Inteligência Artificial
Othello

Daniel Cardoso Assumpção[1] **DRE:113279059**
Lucas Rodrigues Teixeira Nunes[1] **DRE:113202670**

[1]Departamento de Ciência da Computação
Instituto de Matemática
Universidade Federal do Rio de Janeiro

1. Objetivo

Este trabalho tem como objetivo implementar dois agentes distintos capazes de jogar Othello, utilizando como base o código do tabuleiro fornecido na descrição do trabalho.

2. Introdução

Os agentes, assim como o tabuleiro, foram feitos em Python 2.7. Optamos por implementar um agente mais simples, que utiliza busca gulosa (buscando sempre a jogada que maximize a própria pontuação dentre todas as jogadas disponíveis), e outro que utiliza o algoritmo Minimax com Corte Alfa-Beta. O Corte Alfa-Beta é essencial no caso de um agente que jogue Othello, uma vez que a árvore de jogo pode ter um fator de ramificação muito alto e a busca em profundidade pela árvore pode levar muito tempo, piorando bastante o desempenho do agente.

2.1. Regras e Objetivos

Othello trata-se de um jogo que utiliza um tabuleiro 8x8 e peças pretas e brancas. Quem ao final do jogo tiver mais peças no tabuleiro, ganha. O jogo acaba quando nenhum dos jogadores têm jogadas disponíveis, o que geralmente ocorre quando as 64 peças já estão no tabuleiro. As jogadas só podem ser feitas em direção a uma posição livre do tabuleiro e se tomarem pelo menos uma peça do oponente.

3. Agente do Torneio

O agente escolhido para disputar o torneio foi o agente **Minimax**, e ambos os agentes serão detalhados mais à frente.

4. Greedy Agent

4.1. Implementação

Este agente não utiliza nenhuma heurística muito avançada ou elaborada. Ele apenas analisa todas as jogadas possíveis, simula a jogada num clone do tabuleiro e avalia a pontuação resultante após a jogada. Após avaliar todas as jogadas possíveis, a jogada que produza a maior pontuação é a escolhida, e retornada para ser utilizada no método play.

4.2. Exemplos

5. Minimax Agent

5.1. Implementação

Este agente se utiliza do algoritmo Minimax com Corte Alfa-Beta de profundidade 4 (determinado no __init__ da classe e usado nas chamadas recursivas de minimax_corte), para evitar expansões desnecessárias na árvore de jogo (implementado na função minimax_corte). A função de avaliação escolhida foi medir a estabilidade de uma jogada. Ou seja, cada casa do tabuleiro possui um peso escolhido baseado na chance daquela casa ser estável (difícil de ser tomada pelo adversário). Com isso, o agente tenta fazer a jogada mais segura possível. Os pesos de cada casa estão numa matriz do tamanho do tabuleiro, onde o valor da casa é consultado ao calcular a avaliação das jogadas. A matriz de pesos utilizada é a seguinte:

$$\begin{bmatrix} 120 & -20 & 20 & 5 & 5 & 20 & -20 & 120 \\ -20 & -40 & -5 & -5 & -5 & -5 & -40 & -20 \\ 20 & -5 & 15 & 3 & 3 & 15 & -5 & 20 \\ 5 & -5 & 3 & 3 & 3 & 3 & -5 & 5 \\ 5 & -5 & 3 & 3 & 3 & 3 & -5 & 5 \\ 20 & -5 & 15 & 3 & 3 & 15 & -5 & 20 \\ -20 & -40 & -5 & -5 & -5 & -5 & -40 & -20 \\ 120 & -20 & 20 & 5 & 5 & 20 & -20 & 120 \end{bmatrix}$$

Para exemplificar o motivo de tais valores para cada casa, pegaremos como exemplo as casas com maior valor e menor valor. As casas com maior valor, as casas nos cantos com valor 120, são as casas onde as peças não podem ser retiradas, logo, é uma garantia de pontuação. Já as casas com valor -20, são casas estrategicamente ruins, pois, além de ser facilmente perdidas, tendem a abrir uma leque de possibilidades favoráveis ao oponente. Além disso, é utilizado o conceito de mobilidade relativa de uma jogada e a paridade de peças de uma jogada.

Mobilidade relativa se trata de uma tentativa de reduzir a mobilidade do oponente enquanto aumenta sua própria, ou seja, há uma tendência maior em escolher jogadas que levem o oponente a ter menos jogadas disponíveis e que levem o agente a ter mais jogadas disponíveis.

Já a paridade, é um cálculo simples que visa analisar a proporção de peças que o agente pode ganhar ou perder com uma jogada, fazendo com que ele priorize jogadas em que ele possua uma maior vantagem em relação ao seu oponente.

Para priorizar estratégias mais importantes, foram colocados pesos em cada uma delas. Os pesos são:

- Estabilidade: 60
- Mobilidade: 35
- Paridade: 5

6. Considerações finais

O trabalho foi muito importante na compreensão dos algoritmos utilizados, pois com aplicações práticas dos algoritmos em situações do mundo real, as dúvidas são elucidadas mais facilmente. O agente escolhido para disputar o torneio será o agente Minimax, justamente por jogar de forma mais consciente e eficiente.

7. Referências

http://radagast.se/othello/Help/strategy.html

http://www.samsoft.org.uk/reversi/strategy.htm