Universidade Federal do Rio Grande do Sul
Instituto de Informática
Departamento de Informática Aplicada

INF01048 - Fundamentos de IA
Relatório do jogo HEX

Professor: Luís Otávio

Mairo Pedrini - 2285/01-0
Marcos Paulo Berteli Slomp - 2390/01-0
Paulo Sérgio Morandi Júnior - 2767/01-1

21 de janeiro de 2004

Introdução

A proposta deste trabalho da disciplina é desenvolver um jogo com Inteligência Artificial, com o intuito de treinar e aproximar os alunos nos conceitos básicos e aplicações do algoritmo MINIMAX em jogos de dois jogadores. O jogo proposto foi o HEX, criado, inicialmente, por um poeta e matemático, Piet Hein e, posteriormente, "reinventado"por John Nash. O jogo, embora com regras muito simples, apresenta diversas características lógicas e matemáticas fascinantes (não há possibilidade de empate, existem algoritmos que sempre vencem para tabuleiros até uma determinada dimensão) e exige um alto nível de estratégia por parte dos jogadores, além de ter despertado o interesse de inúmeros matemáticos e interessados no assunto - e, o mais curioso, não trata-se de um jogo de grande tradição e é bastante recente (1940-1950). A idéia é muito simples: cada jogador possui dois lados paralelos no tabuleiro, ele deve, então, conectá-los, usando uma seqüência de peças, colocadas alternadamente entre os participantes do jogo.

Algoritmos Utilizados

Minimax

A versão utilizada do Minimax foi a mesma discutida em aula. Segue abaixo o algoritmo utilizado:

```
struct RetornoMinimax
{
    valor;    //Melhor valor da função de avaliação;
    caminho;  //Caminho percorrido para chegar no mentre oselhor valor;
}

MIN_ESTÁTICA //valor mínimo da função de avaliação
PROFUNDIDADE //profundidade da árvore do minimax

germov( posição, jogador )
{
  //Calcula todos os movimentos possíveis para o jogador a
  //partir da posição passada como parâmetro
}

funçãoAvaliação( posição, jogador );
//Descrita detalhadamente na próxima seção

profundoSuficiente( posição, profundidade)
{
  //Verifica se a profundiade da Árvore foi atingida
}

RetornoMinimax MINIMAX(posição, profundidade, jogador)
```

```
1- se ( profundoSuficiente( posição, profundidade ) )
   {
     ReturnMinimax.valor = funçãoAvaliação( posição, jogador )
     ReturnMinimax.caminho = null;

     return ReturnMinimax;
   }
   senão
   {
      sucessores = germov( posição, jogador )
   }
2- se ( sucessores == null ) //Vazio
   {
     ReturnMinimax.valor = funçãoAvaliação(posição,jogador);
     ReturnMinimax.caminho = null;

     return ReturnMinimax;
   }
   senão
   {
       melhorValor = MIN_ESTÁTICA;
       para cada elemento suc de sucessores
       {
          ReturnMinimaxSuc = MINIMAX(suc, PROFUNDIDADE+1, oposto(jogador));
          novo_valor = -ReturnMinimaxSuc.valor;
          se ( novo_valor > melhorValor )
          {
             melhorValor = novoValor;
             melhorCaminho = suc + ReturnMinimaxSuc.caminho;
          }
       }
   }
3- ReturnMinimax.valor = melhorValor;
   ReturnMinimax.caminho = melhorCaminho;
   return ReturnMinimax;
```

Funcão de Avaliação

A nossa primeira função de avaliação levava em conta um cálculo simples: ela percorria o tabuleiro e verificava o número de colunas (ou linhas) não ocupadas pelo jogador em questão; então, subtraia esse total pelo número de linhas (ou colunas) não ocupadas por peças adversárias. Obviamente, embora não muito inteligente, as jogadas com três níveis no minimax saiam em um baixíssimo tempo de resposta, mesmo sem o corte alfa-beta. A partir dessa função de avaliação, procuramos estendê-la para que ela pudesse levar em consideração a existência de **conexões virtuais** (figura 1) (peças vazias no tabuleiro que podem conectar outras peças, ou conjunto de peças, mesmo com o oponente tendo a vantagem de jogar primeiro) nas condições do tabuleiro. Entretanto, mesmo com essa melhoria, o jogo não apresentou grandes melhorias em sua qualidade de jogada e tivemos que participar da primeira parte do campeonato

com ele.

Decidimos, então, no período do recesso, estudar uma técnica promissora de implementar uma função de avaliação para HEX, baseada em associação de resistores. Nesta técnica, à cada casa do tabuleiro era associada uma resistencia, que podia ser 0 se a casa estivesse ocupada pelo jogador, infinita se a casa fosse ocupada pelo jogador adversário e 1 se a casa estivesse vazia. Assim, para cada duas casas adjacentes do tabuleiro, associa-se uma resistencia, que é a soma (associação em série) das resistencias de cada uma das casas. A avaliação para um jogador, com esta técnica, é calcular a resistencia equivalente do circuito.

A fonte de desempenho desta técnica é fundamentada pelas leis de Kirchhoff para as correntes, que nos afirma que a corrente entre elétrica (e, portanto, fixada uma tensão, a resistencia) sempre procura o menor caminho entre dois pontos. Ora, este menor caminho é encontrado calculando-se a resistencia equivalente de todo o circuito.

Para tanto, implementamos uma estrutura de grafo, capaz de armazenar o circuito que representa o tabuleiro, bem como as duas regras clássicas para associação de resistores em série e em paralelo. Entretanto, apenas estas duas regras não foram o suficiente pois, após simplificadas todas as associações série e paralelo de resistores, a função chegava à um circuito complexo, contendo básicamente aquilo que, na teoria da análise de circuitos, são conhecidos como **conexões** π e **conexões T**. Após alguma pesquisa, encontramos um teorema que provava a equivalência entre os dois circuitos, juntamente com fórmulas para converter de um para outro. Se usadas para as conexões corretas, tais conversões permitiriam a simplificação total do circuito.

Entretanto, não existem regras ou heurísticas conhecidas (ou documentadas, nossas pesquisas não encontraram nenhuma) para encontrar tal seqüência de conversões. Sendo assim, fizemos alguns testes empíricos com as seqüências mas, em geral, a função entrava em *loop*. Paralelamente à estes esforços para calcular a resistencia equivalente, já planejávamos uma forma de otimizar o cálculo de seu resultado: ao invés de trabalharmos diretamente com os valores das resistências, trabalhariamos com valores literais (variáveis), sendo o resultado uma função da resistência de cada casa do tabuleiro. Assim, o cálculo da resistência seria apenas a avaliação da função para um estado do jogo. Ainda em cima desta otimização, trabalhamos em um sistema de *caching* de resultados parciais de alguns termos da função: só recalcularíamos os termos que dependessem de variáveis (casas) modificadas. Entretanto, como não conseguimos fazer o cálculo da resistência equivalente, tais otimizações nao puderam ser testadas.

Devido a impossibilidade de implementação por nossa parte dessa proposta, estávamos totalmente sem tempo e então utilizamos nossa velha função de avaliação na segunda parte do campeonato. Adicionamos algumas qualidades à heurística, mas demonstrou piores resultados na prática, ao participarmos da segunda etapa.

A Função de Avaliação Implementada

Incialmente, criamos uma classe que representa o **estado do tabuleiro**. Conforme as peças são colocadas, os valores das células do tabuleiro vão mudando, cada jogador com um código associado à representação da peça como célula do tabuleiro.

Além disso, quando uma peça é colocada no tabuleiro, a classe tabuleiro

verifica a existência de peças virtualmente conectadas e marca aquelas células com um código especial.

Duas células disjuntas são virtualmente conectáveis se existirem células vazias próximas a elas que, mesmo com o adversário jogando primeiro, ainda é possível estabelecer a conexão entre as células na jogada seguinte. A figura 1 ilustra a idéia das peças virtualmente conectadas. Os círculos são as peças que podem gerar as conexões virtuais e os hexágonos cheios representam peças fisicamente colocadas no tabuleiro (pertencentes a um mesmo jogador). Mesmo que o adversário jogue em alguma delas, ainda será possível, na jogada seguinte, conectar as peças supostamente "ameaçadas"pelo adversário.

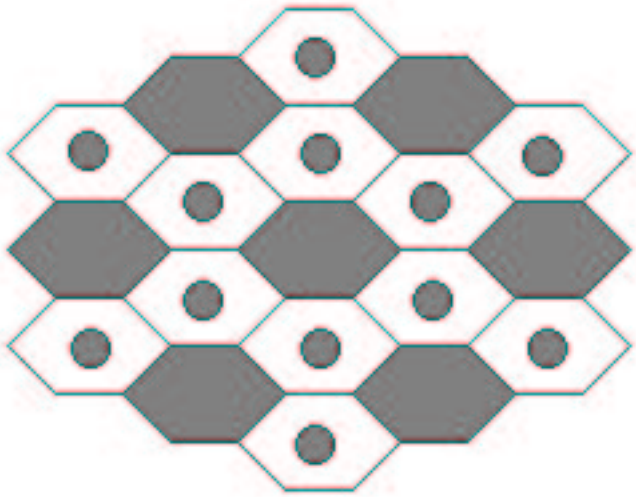

Figura 1: Conexões Virtuais.

A função de avaliação, então, percorre essas células e segue identificando, para o jogador em questão, quantas colunas (ou linhas) possuem células ocupadas FISICAMENTE ou VIRTUALMENTE por ele e, então, subtrai esse total do número máximo de colunas/linhas do tabuleiro. Depois, realiza o mesmo processo para as peças do jogador adversário. Por fim, subtrai os totais obtidos e gera o número correspondente àquela avaliação: quanto menor, melhor. Sempre que o MINIMAX requisita a colocação de uma peça no tabuleiro, a peça e sua vizinhança direta são empilhadas, antes de atualizar os valores do estado do tabuleiro. Conforme o MINIMAX vai retornando as chamadas, ele requisita ao estado do tabuleiro que remova as peças e, conseqüentemente, ele desempilha as antigas vizinhanças, recuperando os estados prévios da árvore do MINIMAX, evitando a sobreposição ou eliminação indevida de informações de estados prévios da árvore.

Conclusão

A proposta do jogo foi muito interssante - e uma implementação de um jogo é algo muito divertido. Entretanto, visto a grande perda de tempo que nos ocorreu na tentativa frustrada de implementar uma heurística avançada, ficamos impossibilitados de elaborar uma função de avaliação realmente poderosa. Encontramos o código fonte de um jogo HEX inteiro, com uma função de avaliação **bastante** forte e competitiva e com código bem comentado (não estamos lembrado ao certo do nome, mas é open source através da GPL), mas preferimos tentar elaborar a nossa própria função. Embora não tenhamos criado um jogo muito competitivo, consideramo-nos aptos e muito bem qualificados nas técnicas de abordagem algoritmica do MINIMAX e corte Alfa-Beta. O jogo HEX também é muito trabalhoso, visto a sua grande gama de possibilidades de jogada, principalmente em tabuleiros muito grandes (como o 11x11) e também acabamos nos preocupando com o tempo de resposta da avaliação, talvez até mais do que a própria qualidade da avaliação em si.