UFSC - Universidade Federal de Santa Catarina Igor Althoff Vidal
INE 5430 - Inteligência Artificial Fabíola Maria Kretzer
Prof. Elder Rizzon Santos

Trabalho 1: Métodos de busca

Descrição

Esta tarefa tem como objetivo simular a inteligência artificial para o jogo Gomoku, também conhecido com 5 em linha. Utilizando das ferramentas apresentadas nas aulas de inteligência artificial, tais como: cálculo da heurística, cálculo da função utilidade, o algoritmo de busca adversária MiniMax, a otimização com poda α e β, dentre outros conceitos.

Modelo dos estados

Como nos jogos vistos em aula, será necessário modelar em forma de grafo, sendo cada vértice um estado do tabuleiro no jogo, ou seja, uma representação das peças no tabuleiro de jogo. Já as arestas serão as transições de um estado do tabuleiro para outro em apenas uma jogada. Todas as arestas terão custo de uma unidade, pois é somente a ação de colocar a peça no tabuleiro. Com o grafo modelado, poderá ser aplicado o algoritmo MiniMax com podas α-β para fazer a busca da melhor jogada.

Estruturas de dados

Como citado anteriormente, cada vértice deverá armazenar um tabuleiro de tamanho 15X15, para isso será utilizado duas listas, a primeira sendo as posições das jogadas do computador e a outra do humano, mapeando a coordenada para um único valor. Lembrando que uma posição pode estar apenas em uma única lista.

Heurística e Utilidade

Como o jogo é baseado em qual jogador faz uma sequência de 5 peças, logo, a quantidade de peças em sequência deve ser um fator importante para a obtenção da lógica dessa I.A. A forma encontrada de contabilizar a heurística é então multiplicar por *ganhos* a cada número de peças em sequência. Foi decidido que o número de jogadas não tem muita influência na heurística/utilidade. Conforme é apresentado na equação abaixo:

$$h = \underbrace{PS_2\lambda_1 Npc_1 + PS_3\lambda_2 Npc_2 + PS_4\lambda_3 Npc_3}_{Computador} - (\underbrace{PS_2\Omega_1 Nh_1 + PS_3\Omega_2 Nh_2 + PS_4\Omega_3 Nh_3}_{Humano}) \quad (1)$$

Sendo as variáveis de entrada definidas como sendo as seguintes:

- PS_2 é 2 quando duas peças estão em sequência;
- PS_3 é 3 quando três peças estão em sequência;;
- PS_4 é 4 quando quatro peças estão em sequência;;
- Npc_1 é o número de duplas abertas que o computador possui;

- Npc_2 é o número de triplas abertas que o computador possui;
- Npc_3 é o número de quadras abertas que o computador possui;
- Nh_1 é o número de duplas abertas que o humano possui;
- Nh_2 é o número de triplas abertas que o humano possui;
- Nh_3 é o número de quadras abertas que o humano possui;

Os ganhos nas quais as entradas serão multiplicadas são:

- $\lambda_1 = 10^1$
- $\lambda_2 = 10^3$
- $\lambda_3 = 10^5$
- $\Omega_1 = 10^1$
- $\Omega_2 = 10^3$
- $\Omega_3 = 10^5$

Dessa forma, caso a heurística esteja mais favorável ao computador ela será positiva, do contrário, esterá favorável ao humano e terá uma heurística negativa.

A utilidade será de forma bastante similar a função heurística, com exceção de que agregará a quíntupla, pois é final de jogo. Conforme equação abaixo:

$$u = h + \underbrace{PS_5\lambda_4 Npc_4}_{Computador} - \underbrace{PS_5\Omega_4 Nh_4}_{Humano} \tag{2}$$

Sendo as variáveis de entrada definidas como sendo as seguintes:

- h é a heurística;
- PS_5 é 5 quando cinco peças estão em sequência;;
- Npc_4 é a quíntupla que computador possui em caso de vitória;
- Nh_4 é a quíntupla que humano possui em caso de vitória;
- Nj é o número da jogada;

Seguindo a mesma lógica da heurística:

- $\lambda_4 = 10^7$
- $\Omega_4 = 10^7$

Lembrando que se o nodo for folha(final de jogo), aplica-se a utilidade, se o nodo não for folha, aplica-se heurística. Sendo assim, quando há uma sequência de *cinco* peças o Npc_4 será *um* em caso se vitória do computador ou Nh_4 será *um* em caso se vitória do humano, ou ainda os dois será zero, em caso de empate, mas nunca os dois serão *um*. As constantes λ e Ω foram pré-definidas nos valores vistos acima, com a ideia de garantir que *uma* sequência contendo n peças tenha mais valor que a soma das sequências com tamanho $n-1$ e inferiores. Assim, supondo que um jogador no máximo terá 100 duplas abertas (valor alto considerando que o tabuleiro tem 225 casas), a pontuação das duplas será $2.10^1.100$, já a pontuação de *uma* tripla é $3.10^3.1$, portanto maior. Isso vale para todas as sequências de peças, garantido que um computador dê preferência em fazer *uma* tripla e e não *duas* duplas, por exemplo.

Otimizações planejadas

Para implementar o algoritmo α e β, foi pensado em alguns métodos de otimização para conseguir podar o maior número de nodos possíveis, são estes:

- Priorizar a produção do filhos cujas peças a serem posicionadas no centro do tabuleiro;

Estas ideias de priorização são importantes, pois quanto colocar peças ao centro do tabuleiro será possível fazer mais jogadas visto que não há a preocupação com as bordas do tabuleiro, assim aumentando o valor da função de heurística. Sendo um tabuleiro de grande dimensão (15x15), estratégias de otimização são interessantes pois permitem o melhor aproveitamento de recursos computacionais. Com essas ações de ordenação de estados apresentado acima, pode-se ter melhorias significativas.

Por motivos de tempo não foram implementadas tais otimizações. Mas a implementação produz apenas os filhos de acordo com a necessidade do algoritmo MiniMax.

Decisões de projeto

- game: são realizadas as verificações das sequências, são feitas a jogadas e também é responsável pelo cáculo da heurística, implementação do MiniMax e detecção de fim de jogo;
- board: realiza a implementação do tabuleiro para o jogo e modifica a aparência do tabuleiro de acordo com as jogadas realizadas.

Detecção de fim de jogo e das sequências

O fim de jogo deveria ser detectado pelo método *is_end_play*, que verifica se algum jogador chegou há uma sequência de 5 peças ou em caso de empate o tabuleiro está completo. Mas não está implementado. Já a verificação das sequências é realizada pelo método $verify\_game$, no qual possui a chamada de outros métodos que verificam as sequências nas linhas, colunas e nas duas diagonais.

Principais métodos

- heuristic_utility: recebe como parâmetro o número de sequências do tabuleiro e retorna o valor da heurística;
- minimax: implementa o algoritmo MiniMax com podas.

Limitações

O software não consegue distinguir quando é uma sequência que está com os dois lados fechados, e quando está com um ou os dois lados abertos. Como nas figuras abaixo, a pontuação é a mesma para a jogada da Figura 1 e da Figura 2. Isso é errado pois uma sequência fechada não pode levar a vitória, já a sequência da Figura 2 mostra que o jogador de peças cinzas está próximo da vitória. Em decorrência disso quando der empate o valor da heurística pode não ser 0, enganando o algoritmo MiniMax.

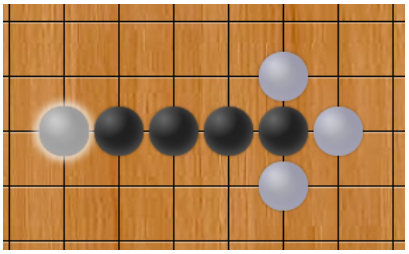

Figura 1: Distribuição de peças.

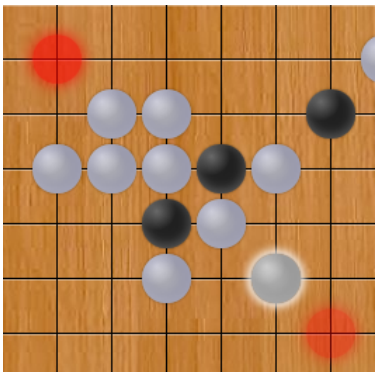

Figura 2: Distribuição de peças.

Problemas

A implementação apresenta alguns problemas, o mais grave é que o algoritmo minimax entra em loop. O outro é que quando há uma sequência de por exemplo 4 peças, a verificação das jogadas, marca como se tivessem 3 sequências de 2 peças, 2 de três peças, e 1 de quatro peças, erroneamente. Mas este problema é fácial de resolver, basta fazer um estudo de cada caso e diminuir o número das sequências com menos peças qua a contabilizada.