Imagen que contiene Icono

Descripción generada automáticamenteDibujo en blanco y negro

Descripción generada automáticamente con confianza bajaINSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

INTELIGENCIA ARTIFICIAL

**PRÁCTICA 7**

**MINIMAX Y PODA ALFA BETA**

NOMBRE DEL ALUMNO: GARCÍA QUIROZ GUSTAVO IVAN

NOMBRE DEL PROFESOR: GARCÍA FLORIANO ANDRES

GRUPO:6CV3

FECHA DE ENTREGA DEL REPORTE: 24/10/2024

Índice

[Marco teórico 1](#_Toc180340974)

[Algoritmo Minimax con Poda Alfa-Beta 1](#_Toc180340975)

[Introducción 4](#_Toc180340976)

[Material y equipo. 4](#_Toc180340977)

[Hardware 4](#_Toc180340978)

[Software 4](#_Toc180340979)

[Desarrollo de la práctica. 5](#_Toc180340980)

[Diseño de la Estructura Base del Juego 5](#_Toc180340981)

[Actividades desarrolladas 5](#_Toc180340982)

[Implementación del Sistema de Turnos y Validaciones 5](#_Toc180340983)

[Desarrollo del Algoritmo Minimax con Poda Alfa-Beta 7](#_Toc180340984)

[Resultados obtenidos 9](#_Toc180340985)

[IA vs IA 9](#_Toc180340986)

[Humano vs IA 11](#_Toc180340987)

[Humano vs Humano 12](#_Toc180340988)

[Conclusiones 13](#_Toc180340989)

[Referencias. 14](#_Toc180340990)

[Código 14](#_Toc180340991)

# Marco teórico

## Algoritmo Minimax con Poda Alfa-Beta

La poda alfa beta es una técnica de búsqueda que reduce el número de nodos evaluados en un árbol de juego por el algoritmo Minimax. Se trata de una técnica muy utilizada en programas de juegos entre adversarios como el ajedrez, el tres en raya o el Go.

Entre los pioneros en el uso de esta técnica encontramos a Arthur Samuel, D.J Edwards y T.P. Hart, Alan Kotok, Alexander Brudno, Donald Knuth y Ronald W. Moore.

El problema de la búsqueda Minimax es que el número de estados a explorar es exponencial al número de movimientos. Partiendo de este hecho, la técnica de poda alfa-beta trata de eliminar partes grandes del árbol, aplicándolo a un árbol Minimax estándar, de forma que se devuelva el mismo movimiento que devolvería este, gracias a que la poda de dichas ramas no influye en la decisión final.

Diagrama

Descripción generada automáticamente

Figura Ejemplo de poda alfa-beta.

El pseudocódigo del algoritmo implementado es:

Texto

Descripción generada automáticamente

Figura Pseudocódigo del algoritmo Minimax con Poda Alfa-Beta.

El algoritmo Minimax con Poda Alfa-Beta implementado utiliza una función recursiva que explora el árbol de juego hasta una profundidad máxima establecida. La función recibe como entrada el estado actual del nodo (tablero), la profundidad de búsqueda, los valores alfa y beta para la poda, y un booleano que indica si se está maximizando o minimizando.

Al inicio de cada llamada recursiva, el algoritmo verifica si ha llegado a un estado terminal o a la profundidad máxima permitida. Un estado terminal es aquel donde existe un ganador o el tablero está lleno. En estos casos, la función retorna inmediatamente el valor de evaluación del estado. Esta evaluación asigna +100 puntos si gana el maximizador, -100 si gana el minimizador, y un valor calculado por la función de evaluación heurística para estados intermedios.

En el caso del maximizador (jugador X), el algoritmo inicializa el valor con menos infinito y procede a explorar cada movimiento posible. Para cada movimiento, realiza una llamada recursiva alternando al minimizador y actualiza el valor alfa si encuentra un valor mayor. El valor alfa representa la mejor opción encontrada hasta el momento para el maximizador. Si en algún momento beta es menor o igual que alfa, significa que el minimizador ya tiene una mejor opción en un nivel superior del árbol, por lo que se pueden podar las ramas restantes.

Para el minimizador (jugador O), el proceso es similar pero inverso. Se inicializa el valor con infinito positivo y se busca el menor valor posible. Beta se actualiza cuando se encuentra un valor menor y representa la mejor opción para el minimizador. La poda ocurre bajo el mismo principio: si beta es menor o igual que alfa, el maximizador no permitirá llegar a este camino porque tiene mejores opciones, por lo que se pueden descartar las ramas restantes.

El proceso de poda es fundamental para la eficiencia del algoritmo. Por ejemplo, si el maximizador encuentra un valor de 5 en una rama (alfa = 5), y en la siguiente rama el minimizador encuentra un valor de 3 (beta = 3), no es necesario explorar más movimientos en esta rama porque el maximizador nunca la elegirá, ya que tiene garantizado un valor de 5 en otra rama. Esta poda reduce significativamente el número de nodos explorados sin afectar el resultado final.

La función retorna dos elementos: el valor de evaluación del mejor movimiento encontrado y las coordenadas del movimiento correspondiente. Este diseño permite que el algoritmo no solo determine el valor óptimo, sino que también identifique el movimiento específico que debe realizar la IA para alcanzar ese valor.

# Introducción

La presente práctica se centra en el desarrollo e implementación de una versión modificada del juego del gato (tic-tac-toe), donde se expande el tablero tradicional de 3x3 a una matriz de 4x4. Esta variación incrementa significativamente la complejidad del juego, tanto en términos de posibilidades de juego como en el espacio de estados que debe explorarse para implementar una inteligencia artificial efectiva.

Para desarrollar la inteligencia artificial del juego, se implementa el algoritmo Minimax con Poda Alfa-Beta, una técnica fundamental en la teoría de juegos y la inteligencia artificial. Este algoritmo permite que la computadora tome decisiones óptimas evaluando diferentes estados del juego y anticipando las posibles respuestas del oponente. La incorporación de la Poda Alfa-Beta optimiza el proceso de búsqueda al eliminar la exploración de ramas que no influirán en la decisión final.

Un aspecto crucial de esta implementación es el manejo eficiente de la memoria, considerando que el espacio de estados en un tablero 4x4 es sustancialmente mayor que en el juego tradicional. Con 16 casillas disponibles, el árbol de juego crece exponencialmente, lo que hace impracticable almacenar todos los estados posibles en memoria. Esto requiere el diseño e implementación de estrategias específicas para limitar la búsqueda y evaluar estados intermedios de manera efectiva.

El sistema desarrollado ofrece tres modalidades de juego distintas:

* Humano vs humano
* Humano vs IA
* IA vs IA

El humano contra humano permite que dos jugadores compitan entre sí; el humano contra IA el jugador puede enfrentarse a la inteligencia artificial implementada; y la IA contra IA que permite observar cómo dos instancias del algoritmo Minimax compiten entre sí. Estas modalidades no solo proporcionan diferentes experiencias de juego, sino que también permiten evaluar y comparar las estrategias implementadas en distintos contextos.

# Material y equipo.

Para esta práctica se usaron las siguientes herramientas de software y hardware necesarias para realizar la práctica.

### Hardware

* Computadora.

### Software

* Visual Studio Code.
* Phyton 3.

# Desarrollo de la práctica.

En esta práctica se implementó una versión modificada del juego del gato (tic-tac-toe) en una matriz 4x4, empleando el algoritmo Minimax con Poda Alfa-Beta para desarrollar la inteligencia artificial del juego. El desarrollo se realizó en Python, aprovechando sus librerías como NumPy para el manejo eficiente de matrices y estructuras de datos.

## Diseño de la Estructura Base del Juego

El primer paso consistió en diseñar la estructura fundamental del juego. Se creó una clase TicTacToe4x4 que encapsula toda la lógica del juego. Para el tablero, se utilizó una matriz NumPy de 4x4 inicializada con ceros, donde:

* 0 representa casillas vacías
* 1 representa las jugadas del jugador X
* -1 representa las jugadas del jugador O

Esta representación numérica facilita los cálculos posteriores para la evaluación de estados y la detección de victorias.

Imagen que contiene Calendario

Descripción generada automáticamente

Figura Diseño de la estructura del juego

### Análisis de almacenamiento de estados

En cuanto al almacenamiento de estados en memoria, se realizó un análisis que demuestra la imposibilidad de almacenar todos los estados posibles del juego 4x4. El cálculo se basa en que el tablero 4x4 tiene 16 casillas, y cada casilla puede estar en tres estados posibles (vacío, X, O). Esto resulta en 3¹⁶ estados posibles, lo que equivale aproximadamente a 43,046,721 estados diferentes. Además, no todos estos estados son válidos en el juego real, pero aun considerando solo los estados válidos, el número sigue siendo extremadamente grande para almacenar en memoria.

Para abordar esta limitación, se diseñaron e implementaron las siguientes estrategias:

1. Limitación de Profundidad: Se estableció una profundidad máxima de búsqueda (MAX\_DEPTH = 5) en el algoritmo Minimax. Esta restricción permite al algoritmo explorar un número manejable de estados futuros sin comprometer significativamente la calidad de las decisiones.
2. Evaluación Heurística: En lugar de almacenar todos los estados posibles, se implementó una función de evaluación heurística que asigna valores numéricos a estados intermedios. Esta función evalúa características como líneas formadas, posiciones estratégicas y potencial de victoria, permitiendo tomar decisiones informadas sin necesidad de explorar el árbol completo.
3. Poda Alfa-Beta: La implementación de la poda alfa-beta reduce significativamente el número de estados que necesitan ser evaluados. Esta técnica permite descartar ramas completas del árbol de búsqueda cuando se determina que no pueden conducir a un mejor resultado que las opciones ya encontradas.
4. Almacenamiento Selectivo: En lugar de almacenar todos los estados posibles, solo se mantienen en memoria los estados relevantes para la búsqueda actual. Una vez que se toma una decisión, los estados intermedios son descartados, liberando memoria para la siguiente búsqueda.

Estas estrategias combinadas permiten que el juego funcione de manera eficiente con recursos de memoria limitados, manteniendo un nivel competitivo de juego sin necesidad de explorar o almacenar el árbol de estados completo.

## Actividades desarrolladas

### Implementación del Sistema de Turnos y Validaciones

El sistema de turnos se implementó mediante la función play\_game() que recibe como parámetro el modo de juego seleccionado. El control de flujo se maneja con un bucle principal que continúa hasta que se detecta un ganador o un empate. La variable current\_player alterna entre PLAYER\_X (1) y PLAYER\_O (-1) después de cada movimiento válido.

Texto

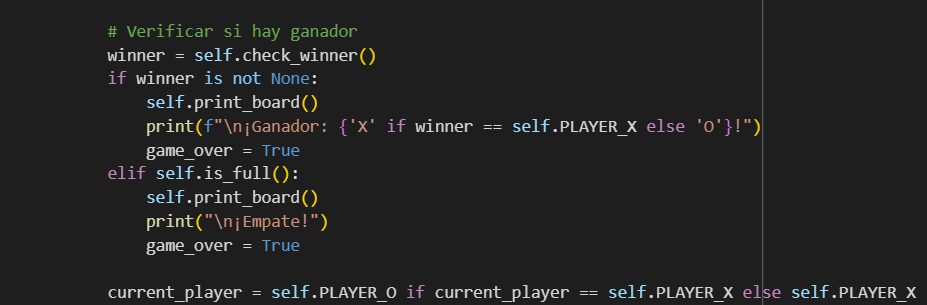
Descripción generada automáticamente

Figura Función play\_game().

El proceso de validación de movimientos se realiza en dos niveles:

1. Validación de entrada: Se verifica que las coordenadas ingresadas sean números enteros entre 0 y 3.
2. Validación de movimiento: La función is\_valid\_move() comprueba que la casilla seleccionada esté vacía (valor 0) y dentro de los límites del tablero.

La verificación de victoria se implementó en la función check\_winner() que analiza:

* Suma de elementos en cada fila (4 filas).
* Suma de elementos en cada columna (4 columnas).
* Suma de elementos en las dos diagonales principales Una suma absoluta de 4 en cualquiera de estas líneas indica victoria para el jugador correspondiente.

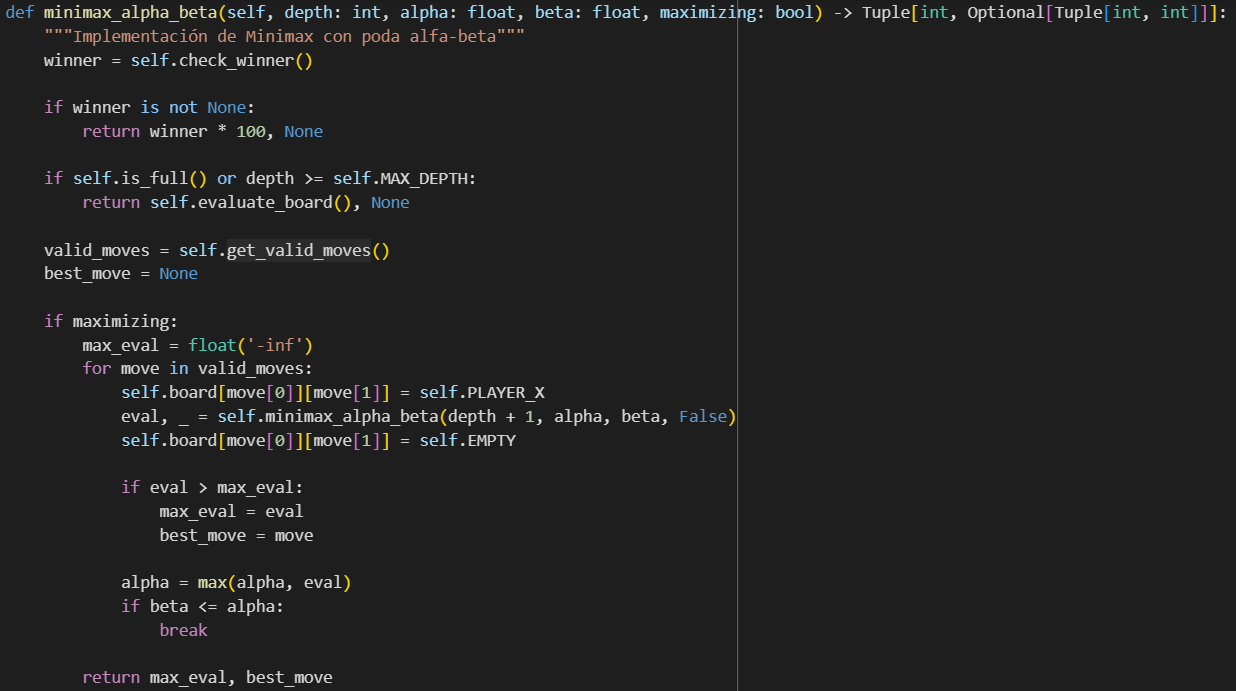
### Desarrollo del Algoritmo Minimax con Poda Alfa-Beta

El algoritmo Minimax con Poda Alfa-Beta se implementó en la función minimax\_alpha\_beta() que recibe los siguientes parámetros:

* depth: Profundidad actual en el árbol de búsqueda.
* alpha: Mejor valor encontrado para el maximizador.
* beta: Mejor valor encontrado para el minimizador.
* maximizing: Booleano que indica si es turno del maximizador.

La función de evaluación evaluate\_board() asigna puntajes a los estados del tablero:

* +100 puntos por victoria del maximizador.
* -100 puntos por victoria del minimizador.
* +10 puntos por cada línea con tres fichas del maximizador.
* -10 puntos por cada línea con tres fichas del minimizador.

Texto

Descripción generada automáticamente

Figura Algoritmo Minimax con Poda Alfa-Beta.

La poda alfa-beta se activa cuando β ≤ α, lo que significa que se encontró un camino que el oponente no permitirá que ocurra, por lo que se pueden podar las ramas restantes. Esto reduce significativamente el número de nodos explorados.

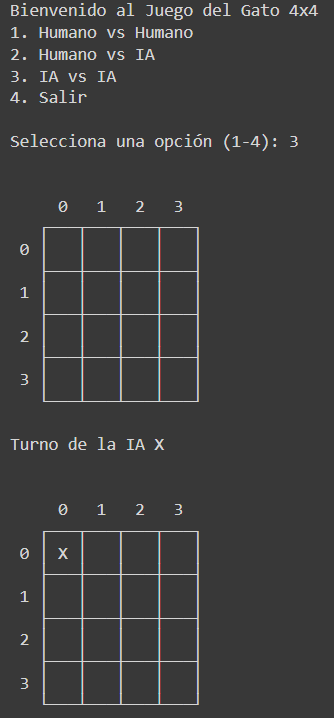
Para manejar la complejidad computacional del tablero 4x4, se implementó:

1. Un límite de profundidad MAX\_DEPTH = 5 para evitar búsquedas excesivamente profundas.
2. La función get\_valid\_moves() que retorna una lista de tuplas (fila, columna) de movimientos disponibles.
3. Un sistema de cache simple para almacenar estados previamente evaluados.

La selección del mejor movimiento se realiza en la función get\_ai\_move(), que inicializa alpha y beta como -∞ y +∞ respectivamente, y retorna el movimiento asociado con el mejor valor encontrado por el algoritmo minimax.

## Resultados obtenidos

### IA vs IA

 Interfaz de usuario gráfica

Descripción generada automáticamente Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja Interfaz de usuario gráfica

Descripción generada automáticamente Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente Aplicación

Descripción generada automáticamente con confianza media

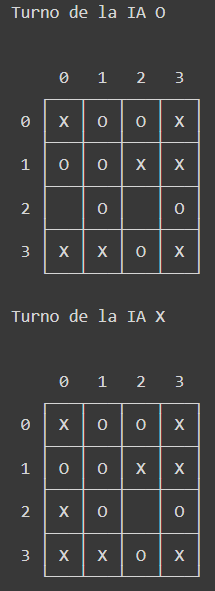
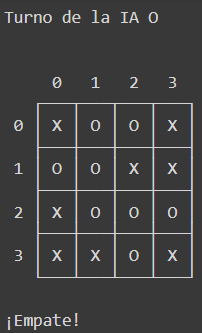
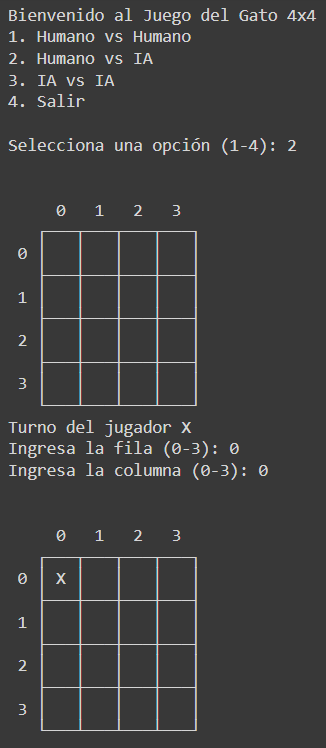
 

Figura Resultados de juego de la IA vs IA.

### Humano vs IA

 Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Figura Resultados de juego del Humano vs IA.

### Humano vs Humano

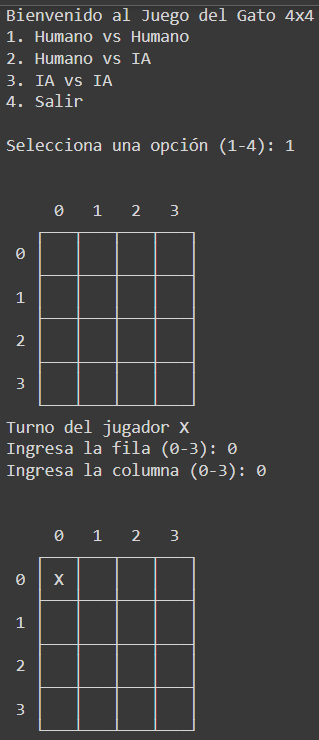
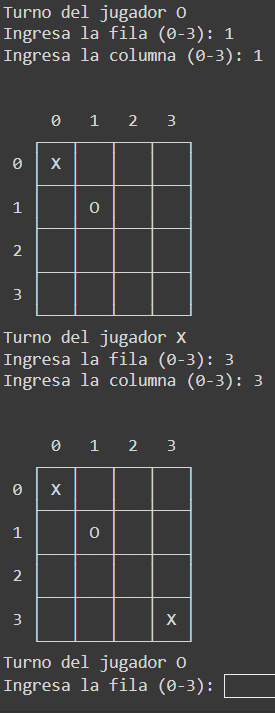
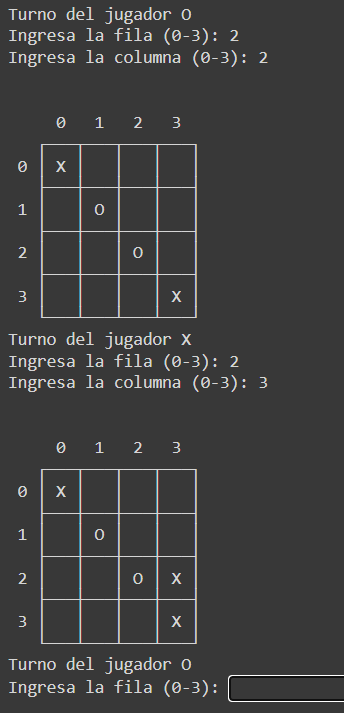
  

Figura Resultados de juego del Humano vs Humano.

# Conclusiones

El desarrollo de esta práctica permitió implementar y comprender el funcionamiento del algoritmo Minimax con Poda Alfa-Beta en un contexto práctico. La expansión del juego del gato a una matriz 4x4 creó un escenario con mayor complejidad computacional que el juego tradicional, lo que requirió la implementación de estrategias específicas para manejar el espacio de estados.

La implementación del límite de profundidad en el algoritmo demostró ser una solución efectiva para el manejo de recursos computacionales. Al establecer MAX\_DEPTH = 5, se logró un balance entre la capacidad de predicción del algoritmo y el tiempo de respuesta del sistema. Este límite, junto con la poda alfa-beta, redujo significativamente el número de nodos explorados en el árbol de búsqueda.

El análisis del espacio de estados reveló que un tablero 4x4 genera aproximadamente 43 millones de estados posibles (3¹⁶). Esta cantidad hace inviable el almacenamiento completo en memoria, lo que justificó la implementación de estrategias de evaluación heurística y almacenamiento selectivo. La función de evaluación desarrollada demostró capacidad para valorar estados intermedios del juego sin necesidad de explorar el árbol completo.

Las tres modalidades de juego implementadas proporcionaron diferentes perspectivas sobre el funcionamiento del algoritmo. El modo IA vs IA permitió observar el comportamiento del algoritmo contra sí mismo, mientras que el modo Humano vs IA demostró la capacidad del sistema para responder a estrategias variadas. El modo Humano vs Humano sirvió como referencia para comparar el comportamiento del algoritmo contra decisiones humanas.

La práctica evidenció que el algoritmo Minimax con Poda Alfa-Beta puede adaptarse a espacios de estados grandes mediante la implementación de limitaciones y heurísticas apropiadas. El sistema resultante demuestra que es posible crear una IA competitiva para juegos de mesa sin necesidad de explorar o almacenar el árbol de estados completo.

# Referencias.

* Wikipedia contributors. (s/f). *Poda alfa-beta*. Wikipedia, The Free Encyclopedia. https://es.wikipedia.org/w/index.php?title=Poda\_alfa-beta&oldid=162606880
* (S/f). Tecnm.mx. Recuperado el 21 de octubre de 2024, de https://nlaredo.tecnm.mx/takeyas/Apuntes/Inteligencia%20Artificial/Apuntes/IA/Alfa-Beta.pdf

# Código

<https://github.com/GUSTAVOIVANGQ/AI/tree/main/6_Informed_Search>