Jugador de Hex

Rodrigo Pino
Adrian Portales
C512

Jugador de Hex utilizando Minimax con Alpha-Beta Prunning

Implementación

Heurística implementada en coolcow_player.py

Alpha-Beta Prunning implementado en coolcow_minimax.py

Juego

El juego consiste en un tablero de NxN donde cada celda tiene forma hexagonal. En cada casilla puede haber una pieza blanca representada por `W`, una pieza negra `B` o la casilla vacía representada por `.`.

Para explorar los posibles estados del juego se tiene un DAG donde cada *nodo* representa uno de esos posible estado del juego y la raíz es el estado actual en el que se encuentra el juego (el tablero está vacío al comienzo del juego).

Las *aristas* que unen a otros nodos (y por ende a nuevos estados de juego) son transiciones válidas entre posibles estados del juego. Una transición válida consiste cuando el jugador correspondiente pone una ficha de su color en una casilla vacía, generando un nuevo estado del juego. Como los estados del juego son únicos y no se pueden levantar piezas del tablero entonces las aristas son dirigidas.

Un *camino* es una secuencia de nodos del árbol enlazados por aristas cuyo último nodo corresponde con un estado terminal del juego para alguno de los dos jugadores. Se puede ver también como una secuencia de estados del juego hasta que ocurre un ganador, donde solo ocurren transiciones válidas.

Una *jugada* es el acto de cambiar de un estado del juego actual a otro válido. Correspondería a moverse desde el nodo raíz del árbol a algún nodo directamente descendiente de este.

Un *jugador* verifica los posibles estados del juego y realiza una *jugada*.

Heurística

Como heurística se analiza en cada jugada el camino más corto necesario para que un jugador gane, donde un camino es la cantidad de piezas necesarias a poner para unir los bordes correspondientes del tablero.

El camino tiene costo *c* donde *c* representa la cantidad de piezas que faltan por posicionar.

Desde el punto de vista del jugador blanco, la calidad del tablero queda determinada por la resta entre la cantidad mínima de fichas necesarias por el jugador negro y el jugador blanco para obtener la victoria respectivamente. Sería: *blackMinPath - whiteMinPath*. Es análogo para el jugador negro: *whiteMinPath - blackMinPath*.

Para calcular le camino más corto se crea un nodo ficticio enlazado con todas las casillas de cada fronteras del tablero (izquierda y derecha para el jugador blanco; superior e inferior para el jugador negro). No se enlazan con el nodo ficticio las casillas de la frontera ocupadas por una pieza de color opuesto.

Después, se busca el camino de costo mínimo entre los nodos ficticios que unen fronteras del tablero opuestas (fronteras que dependen del jugador en cuestión) utilizando `UCS`:

- Caminar a una celda donde haya una casilla ocupada por una ficha del mismo color que el jugador tiene costo 0
- Costo 1 cuando esta vacía.
- No se puede caminar por casillas ocupadas por el jugador opuesto.

Alpha-Beta Prunning

Se modifica la lógica de minimax adicionando *fail-hard Alpha-Beta Prunning*.

Fail-hard obliga al valor de retorno de la función a estar acotado por α y β.

Modificación

En *coolcow_minimax.py* se añaden a las funciones *maxplay* y *minplay* los parámetros α y β, iniciados en $-\infty$ y ∞ respectivamente.

En *maxplay* después de minimizar una rama del nodo actual, si se encuentra una tal que el valor es mayor que β se deja de analizar.

En *minplay* después de maximizar una rama del nodo actual, si se encuentra una tal que su valor es menor que el α actual, se deja de analizar.

Observaciones

Después de aplicar *Alpha-Beta Prunning* se encontró un aumento importante en el performance del jugador *coolcow_minimax* pues disminuyó enormemente el tiempo de análisis de jugadas sin perderse calidad.