# Proyecto: Tic-Tac-Toe con IA utilizando el algoritmo Minimax

**Descripción General:**

Este proyecto consiste en desarrollar el clásico juego de **Tres en Raya (Tic-Tac-Toe)**, donde los jugadores competirán contra una **inteligencia artificial (IA)** que utiliza el **algoritmo Minimax** para tomar decisiones óptimas. Los estudiantes implementarán tanto la lógica del juego como el algoritmo Minimax, lo que permitirá a la IA buscar la mejor jugada en cada turno. Además, como una extensión opcional, se puede añadir la técnica de **poda alfa-beta** para optimizar el rendimiento de la IA, reduciendo el número de estados posibles que el algoritmo debe evaluar.

**Objetivos del Proyecto:**

1. **Desarrollar el juego de Tic-Tac-Toe**, implementando las reglas básicas, la lógica de juego, y la interfaz gráfica o textual.
2. **Implementar el algoritmo Minimax** para la IA, garantizando que esta siempre juegue de manera óptima.
3. **Optimización del rendimiento de la IA** mediante la técnica de **poda alfa-beta** como extensión opcional.
4. **Refuerzo de conceptos de Programación Orientada a Objetos (POO)**, algoritmos de decisión y optimización.

**Instrucciones Detalladas:**

**1. Diseño del Tablero y Reglas Básicas del Juego**

* Implementar una **clase TicTacToe** que contenga la lógica del juego y el tablero:
  + El **tablero** será una matriz de 3x3, donde cada celda puede contener un valor vacío (''), una **X** o una **O**.
  + **Reglas básicas**:
    - Dos jugadores: uno usando **X** y otro **O**.
    - Los jugadores se alternan haciendo jugadas hasta que uno de ellos consiga tres símbolos iguales en línea (horizontal, vertical o diagonal) o el tablero se llene (empate).
  + Implementar un método dibujar\_tablero() que muestre el estado actual del juego. Se puede hacer tanto en **modo texto** (consola) como con una **interfaz gráfica** utilizando librerías como **Pygame** o **Tkinter**.

**2. Comportamiento del Jugador Humano**

* Implementar la lógica para el **jugador humano**, que será controlado manualmente:
  + El jugador elige una posición en el tablero (por ejemplo, utilizando coordenadas como (1,1)).
  + Validar que la posición elegida sea válida (es decir, que no esté ocupada).
  + Permitir al jugador repetir si escoge una posición no válida.

**3. Implementación del Algoritmo Minimax para la IA**

* El algoritmo **Minimax** es un enfoque de búsqueda en árboles de decisión que simula todas las posibles jugadas del juego para tomar la mejor decisión. Se trata de un algoritmo recursivo que evalúa todas las posibles posiciones del tablero y selecciona la jugada que maximiza las posibilidades de ganar o, al menos, minimiza las pérdidas.
* Implementar una **clase IA\_Minimax** que contenga el siguiente método principal:
  + **minimax(tablero, profundidad, es\_maximizar)**: Este método recorre el árbol de posibles movimientos.
    - Si es el turno del jugador **IA (maximizar)**, el algoritmo buscará la jugada que maximice su posibilidad de ganar.
    - Si es el turno del **jugador humano (minimizar)**, el algoritmo buscará la jugada que minimice las posibilidades de ganar de la IA.
    - Evaluar el estado final del tablero en cada recursión. Si un jugador ha ganado o el tablero está lleno (empate), el algoritmo devolverá un valor de utilidad (1 para victoria de la IA, -1 para victoria del jugador humano, 0 para empate).
* **Lógica de evaluación de jugadas**:
  + Si la IA puede ganar en la siguiente jugada, debe tomarla.
  + Si el jugador humano puede ganar en la siguiente jugada, la IA debe bloquearla.
  + El algoritmo evalúa todas las posibles jugadas futuras y selecciona la que minimice el peor resultado posible para la IA.

**4. Integración del Algoritmo Minimax en el Juego**

* La IA utilizará el algoritmo Minimax cada vez que sea su turno para determinar su jugada óptima.
* Implementar un método jugada\_IA() dentro de la clase TicTacToe que llame al algoritmo Minimax para elegir la mejor jugada posible.

**5. Extensión Opcional: Optimización con Poda Alfa-Beta**

* El algoritmo Minimax puede volverse lento si el árbol de decisiones es muy grande. La **poda alfa-beta** es una mejora que reduce el número de nodos que Minimax necesita evaluar, sin afectar el resultado final.
  + **Alfa** representa el valor máximo que el jugador **maximizar** puede obtener.
  + **Beta** representa el valor mínimo que el jugador **minimizar** puede obtener.
* Implementar la poda en el algoritmo Minimax de la siguiente manera:
  + **minimax\_alfa\_beta(tablero, profundidad, alfa, beta, es\_maximizar)**: En cada recursión, el algoritmo compara el valor actual con alfa y beta, y decide si seguir evaluando el resto de nodos o podarlos (es decir, ignorarlos si no pueden mejorar el resultado).

**6. Visualización del Proceso de Decisión (Opcional)**

* Como una mejora adicional, se puede implementar una **visualización gráfica** del proceso de decisión del algoritmo Minimax.
  + Mostrar en pantalla el árbol de decisiones, con las posibles jugadas evaluadas y los valores asignados por el algoritmo.
  + Esto ayudará a los estudiantes a entender cómo funciona el algoritmo y por qué la IA toma ciertas decisiones.

**Requisitos del Proyecto:**

1. **Implementación del juego de Tic-Tac-Toe**, incluyendo el tablero, las reglas y la mecánica básica del jugador humano.
2. **Algoritmo Minimax** correctamente implementado y funcional, permitiendo que la IA juegue de manera óptima.
3. **Integración del algoritmo Minimax** en el juego para que la IA pueda competir contra el jugador humano.
4. **Extensión opcional**: Implementación de la poda alfa-beta para optimizar el rendimiento del algoritmo.
5. **Interfaz gráfica o textual** para visualizar el tablero y las jugadas.

**Evaluación del Proyecto:**

* **Correcta implementación del juego** (tablero, jugabilidad, interacción del usuario) – 30%
* **Funcionamiento del algoritmo Minimax** para la IA – 30%
* **Optimización con poda alfa-beta (opcional)** – 20%
* **Interfaz gráfica y experiencia de usuario** – 10%
* **Claridad y organización del código** (uso de POO, modularidad) – 10%