Relatório do EP5 -- Hex

Por Pedro Pereira, 9778794

Introdução

Esse relatório é *compacto*, logo não irá apresentar o jogo ou o desafio imposto, e apenas como o meu algoritmo resolve os problemas do EP.

O meu algoritmo era para funcionar de uma maneira muito mais complexa, porém devido às restrições de tempo de execução em 0.3s e em tempo de entrega do EP, as funções foram implementadas mas não utilizadas ao seu máximo. Um exemplo disso é a *AlphaBetaPruneMax* e *Min*, que teriam sido utilizadas para uma busca de movimentações muito maior, entretanto apenas pus profundidade 1 nelas porque, senão, o programa não rodava consistentemente abaixo de 0.3s.

Dentro do "strategy.c", você irá achar as táticas que eu utilizei para julgar o tabuleiro e perceber se ele é vantajoso ou se o oponente é que tem a vantagem. Meu algoritmo não julga jogadas individuais, mas sim o tabuleiro inteiro. Isso ocorre pois, para a busca de movimentações Alpha-Beta, é necessário que se julgue o tabuleiro como um todo.

Táticas desenvolvidas

Com um pouco de conhecimento prévio e bastante pesquisa do jogo, percebi várias táticas úteis no jogo.

- **Ponte**
  Uma ponte é uma conexão virtual entre duas peças, na qual não há jeito de impedir a conexão delas. Isso ocorre pois há duas maneiras de conectar as mesmas duas peças. Caso o oponente decida tentar bloquear uma delas, ainda é possível conectar usando o outro caminho, desde que esse seja jogado imediatamente.
- **Cadeia**
  Uma cadeia é uma série de pontes e conexões normais, que formam uma *cadeia* longa. É muito importante a manutenção dessas cadeias, pois são o único jeito de ganhar - conectando uma cadeia de um lado a outro do tabuleiro.
- **Posições críticas** Posições críticas são as posições que mais dão vantagem no tabuleiro. Geralmente posições centrais, que dão aberturas para expansões em qualquer parte do tabuleiro, e bloqueiam bastante o caminho do oponente.

Funções pivotais

As funções mais importantes do EP estão descritas nessa seção.

hasWon

Essa função utiliza de uma fila (queue.h) para fazer uma breadth-first search no tabuleiro, na cor escolhida. O algoritmo verifica se há peças na primeira linha/coluna que são da cor desejada, e insere na fila os seus vizinhos. A partir daí, enquanto a fila não estiver vazia, ela adiciona os vizinhos de mesma cor não visitados de cada peça, e assim por diante até que todos os caminhos possíveis sejam visitados ou até uma peça ter um valor de 13 na linha/coluna. Retorna 1, portanto, se há um caminho até uma peça de valor 13.

AlphaBetaPruneMin e AlphaBetaPruneMax

Essas duas funções são parte da busca de qual movimentação realizar no tabuleiro. Segue uma busca alpha-beta[1], na qual analisa todas as jogadas possíveis de se fazer e, para cada uma delas, analisa a melhor jogada do oponente, e para cada uma delas, analisa a sua melhor jogada, e assim por diante. O algoritmo tenta achar a jogada em que o oponente consegue causar o menor dano, ou seja, maximizar a jogada a se fazer para minimizar a jogada do adversário.

"strategy.c" bridge

Checa se há vizinhos de pontes para cada peça e, se sim, como está a situação da ponte. Caso haja uma peça do inimigo, indica com alguns pontos negativos que a ligação entre as duas peças deve ser feita imediatamente. Caso haja duas peças do inimigo, marca como completamente bloqueada, dando ainda menos pontos. Já para uma ponte virtual (ainda não conectada) ainda mantida, dá pontos positivos, e para uma ponte conectada, ainda mais pontos.

"strategy.c" dfsbridge

Utiliza de uma depth-first search para procurar a maior cadeia de elementos que cada jogador tem. Marca como visitada a peça inicial, e aplica recursivamente uma busca por vizinhos, levando consigo um inteiro contendo o tamanho da cadeia até o momento. Devolve um valor diretamente associado com o tamanho da cadeia.

Considerações: Por que o algoritmo, no final, ficou 'fraco'?

Como dito na introdução, o algoritmo utiliza de táticas para julgar o tabuleiro, para rodar um algoritmo de busca de movimentações que, no final, não foi completamente implementado para não exceder o tempo de busca de movimentos. Isso faz com que o algoritmo só se preocupe no que ele está fazendo, e não olha para as jogadas que o oponente pode fazer. Talvez, se houvesse mais tempo para desenvolvimento do EP, eu conseguiria implementar uma busca um pouco mais eficiente ou que utiliza mais memória para achar a jogada em menor tempo, utilizando uma hash-table, por exemplo.