Implementación del algoritmo Minimax para la toma de decisiones en el juego Tic-Tac-Toe

Bryan Robert Ninamango Arroyo

Desarrollo de Software e Inteligencia Artificial

Lenguaje: C++
Tema: Algoritmos de decisión en juegos
Tipo: Documento técnico de aprendizaje

Enero 2026

1 Introducción

Los juegos por turnos son un género de videojuegos en el que los jugadores se alternan para tomar decisiones y ejecutar acciones, donde cada elección puede modificar por completo el desarrollo de la partida. Cada jugada conduce a un resultado posible: victoria, empate o derrota.

Este proyecto parte de esta idea para desarrollar una inteligencia artificial capaz de analizar distintos escenarios del juego, predecir las mejores y peores jugadas posibles y seleccionar el movimiento óptimo. Para ello, la IA busca maximizar su ganancia o minimizar su pérdida en función de los estados futuros del juego, eligiendo así la mejor decisión en cada turno.

2 Descripción

2.1 Juego Tic-Tac-Toe

Tic-Tac-Toe es un juego por turnos para dos jugadores que se desarrolla en un tablero de 3×3. Cada

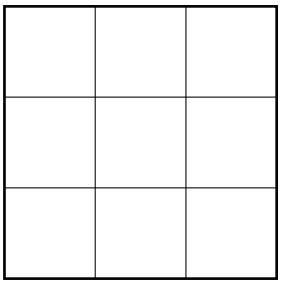

Figure 1: Ejemplo del estado inicial

jugador coloca su símbolo en una casilla vacía del tablero, alternando turnos hasta que se alcanza un estado final del juego. El objetivo del juego es formar una línea de tres símbolos iguales de manera

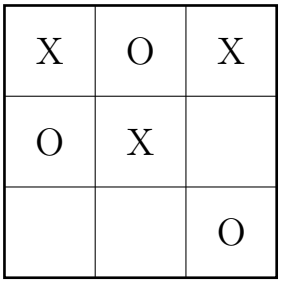

Figure 2: Ejemplo de estado intermedio

horizontal, vertical o diagonal. Un jugador gana la partida cuando logra dicha combinación antes que su oponente. En caso de que el tablero se llene sin que ningún jugador consiga una línea válida, el juego finaliza en empate.

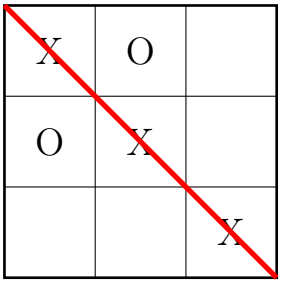

Figure 3: Ejemplo de estado final

2.2 Algoritmo Minimax

Minimax es un algoritmo de toma de decisiones utilizado en la inteligencia artificial, ampliamente aplicado en la teoría de juegos y en el desarrollo de videojuegos. Está diseñado para minimizar la posible pérdida en el peor escenario y, al mismo tiempo, maximizar la ganancia potencial.

1. Maximizando el turno

$$\boxed{\text{Max}(s) = \max_{a \in A(s)} \text{Min}(\text{Result}(s, a))} \tag{1}$$

   donde:

   - $\text{Max}(s)$ es el mayor valor que el jugador MAX puede obtener desde el estado s.
   - $A(s)$ representa el conjunto de acciones posibles desde el estado s.
   - $\text{Result}(s, a)$ es el estado resultante de ejecutar la acción a en el estado s.
   - $\text{Min}(\text{Result}(s, a))$ es el valor que obtiene el jugador MIN desde el estado resultante.

2. Minimizando el turno

$$\boxed{\text{Min}(s) = \min_{a \in A(s)} \text{Max}(\text{Result}(s, a))} \tag{2}$$

   donde $\text{Min}(s)$ es el menor valor que el jugador MAX puede obtener desde el estado s.

3. Estados terminales

$$\boxed{\text{Utility}(s) = \begin{cases} 1 & \text{Si el jugador MAX gana desde el estado } s \\ 0 & \text{Si el juego es un empate del estado } s \\ -1 & \text{Si el jugador MIN gana desde el estado } s \end{cases}} \tag{3}$$

3 Implementación

La implementación del algoritmo Minimax se llevó a cabo en el lenguaje C++, aprovechando su control explícito sobre la memoria y su eficiencia en la ejecución de algoritmos recursivos.

3.1 Representación del Árbol

El problema se modela mediante un árbol, donde cada nodo representa un estado posible del juego.

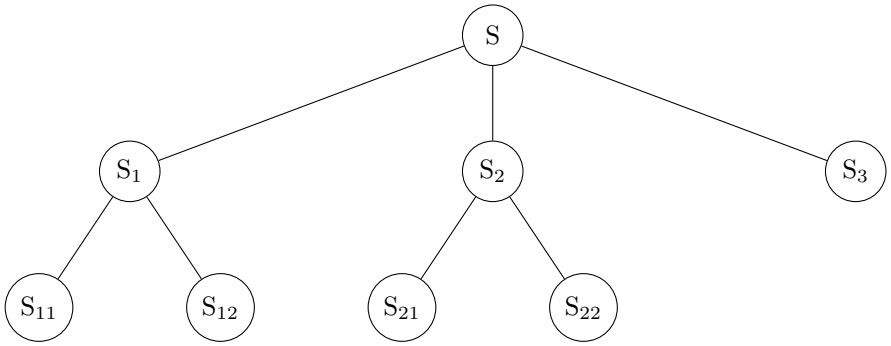

Figure 4: Ejemplo de un árbol de estados

Cada estado del juego se representa mediante un nodo dentro de un árbol de búsqueda. Dicho nodo almacena la siguiente información:

- **state**: Estado actual del juego.
- **action**: Movimiento aplicado para alcanzar este estado.
- **parent**: Puntero al nodo padre dentro del árbol de búsqueda.

- **children**: Conjunto de nodos hijos que representan los estados sucesores.
- **value**: Valor de utilidad asociado al estado, calculado mediante el algoritmo Minimax.

Esta estructura permite reconstruir el camino de decisiones y evaluar cada estado de forma independiente.

3.2 Generación de estados

```
Game result(const Game& state, const Move& action);
```

Genera un nuevo estado del juego aplicando una acción válida sobre un estado existente. Este método evita modificar el estado original, garantizando inmutabilidad durante la búsqueda.

3.3 Detección de estados terminales

```
bool isTerminal(const Game &s);
```

Un estado terminal representa una hoja del árbol de búsqueda. Esto ocurre cuando:

- Uno de los jugadores resulta ganador.
- El juego termina en empate.

3.4 Función de utilidad

```
int utility(const Game &state);
```

Devuelve un valor entero asociado al resultado del juego:

- **Valor positivo**: Victoria del jugador MAX.
- **Valor negativo**: Victoria del jugador MIN.
- **Valor cero**: Empate.

3.5 Expansión del árbol

```
void expand(Node *node);
```

Este método genera todos los posibles estados hijos a partir de un nodo dado, considerando los movimientos disponibles en el estado actual.

La expansión se detiene cuando se cumple alguna de las siguientes condiciones:

- El nodo corresponde a un estado terminal.
- El nodo ya ha sido expandido previamente.

3.6 Algoritmo Minimax

```
int minimax(Node *node);
```

Este método genera todos los posibles estados hijos a partir de un nodo dado, considerando los movimientos disponibles en el estado actual.

La expansión se detiene cuando se cumple alguna de las siguientes condiciones:

- El nodo corresponde a un estado terminal.
- El nodo ya ha sido expandido previamente.

4 Resultados

La inteligencia artificial implementada mediante el algoritmo Minimax presentó un comportamiento consistente y determinista, seleccionando en todo momento el movimiento óptimo para el estado actual del juego.

En todos los escenarios evaluados se observó que:

- La IA evitó estados que conducían a la derrota cuando existía al menos una alternativa válida.
- La IA forzó estados de empate en aquellas situaciones en las que no era posible alcanzar una victoria.
- En estados ganadores, la IA seleccionó movimientos que garantizaban la victoria, independientemente de las decisiones del oponente.

Estos resultados confirman la correcta implementación del algoritmo Minimax sin el uso de técnicas de poda.

5 Conclusiones

La implementación de la inteligencia artificial basada en el algoritmo Minimax permitió validar la eficacia de los métodos clásicos de búsqueda adversaria en juegos por turnos con espacios de estados acotados. A partir del modelado del juego como un árbol de decisiones, fue posible simular partidas completas y seleccionar movimientos óptimos bajo el supuesto de juego racional por parte de ambos jugadores.

Los resultados obtenidos confirman que incluso sin optimizaciones como poda Alpha-Beta, es suficiente para garantizar decisiones correctas y consistentes en juegos de baja a media complejidad. Desde el punto de vista del diseño, la separación entre la lógica del juego y la lógica de decisión facilitó una implementación modular y extensible. Esta arquitectura permite reutilizar el sistema de búsqueda en otros juegos por turnos con mínimos cambios, reforzando la escalabilidad conceptual del enfoque.

En cuanto al rendimiento, se observó que el crecimiento exponencial del árbol de búsqueda constituye la principal limitación del sistema. Esta restricción pone de manifiesto la necesidad de técnicas de optimización y heurísticas cuando se trabaja con juegos de mayor profundidad o factor de ramificación elevado.

Finalmente, este trabajo permitió consolidar conocimientos fundamentales en inteligencia artificial aplicada a videojuegos, incluyendo búsqueda en árboles, evaluación de estados, gestión de memoria dinámica en **C++** y diseño de sistemas de dificultad ajustable. La implementación presentada establece una base sólida para futuras mejoras y optimizaciones, y demuestra la viabilidad del enfoque Minimax como herramienta formativa y práctica en el desarrollo de IA para juegos.

6 Referencias

[1] Corsair. (s. f.). *What is turn-based strategy?* https://www.corsair.com/es/es/explorer/glossary/what-is-turn-based-strategy/

[2] Wikipedia. (s. f.). *Tres en línea.* ttps://es.wikipedia.org/wiki/Tres_en_l

[3] GeeksforGeeks. (2025, 7 de abril). *MiniMax algorithm in artificial intelligence.* https://www.geeksforgeeks.org/artificial-intelligence/mini-max-algorithm-in-artificial-intelligence/

[4] BitBoss. (2020, 22 de octubre). *Algoritmo minimax en 4 minutos* [Video]. YouTube. https://www.youtube.com/watch?v=QJjM7EKDRuc

[5] freeCodeCamp.org. (2019, 7 de agosto). *Simple explanation of the minimax algorithm with tic-tac-toe* [Video]. YouTube. https://www.youtube.com/watch?v=5y2aOZhgq0U

[6] La Geekipedia De Ernesto. (2021, 13 de enero). *Algoritmo minimax y poda alfa-beta* [Video]. YouTube. https://www.youtube.com/watch?v=A1RW7vgtims