Universidade Federal de Santa Catarina
INE5633 - Sistemas Inteligentes
Professor: Elder Rizzon Santos
Alunos: **Raphael Martins e Thiago Mohr**

Trabalho sobre Métodos de busca - Segunda Parte

Introdução

Este trabalho está relacionado com a implementação do jogo *Ligue 4* com a utilização do algoritmo MiniMax como método de busca e a aplicação de heurísticas para o melhoramento de desempenho. Esta segunda parte deveria apresentar a implementação completa do jogo, bem como a do algoritmo MINIMAX, de heurísticas e da função de utilidade. No entanto, não conseguimos concluir como gostaríamos esta implementação.

O motivo da não implementação foi a dificuldade de definir os estados seguintes à uma jogada. Tais estados deveriam compor a árvore que que seria utilizada no algoritmo MINIMAX. As jogadas do computador estão definidas de forma aleatória.

Utilidade

A função de utilidade foi implementada da seguinte forma:

Realiza-se uma leitura do tabuleiro, que é implementado como uma matriz de Peças. No momento em que se encontra uma Peça do tipo avaliado (1 para o jogador ou 2 para o computador), avalia-se seus vizinhos e conta-se o número de vizinhos adjacentes do mesmo tipo em uma mesma direção. A utilidade é calculada pela potência de 10 elevado ao tamanho da sequência somado de um valor fixo. A utilidade será positiva se o tipo avaliado foi o do computador, caso contrário, será negativa.

Estratégias de Avaliação

A cada casa selecionada, o programa deve verificar a vizinhança da casa selecionada. Idealmente, o raio de verificação deve ser de quatro a partir da posição selecionada.

Caso seja identificada uma sequência com 4 peças do mesmo jogador, é definido como um nodo Folha

Sequências

A avaliação das sequências deve levar em consideração o número de espaços vazios em cada extremidade. Uma sequência que possui os dois próximos vizinhos livres deve valer mais que aquela de mesmo tamanho e apenas o próximo vizinho livre. Também, sequências fechadas devem receber valores nos muito mais baixos que os demais.

MinMax

O algoritmo MinMax tem como propósito fazer o computador escolher a jogada que será mais favorável a si. Ele faz isso avaliando a árvore de jogadas, construída a partir de como estará o tabuleiro de acordo com as jogadas que os dois jogadores fazem, alternadamente. Ele escolhe qual jogada será a mais favorável para si quando é a sua vez de jogar, mas também imagina que o oponente fará a melhor jogada possível, e escolhe este caso (o pior possível) quando avalia a possível jogada do oponente. Desta forma é possível fazer a jogada não só tentando completar o seu jogo, mas também impedindo que o oponente ganhe.

Alpha-beta

A poda *alpha-beta* é uma técnica aplicada ao algoritmo MinMax para aumentar a sua eficiência. Durante a análise dos filhos de um determinado nó, quando verifica-se que um deles já é o mais favorável o possível, ou o menos favorável, no caso de se estar analisando a jogada do oponente, os outros nós são descartados pois sua análise é irrelevante.

Implementação

Para a implementação do trabalho a linguagem escolhida foi *Java*, por já ter sido a linguagem utilizada para o primeiro trabalho e por ser a linguagem de maior familiaridade da dupla. Foram implementadas as seguintes classes:

- Heurística - para atribuir pontuações as jogadas, e calcular a heurística utilizada.
- Main - método principal da implementação, onde a matriz inicial é criada.
- MiniMax - classe que faz a análise dos arredores da peça jogada.
- Nodo - onde o jogo acontece, são inseridas as peças no tabuleiro.
- Peca - informando se é uma peça do jogador humano ou jogador máquina.

Anexos

```
public int[] getSequencias(int[] ultimaPosicao, Nodo nodo, int jogador){
        listaSequencias[0]= this.analisaColunaPecaTopo(ultimaPosicao, nodo, jogador);
        listaSequencias[1]= this.analisaLinhaPecaTopo(ultimaPosicao, nodo, jogador);
        listaSequencias[2]= this.analisaDiagDirPecaTopo(ultimaPosicao, nodo, jogador);
        listaSequencias[3]= this.analisaDiagEsqPecaTopo(ultimaPosicao,nodo, jogador);
        return listaSequencias;
}

private int analisaDiagEsqPecaTopo(int[] ultimaPosicao, Nodo nodo, int jogador) {
        int quantidadeSequencia = 1;
        for (int linha = ultimaPosicao[0]; linha <= 0; linha--) {
                for (int coluna = ultimaPosicao[1]; coluna <= 0; coluna--) {
                        if(linha >= 0 && linha<= 6 && coluna >= 0 && coluna <= 7){
                                if(nodo.estado[linha][coluna] == jogador){
                                        quantidadeSequencia++;
                                } else{
                                        return quantidadeSequencia;
                                }
                        }
                }
        }
        return quantidadeSequencia;
}
```

```
public int[] calculaHeuristica(int[] posicoes, Nodo nodo, int jogador){
        int[] listaSequenciasJogador = getSequencias(posicoes,nodo, 1);
        int pontComp = 0;
        int pontJog = 0;
        for (int i = 0; i < listaSequencias.length; i++) {
                if(jogador == 1){
                        if (listaSequenciasJogador[i] == 4){
                                pontJog += 10 * 4;}
                        else{
                                if (listaSequenciasJogador[i] == 3){
                                        pontJog += 10 * 3;
                                }
                                else{
                                        if (listaSequenciasJogador[i] == 2){
                                                pontJog += 10 * 2;
                                        } else{
                                                pontJog += 10;
                                        }
                                }
                        }
                }
                if(jogador == 2){
                        if (listaSequenciasJogador[i] == 4){
                                pontComp += 10 * 4;
                        }
                        else{
                                if (listaSequenciasJogador[i] == 3){
                                        pontComp += 10 * 3;
                                }
                                else{
                                        if (listaSequenciasJogador[i] == 2){
                                                pontComp += 10 * 2;
                                        }
                                        else{
                                                pontComp += 10;
                                        }
                                }
                        }
                }
        }
        int[] heuristica = {pontJog, pontComp};
```

Referências

http://www.inf.ufrgs.br/~engel/data/media/file/inf01048/jogos.pdf

https://web.fe.up.pt/~eol/IA/IA0809/APONTAMENTOS/Alunos_MiniMax.pdf

https://www.youtube.com/watch?v=ceU9sNFaSM8

http://www.professores.uff.br/mquinet/07_IA.pdf