezas también es capturada, por lo que la situación no era tan favorable como parecía. Por tanto nos gustaría prever que es lo que sucederá en cada una de las nuevas posiciones del juego del siguiente movimiento realizado por el oponente. En lugar de aplicar la función de evaluación estática a cada una de las posiciones que acabamos de generar, aplicamos a cada una de ellas el generador de movimientos plausibles, generando un conjunto de posiciones posteriores para cada una de ellas. Si queremos pararnos ahí, en una previsión de dos capas podríamos aplicar la función de evaluación estática a cada una de dichas posiciones, tal y como muestra la figura 3.3.

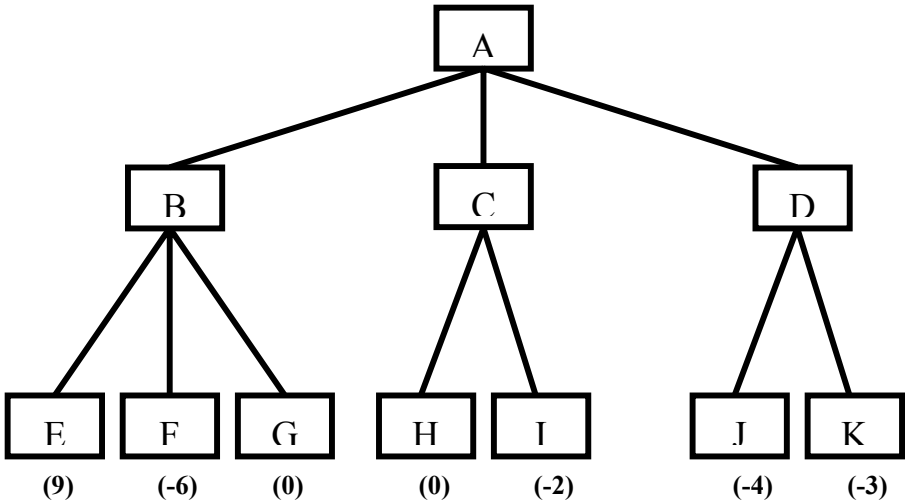


Figura 3.3: Búsqueda de dos capas.

Pero ahora debemos de tener en cuenta el hecho de que el movimiento que se realiza a continuación debe elegirlo el oponente, y por tanto, deberá propagarse hacia atrás al siguiente nivel. Supongamos que realizamos el movimiento B. Entonces el oponente debe elegir entre los movimientos E, F y G. El objetivo del oponente es minimizar el valor de la función de evaluación, por lo que puede esperarse que elija moverse a F. Esto significa que si hacemos el movimiento B, la verdadera posición en la que desembocamos en el siguiente movimiento es muy mala para nosotros. Esto es cierto aunque el nodo E represente una posible configuración que fuese buena para nosotros. Pero ya que no nos toca a nosotros mover en este nivel, no podremos elegirlo. En la figura 3.4 se muestra el resultado de propagar los nuevos valores hacia la raíz del árbol. En el nivel representado por la elección del oponente, se elige el menor valor que se propaga hacia la raíz. En el nivel que representa nuestra elección se ha elegido el nivel máximo.

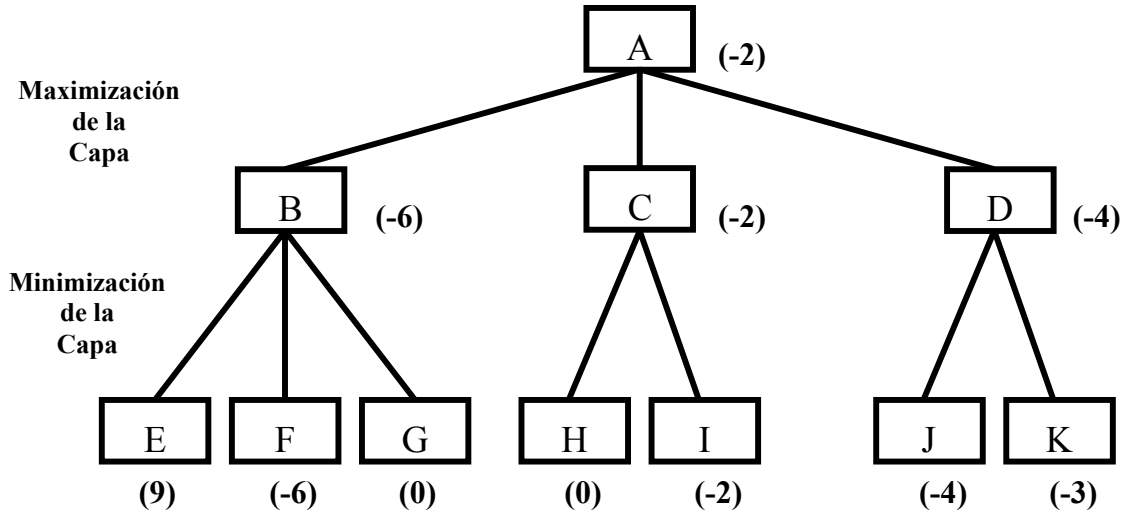


Figura 3.4: Propagación hacia atrás de los valores en una búsqueda de dos capas

Una vez que se han propagado hacia tras los valores del segundo nivel, resulta claro que el movimiento correcto que podemos realizar en el primer nivel, dada la información disponible es C, puesto que no hay nada que el oponente pueda hacer ahí para producir un valor peor que –2. Este proceso puede repetirse para tantos niveles como lo permita el tiempo disponible, y las evaluaciones más precisas que se produzcan pueden usarse para elegir el movimiento correcto en el nivel más alto. La alternancia de maximización y minimización en capas alternas cuando las evaluaciones se envíen de regreso a la raíz, se corresponde con las estrategias opuestas que siguen los dos jugadores y que da a este método el nombre de Minimax. Existen una implementación del método de Minimax que es la poda Alfa-Beta.

Poda Alfa-Beta

El procedimiento minimax es un proceso primero en profundidad. Cada camino se explora tan lejos como lo permita el tiempo, la función de evaluación estática se aplica a las posiciones del juego en el último paso del camino, y el valor debe entonces propagarse hacia atrás en el árbol de nivel en nivel. Una de las ventajas del procedimiento primero en profundidad es que a menudo puede mejorarse su eficiencia usando técnicas de ramificación y acotación, en las que pueden abandonarse rápidamente aquellas soluciones parciales que son claramente peores que otras soluciones conocidas. Pero, así como fue necesario modificar ligeramente nuestro procedimiento de búsqueda para tratar los jugadores maximizador y minimizador, también es necesario modificar la estrategia de ramificación y acotación para incluir dos acotaciones, una para cada uno de los jugadores. Esta estrategia modificada se denomina poda alfa-beta. Requiere el mantenimiento de dos valores umbral, uno que representa la cota inferior del valor que puede asignarse en último término a un nodo maximizante(que llamaremos alfa) y oto que representa la cota superior del valor que pueda asignarse a un nodo minimizante(que llamaremos beta).

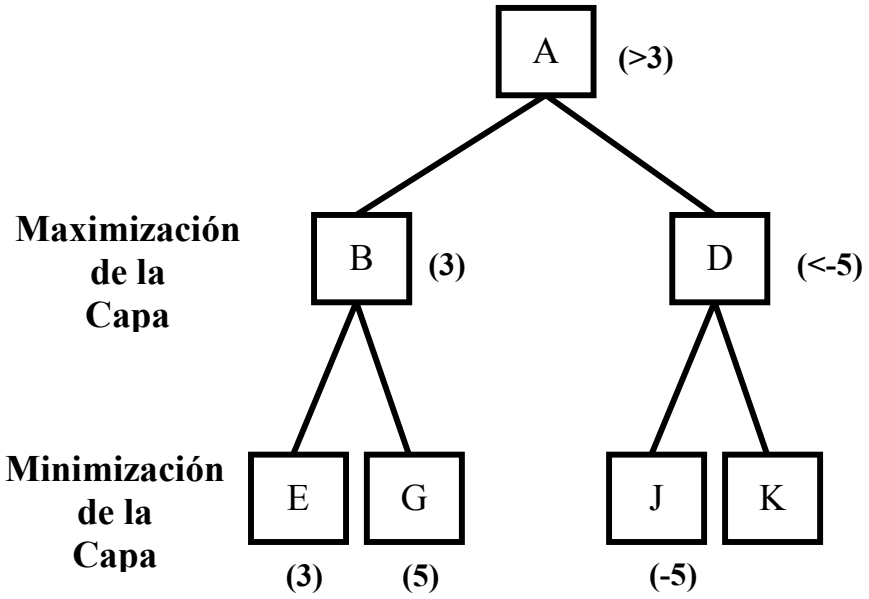


Figura 3.5: Una poda alfa.

Para ver como funciona el procedimiento alfa-beta, consideremos el ejemplo de la figura 3.5.

Después de examinar el nodo F, sabemos que el oponente tiene garantizado un resultado de –5 ó menos en C (puesto que el oponente es el jugador minimizador). Pero también sabemos que tenemos un resultado seguro de 3 o más en el nodo A, donde podemos llegar si movemos a B. Cualquier otro movimiento que produzca un resultado de menos de 3 es peor que el movimiento a B, y podemos ignorarlo. Con solo examinar F podemos asegurar que un movimiento en C es peor (será <= -5), sin necesidad de mirar el resultado del nodo G. Así pues, no necesitamos en absoluto explorar el nodo G. Naturalmente puede parece que la poda de un nodo no justifica el gasto de grabar los límites y examinarlos, pero si estuviésemos explorando un árbol de seis niveles, entonces no solo habremos eliminado un nodo único, sino un árbol entero de 3 niveles de profundidad.

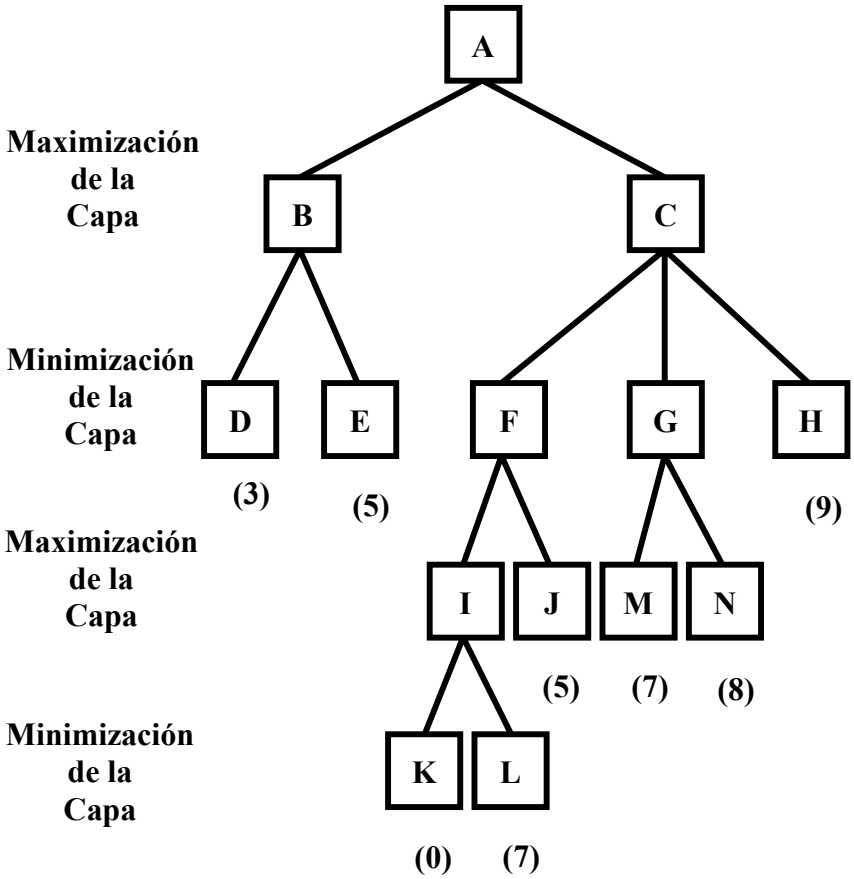


Figura 3.6: Podas alfa y beta

Para ver como pueden usarse los dos umbrales, alfa y beta, consideremos el ejemplo de la figura 3.6. Al buscar este árbol, se explora el árbol entero encabezado por B, descubrimos que en A podemos esperar un resultado de 3 como mínimo. Cuando este valor alfa se pasa hacia F, nos permitirá evitar la exploración de L. Veamos por qué, después de examinar K, vemos que I proporciona un tanteo máximo de 0, lo que significa que F produce un mínimo de 0, pero esto es menor que el valor alfa de 3, por lo que no necesitamos considerar más ramas de I. El jugador maximizado ya sabe que no debe elegir C, y de ahí a I, puesto que si realiza ese movimiento el tanteo resultante no será mayor que 0, y en lugar de ello puede lograrse un tanteo de 3 moviendo a B.

Veamos ahora como utilizar el valor de beta, después de podar cualquier exploración posterior de I, se examina J, que produce un valor de 5, el cual se asigna a su vez como valor de F (puesto que es el máximo de 5 y 0). Este valor se convierte en el valor de beta en el nodo C. Nos garantiza que C será 5 ó menos. A continuación debemos expandir G, en primer lugar se examina M que tiene un valor de 7, el cual se pasa a G como su valor provisional. Pero ahora se compara 7 con beta (5). Es mayor, y el jugador que tiene el turno del nodo C esta tratando de minimizar. Por lo tanto ese jugador no elegirá G, que le conduciría a un resultado de 7 como mínimo, puesto que hay una alternativa de volver a F, lo que producirá un tanteo de 5. Así pues, no es necesario explorar ninguna de las otras ramas de G.

A partir de este ejemplo, se ve que en los niveles maximizantes podemos excluir un movimiento tan pronto como quede claro que su valor será menor que el umbral actual, mientras que en los niveles minimizantes la búsqueda terminará cuando se descubran valores mayores que el umbral actual. La exclusión de un movimiento posible del jugador maximizante significa realmente podar la búsqueda en un nivel minimizante. Veamos de nuevo el ejemplo de la figura 3.4. Una vez determinado que C es un mal movimiento para A, no nos molestaremos en explorar G, o cualquier otro camino, en el nivel minimizante por debajo de C. Por tanto, la forma en que se usan

ahora alfa y beta consiste en que la búsqueda de un nivel minimizante pueda terminar cuando se descubre un nivel menor que alfa, mientras que en el nivel minimizante la búsqueda puede terminar al descubrir un valor mayor que beta. La acotación de la búsqueda en un nivel maximizante cuando se encuentra un nivel alto puede parecer, al principio, opuesto a la intuición, pero si se piensa en que solo llegaremos a un nodo concreto de un nivel maximizante si el jugado minimizante del nivel anterior lo elige, entonces tiene sentido.

El Aprendizaje.

Para realizar el aprendizaje se considera la estructura de una red neuronal simple que es la de una red neuronal de función lineal, la estructura se muestra en la siguiente figura (Figura 3.7.).

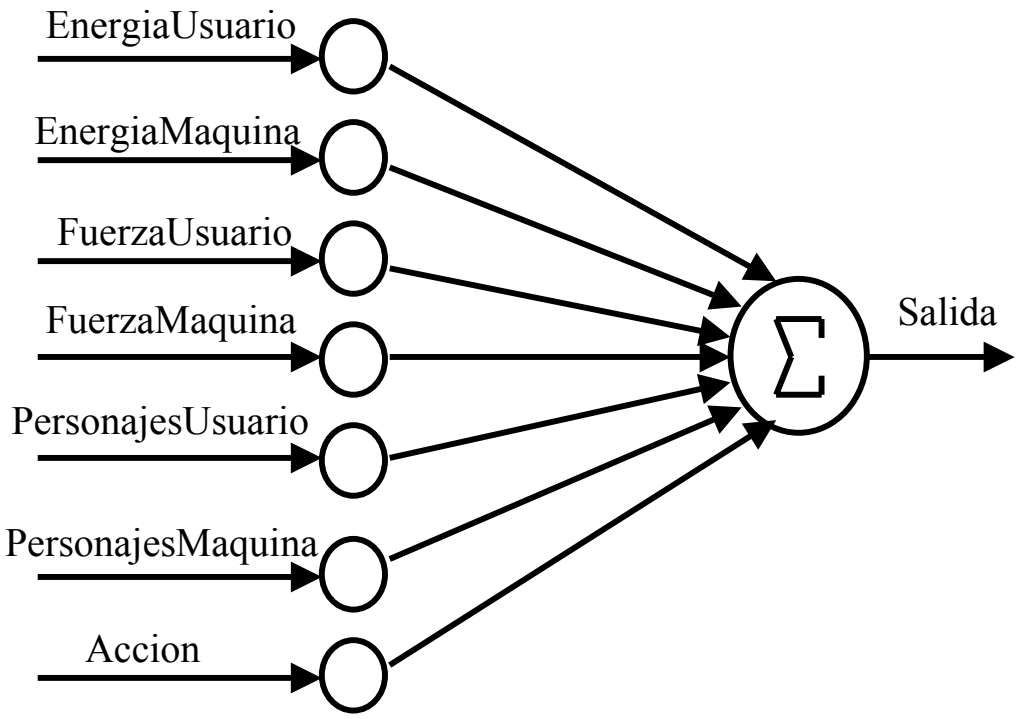


Figura 3.7: Estructura de la red neuronal.

En la figura 3.7. se observan las entradas de la red neuronal las cuales comprenden el estado de una situación en cierto momento del juego, es decir, que estas entradas son las características que se presentan durante una colisión. Una colisión es la interacción de los personajes: "cuando un personaje ya sea Narcotraficante o Policía se encuentra con uno o más personajes que sean Narcotraficantes o Policías". La justificación de considerarlo así es por que la máquina siempre que detecte una colisión tenderá a analizar la situación y poder tomar una decisión. Por ejemplo; cuando dos personajes del mismo bando se encuentran la maquina analizará la situación y decidirá no hacer nada o ignorar a su compañero. Ahora pensemos que un Narcotraficante colisiona con un Policía, la maquina analizará la situación y decidirá atacar o huir, esto depende, del resultado que nos arroje la función heurística que hemos escogido al iniciar el juego. Pero como ya se ha mencionado, un resultado muy grande positivo favorece a los personajes que maneja la maquina y un resultado muy chico negativo favorecerá al usuario.

Por otra parte, el aprendizaje del juego se realiza mediante la red neuronal y el conocimiento adquirido por la maquina se guarda en un archivo. Al iniciar el juego y no existir algún archivo de conocimiento, la maquina no cargará los valores de conocimiento y por lo tanto iniciará sin conocimiento, entonces cada vez que el que se detecte una colisión y las características sean enviadas a la red neuronal, se comparan los patrones de entrada con los patrones que ya se tienen, como la maquina no tiene conocimiento alguno no compara nada, y prosigue a guardar los

patrones de esa situación. Estos valores se almacenarán temporalmente en un arreglo de Nx7, como se muestra en la siguiente figura (Figura 3.8.):

	e1	e2	f1	f2	n1	n2	ac	s
p1	int	int	int	int	int	int	int	int
p2								
p3								
pn								

Figura 3.8: Estructura de la matriz de conocimiento.

Donde:

Pn: Es el número de patrones de conocimiento que posee la maquina.
E1: Es la energía de la maquina en la situación que se analiza.
E2: Es la energía del usuario en la situación que se analiza.
F1: Es la fuerza de la maquina en la situación que se analiza.
F2: Es la fuerza del usuario en la situación que se analiza.
Ac: Es la acción a seguir según la decisión que toma la función heurística.
S: Es el resultado que se obtiene de la función heurística.

Después de guardar los patrones que se muestran en la Figura 3.8. la red neuronal regresa la decisión que consideró mejor. Cada vez que la red neuronal trabaje comparará los patrones que ha guardado con los que entran en ese momento y si no los encuentra los almacenará.

El archivo de conocimiento se abre solo dos veces una es al iniciar el juego, para leer y cargar el conocimiento ya adquirido. Y la otra es la terminar el juego para escribir los nuevos conocimiento adquiridos al final del archivo.

3.4. Descripción Técnica y de codificación del Sistema

Para el funcionamiento correcto del sistema, se realizó un análisis por módulos, de donde se generaron los siguientes:

- Generador de Mapas
- Mapa de Juego
- Rendering
- Detección de Colisiones
- Inteligencia Artificial

Todos los anteriores módulos fueron programados de manera separada y creando diferentes clases ligadas entre ellas para el correcto funcionamiento del sistema.

Las clases que se utilizaron para el desarrollo del sistema y su descripción por funciones miembro y datos miembro, así como sus relaciones se presentan a continuación:

3.4.1. Clase CAmapolaTT0316App:

Esta clase se utiliza para crear la ventana principal donde se mostrará la aplicación, además de contener la función que llamará a las iteraciones multiprocesos donde se realizarán todos los movimientos y acciones del juego.

En toda la ejecución del sistema solo se creará un solo objeto de esta clase, el cual será la ventana donde se mostrará todo el juego, a continuación se hace una descripción de las funciones miembro más relevantes de esta clase:

- **InitInstance:**

Esta función inicializa la ventana donde será mostrado el juego y crea el objeto pMain, derivado de la clase CMain, el cual llevará el control de todo el juego.

- **OnIdle:**

Esta función se está llamando constantemente durante toda la ejecución del sistema hasta que se termina la ejecución del mismo, o cuando el programa este en pausa, esto es, cuando la ventana de la aplicación pierde el foco (cuando se hace cambio de ventana con Alt + Tab). Esta función llama a la función Tick de la clase CMain.

3.4.2. Clase CMapa:

Esta clase es la que contiene la matriz del Mapa del juego, la cual contendrá a todos los elementos que se encuentren en el mapa, además de que se estará actualizando constantemente en cada iteración, además de contener funciones y datos miembro para realizar la actualización y visualización de esta matriz, primero se muestra una breve descripción de los datos miembro más relevantes y posteriormente una descripción de las funciones miembro.

Datos Miembro:

- **MatrizMapa:** Es una matriz de enteros y la representación del mapa principal, en la cual se estarán colocando enteros de acuerdo al elemento que se quiera colocar en el mapa, el objeto a colocar estará definido por el número que contenga la matriz en cierta posición, el gráfico correspondiente a cada número está definido en la función RenderMapa contenida dentro de la clase CMain, la cual será descrita más adelante. La matriz de mapa es de un tamaño de 100 por 60 posiciones, lo que nos da un total de 6000 elementos que pueden ser colocados en un juego, entre objetos dinámicos y estáticos.

- **MatrizPantalla:** Es una matriz de enteros que se utiliza para visualizar la parte de la matriz de mapa que se desee ver, la función RenderMapa contenida en la clase CMain utiliza esta matriz para visualizar el juego en pantalla. La matriz de pantalla es de un tamaño de 20 por 12 posiciones y si se viera en forma ampliada, las dos matrices anteriormente descritas se verían como en la figura 8.

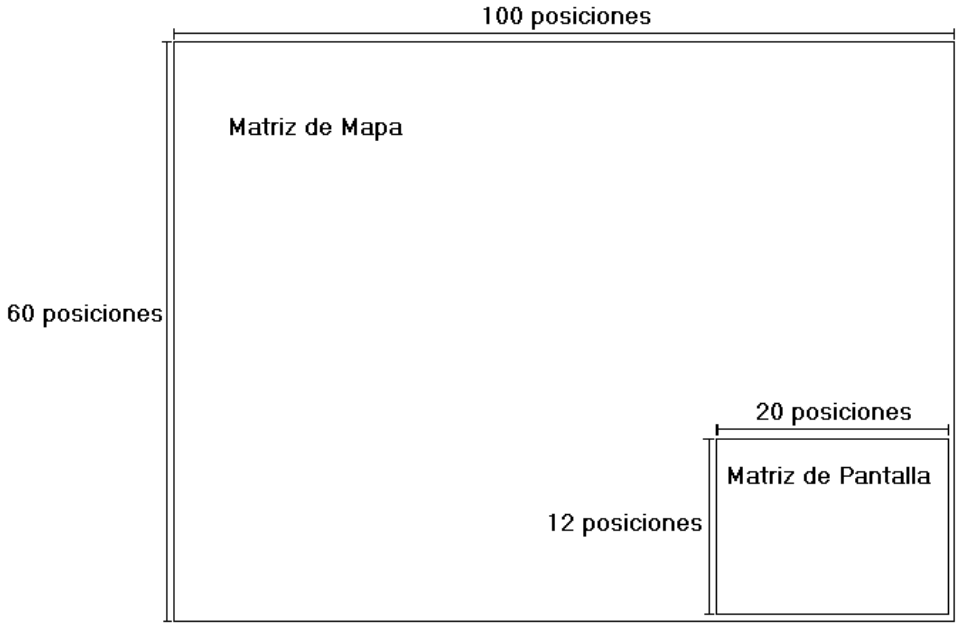


Figura 3.7: Tamaño de matrices de mapa y de pantalla

- **Datos posMatrizX y posMatrizY:** Estos datos le indican a las funciones de mover matriz en que parte de la matriz de mapa se encuentra la matriz de pantalla, para posteriormente visualizar la matriz de pantalla.

Funciones Miembro:

- **Constructor CMapa:**

El constructor de la clase solo inicializa todos los datos miembro de la clase e inicializa el generador de números Gaussiano para la generación del mapa en la función GeneraMapa.

- **Función GenerarMapa:**

Esta función genera números del cero al cuatro dentro de la matriz de mapa mediante un proceso de generación de números al azar utilizando un generador de números Gaussiano. Primeramente se generan números flotantes para las líneas centrales de la matriz, esto es, para cuatro líneas, del centro hacia arriba, del centro hacia abajo, del centro hacia la derecha y del centro hacia la izquierda, a partir de estos números se generan los siguientes tomando el número generado más el número anterior; finalmente se da un valor entre cero y cuatro en la matriz de mapa dependiendo del valor flotante que se tenga en la matriz temporal de números flotantes, todo lo anterior para obtener una matriz de números enteros entre cero y cuatro, la cual será la matriz de mapa.

- **Funciones de MoverMatriz y NoMover:**

Se utilizan para actualizar la matriz de pantalla cuando el usuario realiza un movimiento del mapa o cuando no se realiza ningún movimiento de acuerdo a la posición de los contadores de posición de matriz, esto para posteriormente visualizar el contenido de la matriz de pantalla.

3.4.3. Clase Unidades:

Esta clase se utiliza para la creación de las unidades y construcciones del usuario y de la computadora, para proporcionar características y funciones individuales a cada uno de los objetos dinámicos, como son los soldados activos e instalaciones inmóviles. Cada objeto creado por el usuario o por la computadora, será almacenado en su respectivo arreglo de apuntadores a objetos, esto para mantener el control de condición y posición de cada objeto creado dentro del juego, y saber en que momento debe ser alterado o destruido.

La clase Unidades no cuenta con funciones miembro ya que todas las funciones de las unidades se llevan a cabo utilizando múltiples funciones externas. Los datos miembro importantes de la clase Unidades son:

- **CompMon:** Contador de unidades enemigas que están dentro del rango del objeto.
- **UserMon:** Contador de unidades aliadas que están dentro del rango del objeto.
- **energia:** Energía de la unidad.
- **fuerza:** Fuerza de ataque de la unidad.
- **tipoUnidad:** Bandera que identifica el sprite asociado al objeto.
- **posMonoXini y posMonoYini:** Posición actual del objeto dentro de la matriz de mapa.
- **posMonoXfin y posMonoYfin:** Posición de destino del objeto dentro de la matriz de mapa. Nota: Cuando las posiciones finales e iniciales de un objeto son iguales, significa que el objeto esta en reposo, en caso contrario, el objeto se encuentra en movimiento.
- **posMonoXiniAux y posMonoYiniAux:** Posición siguiente dentro de la ruta de un objeto en movimiento.

3.4.4. Clase CMain:

Esta es la clase principal del juego, es de donde se mandan llamar todas las demás clases y funciones del sistema, y al iniciarse la ejecución del juego se crea un objeto de esta clase por medio de la función InitInstance de la clase CAmapolaTT0316App, con este objeto se llevará todo el control de todo el juego. Esta clase también contiene todas las funciones de graficación, control de acceso al modo exclusivo, manejo de sonidos, creación de paletas de colores y todo lo referente a la visualización y sonido. A continuación se describen los datos miembro y las funciones miembro más significativas de la clase CMain:

Datos Miembro

Dentro de la clase CMain se invocan objetos de clases externas no relacionadas, por lo que es necesario manejarlos por medio de apuntadores apuntadores a objetos; algunos de estos se describen a continuación:

- **pMapa**: Es el apuntador que se utiliza para manejar el objeto Mapa, este es un solo apuntador durante toda la ejecución del juego.
- **pSector[60]:** Arreglo de apuntadores a objetos procedentes de la clase CnodoSec para manejar cada uno de los sectores del mapa.
- **pUserUnit[300]:** Arreglo de apuntadores a objetos procedentes da la clase Unidades, el cual se utiliza para ubicar la condición y posición de todas las unidades móviles(soldados) del usuario.
- **pUserBuilding[300]:** Arreglo de apuntadores a objetos procedentes da la clase Unidades, el cual se utiliza para ubicar la condición y posición de todas las unidades estáticas (construcciones) del usuario.
- **pCompUnit[300]:** Arreglo de apuntadores a objetos procedentes da la clase Unidades, el cual se utiliza para ubicar la condición y posición de todas las unidades móviles(soldados) de la computadora.
- **pCompBuilding[300]:** Arreglo de apuntadores a objetos procedentes da la clase Unidades, el cual se utiliza para ubicar la condición y posición de todas las unidades estáticas (construcciones) de la computadora.
- **numPersonajes y numConstrucciones:** Son contadores que controlan el número de unidades o de construcciones que el usuario ha creado, esto para ubicarlos dentro de los arreglos de objetos y poder manipularlos en cualquier momento.
- **CnumPersonajes y CnumConstrucciones:** Son contadores que controlan el número de unidades o de construcciones que la computadora ha creado, esto para ubicarlos dentro de los arreglos de objetos y poder manipularlos en cualquier momento.
- **Xsector y Ysector:** Estas variables son mandadas a la función GeneraPosSector y le dicen a la computadora a donde debe mover sus unidades de acuerdo al movimiento generado por los procesos de inteligencia artificial que más adelante serán descritos.

También se utilizan algunos contadores para controlar los eventos que se realizan de manera retardada en el juego, como son el aumento automático del capital del usuario y de la computadora, el movimiento de los objetos, la animación de los sprites, etc.

Funciones Miembro

- **RenderMapa:**

Esta función se encarga de visualizar el contenido de la matriz de pantalla de la clase CMapa, haciendo una conversión de las coordenadas de matriz a coordenadas de pantalla y colocando ahí el mapa de bits correspondiente al número que se contenga en dicha posición de matriz de mapa.

Tabla 3.1: Banderas utilizadas por RenderMapa en la matriz de mapa para la visualización y manejo del mapa en pantalla:

Entero	Grafico a Mostrar	Mostrado en Pantalla
0	Vacío	No
1	Roca	Si
2	Arbol tipo 1	Si
3	Agua	Si
4	Arbol tipo 2	Si
21	Soldado raso Usuario	Si
22	Soldado espía Usuario	Si
28	Armería Usuario	Si
29	Barraca Usuario	Si
30	Cuartel Usuario	Si
31	Soldado Narco Computadora	Si
32	Soldado Dealer Computadora	Si
38	Armería Computadora	Si
39	Barraca Computadora	Si
40	Cuartel Computadora	Si
41	Relleno de Construcciones	No
42	Limite de mapa	No

- **MotorJuego:**

Esta función es la de funcionamiento principal, ya que manda llamar a todas las demás funciones de la clase CMain, las cuales a su vez invocan a los demás objetos. Cada vez que se manda llamar a esta función se realiza una iteración, y dentro de esta iteración se realizan todas las operaciones de actualización de posiciones de unidades, movimientos del mapa, etc., es por eso que le llamamos iteraciones multiprocesos, con lo cual se sustituye el uso de multihilos. En esta función también se realiza la visualización de los elementos de interfaz de usuario del juego, como son el teclado de funciones y los mensajes de texto para indicar el estado del juego. En conclusión, a esta función se le puede llamar como el motor o master del juego.

- **Ruta:**

La función ruta se encarga de generar la siguiente posición a donde el objeto en cuestión se moverá siguiendo la ruta hacia el punto final indicado por el usuario o por la computadora, y se utiliza para todos los objetos generados y que sean activos, esto quiere decir, sólo se aplica a las unidades de combate. La función ruta se invoca cada vez que alguno de los objetos ya sea del usuario o de la computadora, tiene sus coordenadas iniciales y finales diferentes, y se sigue invocando hasta que el objeto con estas características, vuelve al estado de tener sus coordenadas iniciales y finales iguales.

- **Colision:**

La función de colisión se encarga de la detección de alguna interacción de un objeto con otro, es decir, detecta cuando una unidad de combate se encuentra con otra o con alguna construcción enemiga, cada que ocurre una colisión, se verifican cuales fueron los elementos que participaron en ésta para determinar la acción que realizará el objeto que detectó la misma. Esta función trabaja en conjunto con la función de visión.

- **Vision:**

La función de visión se encarga de generar un rango de visión para cada unidad de combate ya sea del usuario o de la computadora, éste rango de visión se define a partir de la posición en la que se encuentra el objeto, hasta n posiciones alrededor de éste, esta función nos sirve para que el objeto pueda distinguir los elementos que entran en su visión y de esta forma poder realizar una detección de colisión.

- **OnUsrCreaEscena:**

Esta función es la primera en ejecutarse ya creada la ventana del sistema, y se encarga de la inicialización de todos los objetos del juego, del mapa y de las unidades iniciales del usuario y de la computadora, además de inicializar todos los apuntadores y contadores.

- **OnUsrRestauraSuperficies:**

Establece cada uno de los mapas de bits que se utilizarán y que serán visualizados para cada entero de la matriz de mapa, además calcula el tamaño de éstos mapas de bits para que sean mostrados correctamente.

- **EliminaUnit:**

Toma el objeto deseada y convierte su apuntador a Nulo, además de que elimina su presencia de la matriz de mapa y recorre todos los demás objetos que estén delante de él en el arreglo de objetos y los recorre para no dejar huevos vacíos en el arreglo; funciona simulando un vector de java.

- **ChecaSector:**

En análisis posteriores, se dedujo que sería necesario el uso de sectores de mapa, esto es, seccionar el mapa por cuadrantes para controlar mejor los movimientos de la computadora, por esto se desarrollo la función ChecaSector, la cual guarda en un objeto Sectores, el cual es apuntado por un arreglo de apuntares de 60 posiciones, las características de cada sector de mapa, como son, número de unidades del usuario, número de unidades de la computadora, energía de usuario, energía de computadora, entre otros, esto para conocer las condiciones en que se encuentra el sector destino del siguiente movimiento de la computadora y así generar jugadas más inteligentes. En total se cuentan con 60 sectores, ya que cada sector comprende de 10 posiciones de ancho por 10 posiciones de alto; posteriormente se hablará más profundamente de la clase CNodoSec, la cual controla el uso de sectores del mapa y también se hablara del uso de sectores para la inteligencia artificial.

- **GeneraPosSector:**

Con esta función se asigna una posición a cada una de las unidades de la computadora de acuerdo al sector a donde debe moverse.

3.4.5. Clase CNodoSec:

Esta clase se utiliza para poder guardar la situación de cada sector del mapa, es decir, contabiliza el número de personajes de la máquina y las características de estos personajes en el sector. De igual forma lo hace para el oponente. Los datos miembros de la clase CNodoSec son:

- **Naliados:** Guarda el número de personajes que tiene la maquina en el sector.
- **Nenemigos:** Guarda el número de personajes que tiene el oponente en el sector.
- **energia1:** Guarda la sumatoria de la energía de los personajes de la maquina en el sector.
- **energia2:** Guarda la sumatoria de la energía de los personajes del oponente en el sector.
- **fuerza1:** Guarda la sumatoria de la fuerza de los personajes de la maquina en el sector.
- **fuerza2:** Guarda la sumatoria de la fuerza de los personajes del oponente en el sector.

El constructor de la clase CNodoSec inicializa todo los datos miembros a cero.

3.4.6. Clase CDeep:

Esta clase se utiliza para guardar las características de los nodos del árbol de juego. Los datos miembros son:

- **sec:** Guarda el numero del sector en el que se encuentra.
- **mov:** Guarda el posible movimiento según el sector.
- **valor:** Guarda el valor del movimiento.
- **x:** Guarda la posición X del posible movimiento.
- **y:** Guarda la posición Y del posible movimiento.
- **accion:** Guarda la acción a seguir del movimiento.
- **CDeep *Hijo:** Son los movimientos que se generan a partir del nodo padre.
- **CDeep *Padre:** Apunta al nodo padre.

El contructor de la clase CDeep inicializa los apuntadores Hijo y Padre a NULL y el valor de sec y mov a cero.

A continuación se describen funciones que son utilizadas tanto por la clase CNodoSec y por la clase CDeep, por lo que se tomaron como funciones globales a las dos clases.

- **void MinMax (Deep *P ,int Jugador, int Prof);**

Esta función crea el árbol del juego. A partir de un estado inicial del juego, esta función genera los posibles movimientos que se pueden realizar y a su vez genera los posibles movimientos para cada posible movimiento encontrado del estado inicial, esta función es recursiva y parará al llegar a la profundidad 6 o al encontrar el movimiento que nos lleva al éxito.

El parámetro P, es el nodo padre, el cual se toma como nodo inicial para crear los posibles movimientos.
El parámetro jugador es una bandera para saber si es el turno del oponente o de la maquina.
El parámetro Prof nos indica el nivel en el que estamos.

- **void BuscaActual (void);**

Esta función lleva a cabo una simulación de los posibles movimientos generados y lleva el seguimiento del juego para poder tomar las características de cada situación según se vayan realizando los movimientos.

- **void MejorVal (Deep *P, int op);**

Esta función realiza la propagación hacia arriba del mejor valor encontrado entre los movimientos generados según el jugador.

- **int ObtieneNodos (Deep *P, int Jugador, int Prof);**

Se encarga de generar los nodos del nodo padre que recibe.

- **void IniSectores (void);**

Guarda las características de los personajes tanto de la máquina como del oponente en cada sector, es decir, en cada clase del arreglos de apuntadores a apuntadores de CnodoSec se guardan el número de personaje, la fuerza y energía por cada jugador.

- **void MapaMov (int x, int y, int mov[ ]);**

Genera los posibles movimientos del sector actual a los sectores de alrededor y checa que estos movimientos no excedan los limites del Mapa y por último activa los movimientos en el arreglo mov.

- **int ReturnSec (int Sec, int Mov);**

Regresa el sector del posible movimiento generado.

- **int ChecaObstaculo (int Cantidad, int Sector, int Jugador, int pos[ ]);**

Checa que los sectores de los movimientos generados en MapaMov contengan el espacio suficiente para poder realizar el movimiento, es decir, que los personajes que van a realizar el movimiento puedan moverse todos o la mitad al sector indicado. De lo contrario se descarta el movimiento generado.

- **int FuncionEval (int Jugador,int SecI, int SecM);**

Esta función evalúa el posible movimiento de un sector a otro según las características de estos, y nos regresa un valor que nos indica la ventaja o desventaja en esemovimiento.

- **void EnemigoCercano (int Sector, int Jugador);**

Cuando en el posible movimiento del sector no contiene algún personaje de la máquina o algún personaje del oponente, se considera como un movimiento de avance. Sí los movimientos de avance son varios encogerá sólo uno conforme realice la búsqueda del personaje del oponente o de la maquina más cercano que encuentre.

- **void Valor (Deep *P, int Jugador);**

En esta función se crean los nodos hijo de cada nodo padre según todos los posibles movimientos generados del mapa. La forma de hacer esto es escoger los dos mejores valores encontrados, los dos peores valores encontrados y los dos valores medios encontrados. El parámetro de mejores y peores depende del jugador, ya que para la maquina los valores más positivos serán los mejores, mientras que para el usuario los mejores valores serán los más negativos.

Estos valores se acomodan en el árbol de tal forma que siempre examine primero los mejores valores de cada jugador.

- **void ComparaNodo (Deep *P);**

Un vez generamos los nodos hijo del nodo padre se checa que los valores de los nodos hijos no se repitan, dándole preferencia a los nodos que tiene como acción atacar en primer lugar, la acción unirse con aliados en segundo lugar y por último la acción de moverse.

GLOSARIO DE TÉRMINOS

Objeto Dinámico: Tipo de elemento que puede ser controlado o manipulado de alguna forma por el usuario dentro del juego, algunos sólo pueden ser ubicados donde el usuario lo desee y no podrán ser movidos en ningún momento, y otros podrá ser manipulada su posición y comportamiento constantemente.

Objeto Estático (Obstáculos de mapa): Tipo de objeto o elemento que no podrá ser manipulado por el usuario, y su posición será siempre la misma dentro del mapa.

Colisión: Momento en el que dos objetos interactúan o se interceptan.

Arreglos de Objetos Dinámicos: Arreglo de apuntadores a objetos de un máximo de 300 posiciones donde se almacenarán todos los objetos dinámicos (unidades activas capaces de moverse o atacar) creados en el juego.

Arreglos de Objetos Estáticos: Arreglo de apuntadores a objetos de un máximo de 300 posiciones donde se almacenarán todos los objetos estáticos (unidades pasivas capaces de crear unidades activas como construcciones) creados en el juego.

Matriz de Mapa de Juego: Matriz de 100 x 60 la cual representa el tamaño total del mapa que se usará, y donde cada posición de la matriz es un objeto que se encuentra en el mapa.

Matriz de Pantalla: Matriz de 20 x 12 que representa la parte del mapa que será visualizada en pantalla, ésta matriz se estará moviendo dentro de la matriz de mapa de juego de acuerdo al movimiento del ratón por los bordes de la pantalla. Cada posición de esta matriz representa un sprite.

Sprite: Unidad de imagen de 52 x 64 pixeles que representa una posición en la matriz de pantalla.

Render: Dibujar o visualizar un sprite en pantalla.

CONCLUSIONES

- El desarrollo de sistemas de entretenimiento, como juegos de video, hoy en día tiene la necesidad de agregar factores que cambien constantemente con la jugabilidad y, si es posible, con la temática o con los elementos que contiene el juego, para mantener los juegos entretenidos a pesar de que el usuario los utilice por largo tiempo, con esto se mantendrá el interés por el mismo juego, redituando aún más a los productores de estos juegos.

- Los juegos actuales contienen características como generación de escenarios al azar, las llamadas bombas de tiempo de personajes (generación de personajes nuevos a través del tiempo, a partir de que el juego fue ejecutado por primera vez) o creación de armas o elemento nuevos utilizando funciones evolutivas, con todo lo anterior, los juegos se vuelven cada vez más entretenidos y duraderos.

- Tomando nuestro caso como ejemplo, los juegos también están siendo equipados con técnicas de inteligencia artificial y algoritmos evolutivos para que el modo de juego de la computadora sea aún más real y se compare con la forma de jugar de jugadores humanos, para así ofrecer un mayor reto al usuario y darle una razón más a los juegos para que duren más en el gusto de los usuarios.

- Las técnicas evolutivas y de inteligencia artificial también pueden ser utilizadas en aplicaciones ajenas al entretenimiento, como procesadores de textos, sitios web o herramientas de modelación gráfica, para deducir las necesidades más especificas de los usuarios y poder autoprogramarse para cumplir con estas necesidades. Con lo anterior, se pueden agregar funciones adaptativas al usuario a las aplicaciones.

- El presente desarrollo fue solo una forma de implementar algoritmos de inteligencia artificial a un juego, pero posiblemente en el futuro se puedan aplicar los mismos algoritmos utilizados en este sistema en otras aplicaciones o incluso en otro tipo de juegos.

- Durante el desarrollo de este Trabajo Terminal, los alumnos pudieron conocer más a fondo las herramientas utilizadas, y sacar provecho de los recursos del sistema operativo utilizado, para en un futuro ser capaces de realizar aplicaciones con herramientas más completas o especializadas, además, fueron capaces de implementar algoritmos de inteligencia artificial que son muy utilizados actualmente en el desarrollo de aplicaciones inteligentes.

BIBLIOGRAFÍA

- Microsoft Visual C++ 6: Programación Avanzada en Win32
  Francisco Javier Cevallos
  Editorial Alfaomega.

- Manuales Users: 3D Studio Max
  Daniel Venditi
  Editorial MP Ediciones

- A fondo DirectX
  Bradley Bargen, Peter Donelly
  Editorial Microsoft Press

- Genetic Algorithms Library
  MIT
  http://lancet.mit.edu/ga/

- Redes Neuronales: Algoritmos, aplicaciones y técnicas de programación
  James Freeman
  Editorial Adison Wesley

- Neural Network Toolbox
  Haggan, Demut, Beale
  Editorial Mathworks

- Inteligencia Artificial Segunda Edición
  Elaine Rich y Kevin Knight
  Editorial Mc Graw Hill