Informe: Implementación de Jugador Inteligente para Hex

Autor: Abel Ponce González
Grupo: 311
2025

Índice

1. Implementación del Jugador Inteligente **2**
1.1. Algoritmo Minimax con Poda Alfa-Beta 2

2. Heurísticas Utilizadas **2**
2.1. Evaluación Estratégica 2
2.2. Versión Simplificada 2

3. Priorización de Movimientos **3**
3.1. Criterios de Selección 3

4. Optimizaciones Clave **3**
4.1. Poda Alfa-Beta 3
4.2. Tabla de Transposición 3

1. Implementación del Jugador Inteligente

1.1. Algoritmo Minimax con Poda Alfa-Beta

El núcleo del jugador utiliza el algoritmo **Minimax** con optimizaciones de poda alfa-beta para reducir el espacio de búsqueda. La implementación incluye:

- **Profundidad adaptable**: Ajusta la profundidad de búsqueda según movimientos restantes.
- **Tabla de transposición**: Evita recálculos mediante caché de estados evaluados.
- **Función de evaluación heurística**: Combina múltiples factores posicionales.

Listing 1: Esqueleto de Minimax

```
def minimax(self, board, depth, alpha, beta, maximizing):
    if depth == 0 or board.check_winner():
        return self.evaluate_board(board)
    # Poda alfa-beta y seleccion de movimientos
    return best_score
```

2. Heurísticas Utilizadas

2.1. Evaluación Estratégica

La función de evaluación en `board_evaluation.py` considera:

- **Distancia al borde**: Calculada mediante A* (Jugador 1: horizontal, Jugador 2: vertical).
- **Control del centro**: Bonificación por cercanía al centro del tablero.
- **Conectividad**: +20 puntos por cada ficha adyacente propia.

$$\text{Score} = (\text{Distancia\_oponente} - \text{Distancia\_jugador}) \times 10 + \text{Bono\_centro}$$

2.2. Versión Simplificada

Para evaluaciones rápidas:

```
def evaluate_board_simplified(self, board):
    score = 0
    for i, j in board.player_positions[self.player_id]:
        score += j * 2 if self.player_id == 1 else i * 2  # Direccionalidad
    return score
```

3. Priorización de Movimientos

3.1. Criterios de Selección

En `move_selection.py`, los movimientos se ordenan mediante:

1. **Victoria inmediata**: Máxima prioridad si conecta bordes.
2. **Bloqueo**: +1000 puntos si evita victoria oponente.
3. **Bifurcaciones**: +100 puntos por 2+ fichas adyacentes.
4. **Proximidad al centro**: $(N - \text{distancia\_al\_centro}) \times 2$.

Listing 2: Priorización

```
def prioritize_moves_simplified(self, board, player_id):
    moves = sorted(moves, key=lambda x: x[0], reverse=True)
    return [move for _, move in moves]
```

4. Optimizaciones Clave

4.1. Poda Alfa-Beta

Reduce el árbol de búsqueda descartando ramas subóptimas:

- **Poda Alpha**: Ignora nodos peores que el mejor valor actual (maximizador).
- **Poda Beta**: Similar para el minimizador.

4.2. Tabla de Transposición

Almacena estados evaluados para reutilizar resultados:

```
transposition_table = {
    board_hash: (score, depth)
}
```