ELEMENTOS DE INTELIGÊNCIA ARTIFICIAL E CIÊNCIA DE DADOS

1º TRABALHO PRÁTICO

MARIANA SERRÃO - UP202109927
MAFALDA AIRES - UP202106550

MÉTODOS DE PESQUISA PARA JOGOS COM ADVERSÁRIOS

O jogo escolhido foi o **Mancala**, um jogo de raciocínio lógico, para dois jogadores, em que o objetivo é recolher o maior número de sementes possível.

O jogo vai ser representado numa classe em Python, com o nome de **Board**, que vai conter duas listas, uma com as sementes disponíveis em cada casa do jogo e uma com o total de sementes que cada jogador recolheu até ao momento.

Formulação do Problema

1 REPRESENTAÇÃO DO ESTADO

P1[qt, c]; P2[qt,c]; C1[qt, c]; C2[qt, c]; ...; C12[qt, c].

2 ESTADO INICIAL

P1[0, 48]; P2[0,48]; C1[4, 48]; C2[4, 48]; ...; C12[4, 48].

3 OPERADORES

Jogar_C1: C1[qt-1, 48]; C2[qt+1, 48]; ...

...

Jogar_C12: C12[qt-1, 48]; P1[qt+1, 48]; ...

4 OBJETIVO

P1(qt) > P2(qt) --> P1 ganha o jogo.

5 FUNÇÃO DE UTILIDADE

Vitória = +1; Derrota = -1; Empate = 0;

ABORDAGEM

Foi criada uma função de avaliação (função utilidade), usada para avaliar uma determinada posição do jogo e determinar a sua "utilidade" para um jogador específico. Esta retorna um valor numérico que indica o quão vantajosa é a posição atual para o jogador em causa.
Na nossa implementação, a função avalia a diferença entre a pontuação do jogador atual e a pontuação do jogador oponente (armazenadas nas variáveis mancala_atual e mancala_oponente, respectivamente). Se o jogo já tiver terminado, a função retorna um valor de 100 se o jogador atual ganhou, -100 se o jogador oponente ganhou, ou 0 se houve um empate. Em todos os casos, a diferença de pontuação é adicionada ao valor de retorno para indicar o quão vantajosa é a posição atual para o jogador.

```python
#função utilidade
def utilidade(self, jog):
    mancala_atual = self.mancalas[jog-1]
    if jog==1:
        outro_jog = 2
        mancala_oponente = self.mancalas[1]
    elif jog==2:
        outro_jog = 1
        mancala_oponente = self.mancalas[0]
    if self.fim_jogo() != -1:
        pontuacao = mancala_atual - mancala_oponente
        if self.fim_jogo() == jog:
            return 100 + pontuacao
        elif self.fim_jogo() == outro_jog:
            return -100 - pontuacao
        else:
            return 0
    else:
        return mancala_atual - mancala_oponente
```

ALGORITMOS DE PESQUISA IMPLEMENTADOS

MINIMAX COM CORTES ALFA-BETA

O algoritmo minimax baseia-se numa árvore de pesquisa, onde cada nó representa um estado do jogo. A ideia é percorrer a árvore, avaliando qual o melhor nó a escolher em cada expansão. Existem dois jogadores: o minimizador, cujo objetivo é ter a utilidade menor possível e o maximizador, que pretende ter a maior utilidade possível. Assim, se for a vez do minimizador, o nó escolhido será aquele com um menor aumento da sua utilidade e, com o maximizador, o nó será o que implica uma menor perda da utilidade. Com os cortes Alfa-Beta, Alfa é o valor da melhor jogada do maximizador, e Beta o melhor do minimizador, tendo em conta a função utilidade. Durante a procura, se o valor Beta for menor ou igual a Alfa, então não se explora os outros ramos do nó.

ÁRVORE DE PESQUISA MONTE CARLO

O MCTS trabalha criando uma árvore de pesquisa que representa todas as possíveis jogadas a partir da posição atual do jogo. Em seguida, simula jogos aleatórios até o final a partir de cada nó da árvore, para estimar a probabilidade de vitória a partir daquele nó. Durante a simulação, o algoritmo segue uma política de seleção (função selection) que faz a exploração (função expand) dos nós, escolhendo aleatoriamente um dos nós que registaram os maiores valores na aplicação da função Upper Confidence Bound. De seguida, realiza a simulação do jogo com esse nó e a retropropagação. Repete este processo até que um limite de tempo seja atingido. Depois disso, a melhor jogada é escolhida com base na contagem de visitas dos nós da árvore.

RESULTADOS

MODO DIFÍCIL - COMPUTADOR VS COMPUTADOR

JOGADOR 1: MINIMAX COM CORTES ALFA-BETA
JOGADOR 2: MINIMAX COM CORTES ALFA-BETA

Número de Jogadas	**40**
Tempo de Execução	**0.05 s**

GANHA O JOGADOR 1

JOGADOR 1: MONTE CARLO
JOGADOR 2: MONTE CARLO

Número de Jogadas	**74**
Tempo de Execução	**186 s**

GANHA O JOGADOR 1

RESULTADOS

MODO DIFÍCIL – COMPUTADOR VS COMPUTADOR

GADOR 1: MINIMAX COM CORTES ALFA-BETA
JOGADOR 2: MONTE CARLO

Número de Jogadas	**32**
Tempo de Execução	**57.03 s**

GANHA O JOGADOR 1

JOGADOR 1: MONTE CARLO
JOGADOR 2: MINIMAX COM CORTES ALFA-BETA

Número de Jogadas	**59**
Tempo de Execução	**63.07 s**

GANHA O JOGADOR 2

CONCLUSÕES

Em jogos de informação completa, como é o caso do Mancala, é possível utilizar o minimax com cortes alfa-beta para encontrar a melhor jogada. Isto ocorre uma vez que todas as informações relevantes para a tomada de decisão estão disponíveis aos jogadores.

Por outro lado, embora o MCTS também possa ser usado em jogos de informação completa, a sua eficiência não é tão elevada, uma vez que depende de simulações aleatórias e necessita de realizar um grande número de simulações para obter bons resultados, o que dá origem a um tempo de execução mais longo, como observado nos resultados.

REFERÊNCIAS

- https://github.com/cypreess/py-mancala
- https://github.com/Priyansh-15/Mini-Max-Algorithm-Based-Mancala-Board-Game
- https://towardsdatascience.com/simulating-mancala-what-happens-when-i-push-this-game-to-its-limits-28d9c0a58616