INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Ingeniería en Sistemas Computacionales

Inteligencia Artificial

Profesor:

Andrés García Floriano

Ejercicio:

MiniMax y poda Alfa-Beta

Alumnos:

Carmona Santiago Yeimi Guadalupe

Hernández Suárez Diego Armando

Flores Osorio Adolfo Ángel

Grupo:6CV3

Contenido

No se encontraron entradas de tabla de contenido.

Introducción

El proyecto consiste en la implementación de una versión ampliada del juego del gato (tic-tac-toe) en una matriz de 4x4. Esta variación utiliza el algoritmo **Minimax con poda alfa-beta** para optimizar la toma de decisiones en las jugadas de la inteligencia artificial (IA). El objetivo es ofrecer tres modalidades de juego: Humano vs humano, Humano vs IA, y IA vs IA.

Marco teórico

Juego del Gato

El juego del gato, conocido también como "tic-tac-toe", es un juego de estrategia donde dos jugadores, alternando turnos, colocan su símbolo (X o O) en un tablero de 3x3 con el objetivo de conseguir tres símbolos en línea, ya sea horizontal, vertical o diagonalmente. Pero en el desarrollo de esta práctica, nuestro tablero será de 4x4.

Algoritmo Minimax

El algoritmo **Minimax** es una técnica utilizada para tomar decisiones óptimas en juegos de dos jugadores con información perfecta, como ajedrez o el gato. Este algoritmo explora todos los movimientos posibles de los jugadores hasta el estado final, maximizando las ganancias del jugador actual y minimizando las del oponente.

En el contexto del gato 4x4, el espacio de búsqueda se incrementa considerablemente, y aquí es donde entra en juego la poda alfa-beta.

Poda Alfa-Beta

La **poda alfa-beta** es una optimización sobre el algoritmo Minimax. Permite evitar la evaluación de ciertas ramas del árbol de búsqueda cuando ya se ha encontrado

un camino mejor en una rama anterior. Esto reduce significativamente el tiempo de ejecución del algoritmo, manteniendo el mismo resultado que el Minimax tradicional.

Validación de Entradas

Cuando los jugadores humanos interactúan con un juego, es común que se introduzcan entradas incorrectas o fuera de los límites aceptados. En este proyecto, es fundamental validar esas entradas para garantizar que el flujo del juego sea coherente y que las jugadas se realicen dentro del tablero definido.

Desarrollo de la práctica

Se define un tablero de 4x4, donde las casillas vacías están representadas por el símbolo '-', el jugador humano usa 'X' y la IA utiliza 'O'.

```python
import math

# Constantes
JUGADOR_X = 'X'
JUGADOR_O = 'O'
VACIO = '-'

# Crear un tablero vacío 4x4
def crear_tablero():
    return [[VACIO for _ in range(4)] for _ in range(4)]
```

Se implementa una función para mostrar el estado actual del tablero y otra para validar si el tablero está lleno o si un jugador ha ganado.

```python
# Imprimir el tablero
def imprimir_tablero(tablero):
    for fila in tablero:
        print(' '.join(fila))
    print()

# Verificar si el tablero está lleno
def tablero_lleno(tablero):
    return all(celda != VACIO for fila in tablero for celda in fila)
```

El juego finaliza cuando un jugador consigue cuatro símbolos consecutivos en una fila, columna o diagonal.

```python
# Función para verificar si un jugador ha ganado (4 en línea)
def verificar_ganador(tablero, jugador):
    # Verificar filas, columnas y diagonales
    for i in range(4):
        # Filas
        if all(tablero[i][j] == jugador for j in range(4)):
            return True
        # Columnas
        if all(tablero[j][i] == jugador for j in range(4)):
            return True

    # Diagonales
    if all(tablero[i][i] == jugador for i in range(4)):
        return True
    if all(tablero[i][3-i] == jugador for i in range(4)):
        return True

    return False

```

La función evaluar_tablero devuelve un valor basado en el estado actual del tablero. Si un jugador gana, devuelve 100 o -100 dependiendo del jugador. Si no, se considera un empate o estado intermedio.

```python
# Función heurística (cuenta líneas parciales)
def evaluar_tablero(tablero):
    if verificar_ganador(tablero, JUGADOR_X):
        return 100
    elif verificar_ganador(tablero, JUGADOR_O):
        return -100
    else:
        return 0  # Empate o estado no decisivo
```

`

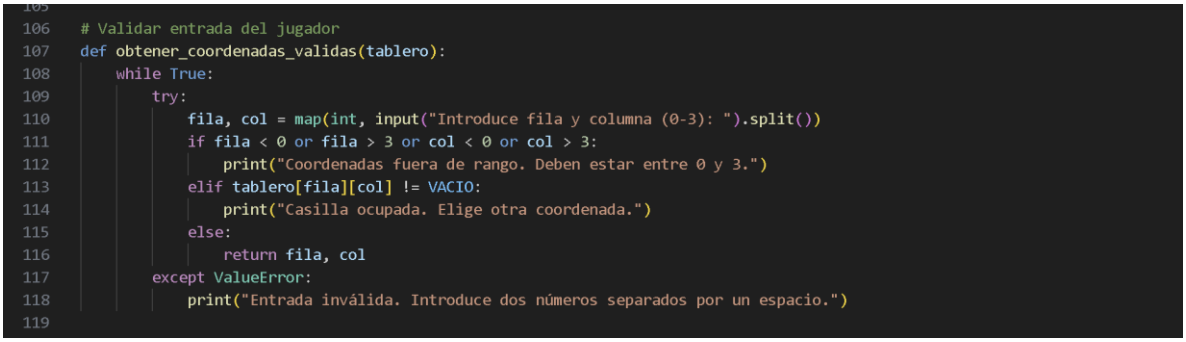
El algoritmo Minimax se aplica junto con la poda alfa-beta para optimizar el rendimiento. Se fija un límite de profundidad para evitar explorar todos los estados posibles.

```python
# Minimax con poda alfa-beta
def minimax(tablero, profundidad, alfa, beta, es_maximizador):
    evaluacion = evaluar_tablero(tablero)

    # Caso terminal: victoria, derrota o empate
    if evaluacion == 100 or evaluacion == -100 or tablero_lleno(tablero):
        return evaluacion

    if profundidad == 0:  # Limitar la profundidad
        return evaluar_tablero(tablero)  # Evaluar el estado intermedio

    if es_maximizador:
        max_eval = -math.inf
        for i in range(4):
            for j in range(4):
                if tablero[i][j] == VACIO:
                    tablero[i][j] = JUGADOR_X
                    evaluacion = minimax(tablero, profundidad-1, alfa, beta, False)
                    tablero[i][j] = VACIO
                    max_eval = max(max_eval, evaluacion)
                    alfa = max(alfa, evaluacion)
                    if beta <= alfa:
                        break
        return max_eval
    else:
        min_eval = math.inf
        for i in range(4):
            for j in range(4):
                if tablero[i][j] == VACIO:
                    tablero[i][j] = JUGADOR_O
                    evaluacion = minimax(tablero, profundidad-1, alfa, beta, True)
                    tablero[i][j] = VACIO
                    min_eval = min(min_eval, evaluacion)
                    beta = min(beta, evaluacion)
                    if beta <= alfa:
                        break
        return min_eval

```

Se implementan tres modalidades: **Humano vs humano**, **Humano vs IA**, y **IA vs IA**, cada una gestionando los turnos y verificando el ganador o el empate.

```python
# Modos de juego
def humano_vs_humano():
    tablero = crear_tablero()
    jugador_actual = JUGADOR_X
    while True:
        imprimir_tablero(tablero)
        print(f"Turno de {jugador_actual}")
        fila, col = obtener_coordenadas_validas(tablero)
        tablero[fila][col] = jugador_actual
        if verificar_ganador(tablero, jugador_actual):
            imprimir_tablero(tablero)
            print(f"Jugador {jugador_actual} gana!")
            break
        elif tablero_lleno(tablero):
            imprimir_tablero(tablero)
            print("Empate!")
            break
        jugador_actual = JUGADOR_O if jugador_actual == JUGADOR_X else JUGADOR_X
```

La función obtener_coordenadas_validas asegura que el usuario ingrese coordenadas válidas dentro del rango y que las casillas no estén ocupadas.



```python
# Validar entrada del jugador
def obtener_coordenadas_validas(tablero):
    while True:
        try:
            fila, col = map(int, input("Introduce fila y columna (0-3): ").split())
            if fila < 0 or fila > 3 or col < 0 or col > 3:
                print("Coordenadas fuera de rango. Deben estar entre 0 y 3.")
            elif tablero[fila][col] != VACIO:
                print("Casilla ocupada. Elige otra coordenada.")
            else:
                return fila, col
        except ValueError:
            print("Entrada inválida. Introduce dos números separados por un espacio.")

```

Conclusión

El desarrollo de este proyecto fue enriquecedor, ya que permitió aplicar conceptos avanzados de inteligencia artificial, como el **Minimax** y la **poda alfa-beta**, en un contexto práctico de juegos. Al trabajar con un tablero más grande (4x4), el desafío aumentó, dado que el número de estados posibles crece exponencialmente. La poda alfa-beta resultó esencial para reducir el tiempo de cálculo y mantener el rendimiento del juego.

Uno de los aspectos más importantes fue implementar una validación robusta de las entradas, garantizando que el juego no se interrumpiera por errores del usuario. Este tipo de validaciones es crucial en proyectos interactivos, donde se espera que el sistema sea capaz de manejar situaciones imprevistas.

Este proyecto podría expandirse en el futuro, añadiendo más niveles de dificultad para la IA, utilizando redes neuronales en lugar de Minimax, o implementando una versión online para múltiples jugadores.