ESDI III

Universidad Panamericana Campus Aguascalientes

Cristian Aragón Salazar - 0250005

Quixo Bot (Github)

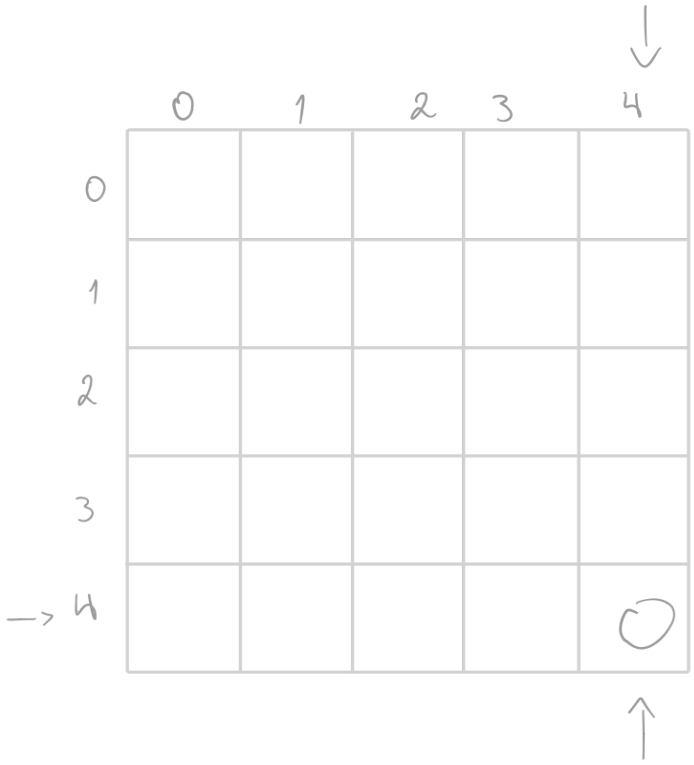
Aún más información en el README.md de github

IDEA

Pensamiento de la idea

1. En este proyecto empecé pensando en hacerlo con alguna búsqueda de la que habíamos visto el parcial anterior, pero al momento de que presentaron el tema de minimax y empezar a investigar más sobre él, vi que era un algoritmo justo para las condiciones de este proyecto, en el que teníamos dos jugadores, es por turnos, tu puntuación afecta a la del oponente y podemos conocer los diferentes estados.
2. En un principio pensaba hacerlo solo con los diferentes tableros, pero conforme iba avanzando fui viendo que era mejor tener los nodos y el árbol tal cual hecho, por lo que decidí implementarlo a manera de árbol con nodos definidos.

3. Usé el tablero que se ponía en el template, que lo represento de la siguiente manera:



Problemas

1. Al momento de pensar en cómo realizar los movimientos me di cuenta que podía hacerlo modificando directamente la matriz y desplazando los elementos que ya tenía en ella, por lo que así lo realicé, pero después tenía que limitar los movimientos a los movimientos permitidos, y tuve varias ideas, hacer una función que revisara si ese movimiento era permitido, pero al momento de tener que llamar a cada movimiento, la función tendría que revisar si el movimiento es válido, lo que haría que se perdiera tiempo en ese llamado, por lo que vi que cada movimiento solo tiene 11 fichas disponibles para mover, lo que no representaba algo complicado de conocer, así que lo pasé a un arreglo en la cual se tiene las piezas que se pueden mover en cada diferente movimiento.
2. Otro problema que se me presentó fue que el bot pensaba el movimiento en mucho tiempo, por lo que decidí implementar la poda Alpha Beta y cambiarle la profundidad al árbol para que solo sea de 2, por lo que ahora mi bot ya tomaba su decisión en menos de un segundo, lo que estaba especificado.
3. El último problema que se me presentó fue la parte de llevar consistencia en el match, es decir mantener y actualizar el estado del juego de manera coherente y sin errores, por lo que hice uso de nodos para poder llevar un mejor registro de todo el match y así

encapsular el estado del tablero, el movimiento realizado y las conexiones entre nodos en el árbol de juego.

FUNCIONAMIENTO

quixo_bot.py:

Clase *GameNode*:

En esta clase se cuenta con varios métodos, entre los cuales primero está *add_child,* el cual añade un hijo a la lista de hijos del nodo actual, después se encuentra *is_terminal,* en la cual se verifica si el nodo es terminal, es decir, que ya no tiene hijos. Luego se tiene el método *evaluate,* en el cual se evalúa el estado del tablero y se le asigna un puntaje. Y por último, el método de *evaluate_line*, el cual evalúa una línea del tablero, ya sea en fila, columna o diagonal y asigna un puntaje en base a la cantidad de símbolos del bot y los espacios vacíos.

Clase *GameTree*:

En esta clase se tienen los métodos *build_tree* y *expand_node,* en el primero se construye el árbol de juego hasta una profundidad dada, y en el segundo se expande un nodo generando sus hijos y evaluando sus estados si se alcanza la profundidad máxima o se detecta una condición de victoria.

Clase *AlphaBeta:*

Se tienen métodos *max_value* y *min_value,* en el caso de *max_value* se calcula el valor máximo de un nodo y en el de *min_value,* el valor mínimo de un nodo.

Clase *QuixoBot:*

Esta es la clase más extensa, hago un resumen de esta a continuación:

1. play_turn(self, board):
    - Realiza un turno del bot.
    - Construye un árbol de juego hasta una cierta profundidad.
    - Utiliza el algoritmo alfa-beta para seleccionar el mejor movimiento.
    - Aplica el movimiento seleccionado al tablero y devuelve el nuevo estado del tablero.
2. Métodos de Movimiento:
    - move_right(self, board, row, col, end_col=4): Mueve una pieza hacia la derecha.
    - move_left(self, board, row, col, end_col=0): Mueve una pieza hacia la izquierda.
    - move_up(self, board, row, col, end_row=0): Mueve una pieza hacia arriba.

- move_down(self, board, row, col, end_row=4): Mueve una pieza hacia abajo.
- Estos métodos aseguran que solo se realicen movimientos válidos y actualizan el tablero en consecuencia.

3. print_board(self, board=None):
    - Imprime el estado actual del tablero en un formato tabular fácil de leer.
4. reset(self, symbol):
    - Reinicia el tablero y establece el símbolo del bot, permitiendo empezar un nuevo juego.
5. is_winner(self, board, symbol):
    - Verifica si un símbolo ha ganado el juego evaluando todas las filas, columnas y diagonales del tablero.
6. is_full(self, board):
    - Verifica si el tablero está lleno, es decir, si no hay celdas vacías.
7. generate_moves(self, board, symbol):
    - Genera una lista de todos los movimientos válidos para el símbolo dado.
8. is_valid_move(self, board, direction, row, col, symbol):
    - Verifica si un movimiento en una dirección específica es válido.
9. apply_move(self, board, move, symbol):
    - Aplica un movimiento al tablero y devuelve el nuevo estado del tablero, utilizando métodos de movimiento específicos.

Heurística

evaluate_line

- Bot ganando:
    - bot_count == 5: Retorna 1000 si el bot tiene una línea completa.
    - opp_count == 5: Retorna -1000 si el oponente tiene una línea completa.
- Casi completando la línea:
    - bot_count == 4 and empty_count == 1: Retorna 50 si el bot tiene cuatro símbolos en línea y una celda vacía.
    - opp_count == 4 and empty_count == 1: Retorna -50 si el oponente tiene cuatro símbolos en línea y una celda vacía.
- Tres en línea:
    - bot_count == 3 and empty_count == 2: Retorna 10 si el bot tiene tres símbolos en línea y dos celdas vacías.
    - opp_count == 3 and empty_count == 2: Retorna -10 si el oponente tiene tres símbolos en línea y dos celdas vacías.

- Dos en línea:
    - bot_count == 2 and empty_count == 3: Retorna 5 si el bot tiene dos símbolos en línea y tres celdas vacías.
    - opp_count == 2 and empty_count == 3: Retorna -5 si el oponente tiene dos símbolos en línea y tres celdas vacías.
- Uno en línea:
    - bot_count == 1 and empty_count == 4: Retorna 1 si el bot tiene un símbolo en línea y cuatro celdas vacías.
    - opp_count == 1 and empty_count == 4: Retorna -1 si el oponente tiene un símbolo en línea y cuatro celdas vacías.
- Ninguna condición favorable:
    - Retorna 0 si ninguna de las condiciones anteriores se cumple.