Instituto Politécnico Nacional

Escuela Superior de Computo

Ingeniería en Sistemas Computacionales

Sexto semestre

Grupo: 6CV3

Inteligencia Artificial

Laboratorio 7: Minimax y Poda Alfa Beta

Presentado por:

Flores Vicencio Marco Isaí

Pérez González Jesús

Reyes Núñez Sebastián

Fecha: 17/10/2024

Indice

Introducción

Técnica Minimax

Minimax es un algoritmo de toma de decisiones utilizado en juegos de suma cero con dos jugadores, como el ajedrez o el gato. El objetivo es encontrar la mejor jugada posible asumiendo que el oponente también jugará de manera óptima.

Características principales:

1. Búsqueda en profundidad: Explora el árbol de posibilidades hasta llegar a un estado terminal o una profundidad predefinida.
2. Alternancia de roles: En cada nivel del árbol, se alternan los roles de "maximizador" y "minimizador".
3. Propagación de valores: Los valores se propagan desde las hojas hasta la raíz del árbol.

El algoritmo funciona de la siguiente manera:

- El jugador maximizador busca la jugada que le dé el mayor valor posible.
- El jugador minimizador busca la jugada que le dé el menor valor posible al oponente.

Poda Alfa-Beta

La poda Alfa-Beta es una mejora del algoritmo Minimax que reduce significativamente el número de nodos evaluados en el árbol de búsqueda sin afectar el resultado final.

Características principales:

1. Mantiene dos valores: alfa (mejor valor para el maximizador) y beta (mejor valor para el minimizador).
2. Poda ramas: Elimina ramas del árbol que no pueden influir en la decisión final.
3. Mejora la eficiencia: Permite una búsqueda más profunda en el mismo tiempo.

Funcionamiento:

- Si un jugador encuentra una movida que es peor que una movida previamente examinada, esa rama se puede podar.
- Alfa representa el mínimo puntaje que el jugador maximizador puede garantizar.
- Beta representa el máximo puntaje que el jugador minimizador puede garantizar.

Juego del Gato en Matriz 4x4

El juego del gato en una matriz 4x4 es una variante del juego clásico que se juega en un tablero más grande.

Características:

1. Tablero: Una cuadrícula de 4x4, con 16 casillas en total.
2. Jugadores: Dos jugadores, generalmente representados por "X" y "O".
3. Objetivo: Alinear cuatro de sus símbolos en línea recta (horizontal, vertical o diagonal).
4. Turnos: Los jugadores alternan turnos, colocando su símbolo en una casilla vacía.

Diferencias con la versión 3x3:

- Mayor complejidad: Más posiciones posibles y estrategias más elaboradas.
- Más opciones de victoria: Cuatro en línea en lugar de tres.
- Juego más largo: Típicamente requiere más movimientos para alcanzar un resultado.
- Menor probabilidad de empate: El tablero más grande ofrece más oportunidades de victoria.

Desafíos para la IA:

1. Árbol de juego más grande: Aumenta la complejidad computacional.
2. Evaluación de posiciones: Requiere considerar más patrones y posibilidades.
3. Balance entre profundidad y amplitud: La búsqueda debe equilibrar la profundidad de análisis con la amplitud de opciones.

Desarrollo

Para el desarrollo de esta práctica, se represento al tablero del gato como la clase Gato4x4 representa el estado del juego y contiene métodos para hacer movimientos, verificar el ganador y obtener movimientos válidos.

```python
class Gato4x4:
    def __init__(self):
        self.tablero = [' ' for _ in range(16)]
        self.jugador_actual = 'X'

    def hacer_movimiento(self, posicion: int) -> bool:
        if self.tablero[posicion] == ' ':
            self.tablero[posicion] = self.jugador_actual
            self.jugador_actual = 'O' if self.jugador_actual == 'X' else 'X'
            return True
        return False

    def obtener_movimientos_validos(self) -> List[int]:
        return [i for i, casilla in enumerate(self.tablero) if casilla == ' ']
```

Para validar la victoria de un jugador, se verifica que una fila columna o diagonal no se haya llenado del mismo valor, después regresa el carácter del ganador

```python
def verificar_ganador(self) -> Optional[str]:
    # Verificar filas y columnas
    for i in range(4):
        if self.tablero[i*4] == self.tablero[i*4+1] == self.tablero[i*4+2] == self.tablero[i*4+3] != ' ':
            return self.tablero[i*4]
        if self.tablero[i] == self.tablero[i+4] == self.tablero[i+8] == self.tablero[i+12] != ' ':
            return self.tablero[i]

    # Verificar diagonales
    if self.tablero[0] == self.tablero[5] == self.tablero[10] == self.tablero[15] != ' ':
        return self.tablero[0]
    if self.tablero[3] == self.tablero[6] == self.tablero[9] == self.tablero[12] != ' ':
        return self.tablero[3]

    if ' ' not in self.tablero:
        return 'Empate'

    return None
```

La función minimax implementa el algoritmo Minimax con Poda Alfa-Beta.

```python
def minimax(estado: Gato4x4, profundidad: int, alpha: float, beta: float, es_maximizador: bool) -> Tuple[int, float]:
    ganador = estado.verificar_ganador()
    if ganador == 'X':
        return None, 1
    elif ganador == 'O':
        return None, -1
    elif ganador == 'Empate':
        return None, 0

    if profundidad == 0:
        return None, 0

    if es_maximizador:
        mejor_valor = -math.inf
        mejor_movimiento = None
        for movimiento in estado.obtener_movimientos_validos():
            nuevo_estado = Gato4x4()
            nuevo_estado.tablero = estado.tablero.copy()
            nuevo_estado.jugador_actual = estado.jugador_actual
            nuevo_estado.hacer_movimiento(movimiento)
            _, valor = minimax(nuevo_estado, profundidad - 1, alpha, beta, False)
            if valor > mejor_valor:
                mejor_valor = valor
                mejor_movimiento = movimiento
            alpha = max(alpha, mejor_valor)
            if beta <= alpha:
                break
        return mejor_movimiento, mejor_valor
    else:
        mejor_valor = math.inf
        mejor_movimiento = None
        for movimiento in estado.obtener_movimientos_validos():
            nuevo_estado = Gato4x4()
            nuevo_estado.tablero = estado.tablero.copy()
            nuevo_estado.jugador_actual = estado.jugador_actual
            nuevo_estado.hacer_movimiento(movimiento)
            _, valor = minimax(nuevo_estado, profundidad - 1, alpha, beta, True)
            if valor < mejor_valor:
                mejor_valor = valor
                mejor_movimiento = movimiento
            beta = min(beta, mejor_valor)
            if beta <= alpha:
                break
        return mejor_movimiento, mejor_valor
```

Definimos que con cada estado que se evalúa es 1 si el jugador con la “x” puede ser ganador y como un -1 para el estado en el que el jugador ‘o’ podría ser ganador.

Se itera a través de la profundidad del árbol de estados y dependiendo en que nivel se encuentra va a maximizar o minimizar y en cada caso siempre se va comparando y guardando la mejor opción para maximizar o minimizar, esto se hace recursivamente hasta llegar a un punto de paro, ya sea por la técnica de poda Alpha beta o por haber llegado al nivel de profundidad 5, este es el nivel máximo de búsqueda que hará la IA antes de tomar una decisión.

En este punto solo nos queda definir los modos de juego que tendremos primero el juego humano-humano

```python
def jugar_humano_vs_humano():
    juego = Gato4x4()
    while True:
        juego.imprimir_tablero()
        print(f"Turno del jugador {juego.jugador_actual}")
        movimiento = int(input("Ingrese la posición (0-15): "))
        if juego.hacer_movimiento(movimiento):
            ganador = juego.verificar_ganador()
            if ganador:
                juego.imprimir_tablero()
                if ganador == 'Empate':
                    print("¡Es un empate!")
                else:
                    print(f"¡El jugador {ganador} ha ganado!")
                break
        else:
            print("Movimiento inválido, intente de nuevo.")
```

Mientras no se haya llenado el tablero seguiremos pidiendo posiciones a cada jugador validando en cada una si no se ha encontrado a un ganador

Lo mismo para el modo de juego de IA-humano

```python
def jugar_humano_vs_ia():
    juego = Gato4x4()
    while True:
        juego.imprimir_tablero()
        if juego.jugador_actual == 'X':
            print("Turno del jugador humano (X)")
            movimiento = int(input("Ingrese la posición (0-15): "))
            if not juego.hacer_movimiento(movimiento):
                print("Movimiento inválido, intente de nuevo.")
                continue
        else:
            print("Turno de la IA (O)")
            movimiento, _ = minimax(juego, 5, -math.inf, math.inf, False)
            juego.hacer_movimiento(movimiento)

        ganador = juego.verificar_ganador()
        if ganador:
            juego.imprimir_tablero()
            if ganador == 'Empate':
                print("¡Es un empate!")
            else:
                print(f"¡El jugador {ganador} ha ganado!")
            break
```

Para el caso de la IA se pone el límite comentado anteriormente de 5 niveles de profundidad en el árbol de decisiones para evitar tiempos uy largos de ejecución.

Finalmente el caso de IA-IA en donde cada jugador tomara siempre la mejor decisión posible dentro de los 5 niveles de profundidad, por lo que lo esperado es que siempre termine en empate

```python
def jugar_ia_vs_ia():
    juego = Gato4x4()
    while True:
        juego.imprimir_tablero()
        print(f"Turno de la IA ({juego.jugador_actual})")
        movimiento, _ = minimax(juego, 5, -math.inf, math.inf, juego.jugador_actual == 'X')
        juego.hacer_movimiento(movimiento)

        ganador = juego.verificar_ganador()
        if ganador:
            juego.imprimir_tablero()
            if ganador == 'Empate':
                print("¡Es un empate!")
            else:
                print(f"¡La IA {ganador} ha ganado!")
            break
```

Funcionamiento

Humano VS humano

```
Turno del jugador O
Ingrese la posición (0-15): 10
O| |X|X
-------
 |O| |X
-------
 | |O|
-------
 | | |
Turno del jugador X
Ingrese la posición (0-15): 11
O| |X|X
-------
 |O| |X
-------
 | |O|X
-------
 | | |
Turno del jugador O
Ingrese la posición (0-15): 15
O| |X|X
-------
 |O| |X
-------
 | |O|X
-------
 | | |O
¡El jugador O ha ganado!
```


```
Turno del jugador O
Ingrese la posición (0-15): 10
O| |X|X
-------
 |O| |X
-------
 | |O|
-------
 | | |
Turno del jugador X
Ingrese la posición (0-15): 11
O| |X|X
-------
 |O| |X
-------
 | |O|X
-------
 | | |
Turno del jugador O
Ingrese la posición (0-15): 15
O| |X|X
-------
 |O| |X
-------
 | |O|X
-------
 | | |O
¡El jugador O ha ganado!
```

Humano VS IA

```
Bienvenido al juego del Gato 4x4
1. Humano vs Humano
2. Humano vs IA
3. IA vs IA
Seleccione el modo de juego: 2
 | | |
-------
 | | |
-------
 | | |
-------
 | | |
Turno del jugador humano (X)
Ingrese la posición (0-15): 0
X| | |
-------
 | | |
-------
 | | |
-------
 | | |
Turno de la IA (O)
X|O| |
-------
 | | |
-------
 | | |
-------
 | | |
Turno del jugador humano (X)
Ingrese la posición (0-15): 2
X|O|X|
-------
 | | |
-------
 | | |
-------
 | | |
Turno de la IA (O)
X|O|X|O
-------
 | | |
-------
 | | |
-------
 | | |
```

```
Turno del jugador humano (X)
Ingrese la posición (0-15): 10
X|O|X|O
-------
X|O|O|X
-------
 | |X|
-------
 | | |
Turno de la IA (O)
X|O|X|O
-------
X|O|O|X
-------
 |O|X|
-------
 | | |
Turno del jugador humano (X)
Ingrese la posición (0-15): 11
X|O|X|O
-------
X|O|O|X
-------
 |O|X|X
-------
 | | |
Turno de la IA (O)
X|O|X|O
-------
X|O|O|X
-------
O|O|X|X
-------
 | | |
Turno del jugador humano (X)
Ingrese la posición (0-15): 15
X|O|X|O
-------
X|O|O|X
-------
O|O|X|X
-------
 | | |X
Turno de la IA (O)
X|O|X|O
-------
X|O|O|X
-------
O|O|X|X
-------
O| | |X
¡El jugador O ha ganado!
```

IA VS IA

```
Bienvenido al juego del Gato 4x4
1. Humano vs Humano
2. Humano vs IA
3. IA vs IA
Seleccione el modo de juego: 3
 | | |
-------
 | | |
-------
 | | |
-------
 | | |
Turno de la IA (X)
X| | |
-------
 | | |
-------
 | | |
-------
 | | |
Turno de la IA (O)
X|O| |
-------
 | | |
-------
 | | |
-------
 | | |
Turno de la IA (X)
X|O|X|
-------
 | | |
-------
 | | |
-------
 | | |
Turno de la IA (O)
X|O|X|O
-------
 | | |
-------
 | | |
-------
 | | |
```

```
Turno de la IA (O)
X|O|X|O
-------
X|O|X|O
-------
X|X|O|
-------
O| | |
Turno de la IA (X)
X|O|X|O
-------
X|O|X|O
-------
X|X|O|X
-------
O| | |
Turno de la IA (O)
X|O|X|O
-------
X|O|X|O
-------
X|X|O|X
-------
O|O| |
Turno de la IA (X)
X|O|X|O
-------
X|O|X|O
-------
X|X|O|X
-------
O|O|X|
Turno de la IA (O)
X|O|X|O
-------
X|O|X|O
-------
X|X|O|X
-------
O|O|X|O
¡Es un empate!
```

Conclusión

Esta implementación demuestra cómo los conceptos teóricos de la inteligencia artificial pueden aplicarse efectivamente en un problema práctico. La práctica no solo logró crear un juego funcional, sino que también proporcionó importantes insights sobre el manejo de la complejidad computacional y la implementación de algoritmos de búsqueda en situaciones con recursos limitados.

La experiencia resalta la importancia de encontrar balances adecuados entre diferentes aspectos del desarrollo: profundidad vs. velocidad, complejidad vs. usabilidad, y optimización vs. mantenibilidad. Estos aprendizajes son valiosos no solo para el desarrollo de juegos, sino para cualquier proyecto que involucre IA y optimización de recursos. En donde resaltaron aspecto como: Algoritmos de IA que nos dio Comprensión práctica de Minimax Importancia de la optimización mediante poda Alfa-Beta y manejo de restricciones de recursos en el diseño de Software, importancia de la planificación inicial, valor de la modularización y balance entre funcionalidad y rendimiento, además de teoría de Juegos: Complejidad de juegos aparentemente simples, importancia de la evaluación de estados y estrategias para manejar espacios de búsqueda grandes

Referencias

Deep Learning for Tic-Tac-Toe: A Comparative Analysis"

- IEEE Conference on Artificial Intelligence and Gaming “Comparación entre métodos tradicionales y aprendizaje profundo”

Efficient State Space Search in Game Playing Algorithms"

- ACM Computing Surveys

GeeksforGeeks: "Mini-Max Algorithm in Game Theory"

- Tutorial detallado con implementaciones
  URL: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/