U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Jogo com Oponentes
(Connect Four)

Ricardo Araújo Amorim, up202107843
David Rafael Pereira Nogueira, up202108293
Pedro Morim Figueiredo Andrade Leitão, up202107852

Porto
2023

U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Índice

Introdução

Jogos adversariais são jogos com dois jogadores que se enfrentam com o objetivo de ganhar/derrotar o seu oponente. Exemplos de jogos adversariais incluem xadrez, damas, Go, entre outros. A solução deste tipo de jogos é um campo na inteligência artificial muito explorado e inclui uma alta quantidade de algoritmos diferentes. Como tal, neste relatório, vamos analisar e comparar três desses algoritmos: Minimax, Poda alfa-beta e Monte Carlo Tree Search (MCTS) aplicados ao jogo Connect Four.

Min-Max

O algoritmo Minimax é um dos algoritmos mais conhecidos para solucionar jogos adversariais. Nele, os dois jogadores são chamados Max e Min. Max tenta obter a maior pontuação possível, enquanto o Min tenta obter a pontuação mais baixa possível.

O objetivo do algoritmo é encontrar a melhor jogada para o jogador em questão, tendo em conta as possíveis jogadas do adversário. O algoritmo funciona através da construção de uma árvore de pesquisa, em que cada nó representa um estado possível do jogo e cada ramificação representa uma jogada. O algoritmo avalia as folhas da árvore (ou seja, os estados em que o jogo termina) através de uma função de avaliação, que atribui uma pontuação a cada estado. Desta forma, o algoritmo vai explorando a árvore de forma recursiva, alternando entre os jogadores, até chegar à raiz da árvore, que representa o estado atual do jogo.

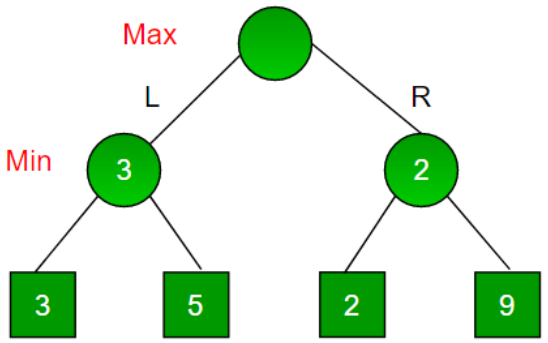


Fig.1

Alpha-Beta

O algoritmo Alpha-Beta Pruning é uma variação do algoritmo Minimax que reduz a quantidade de nós a serem avaliados. O algoritmo usa dois valores, alpha e beta, para manter o intervalo de valores possíveis de um nó. Alpha representa o maior valor encontrado até ao momento e beta representa o menor valor encontrado até ao momento. Quando um nó é avaliado, o algoritmo compara o valor do nó com os valores alpha e beta e atualiza os valores, se necessário. Quando estiver num nó num nível de máximo, se alpha ≥ que o beta do nó pai, então o algoritmo pode parar de avaliar o restante dos nós filhos desse nó, porque o jogador atual não vai escolher essa jogada. Da mesma forma, quando estiver num nó num nível de mínimo, se beta ≤ que o alpha do nó pai, então o algoritmo pode parar de avaliar o restante dos nós filhos desse nó, porque, mais uma vez, o jogador atual não vai escolher essa jogada.

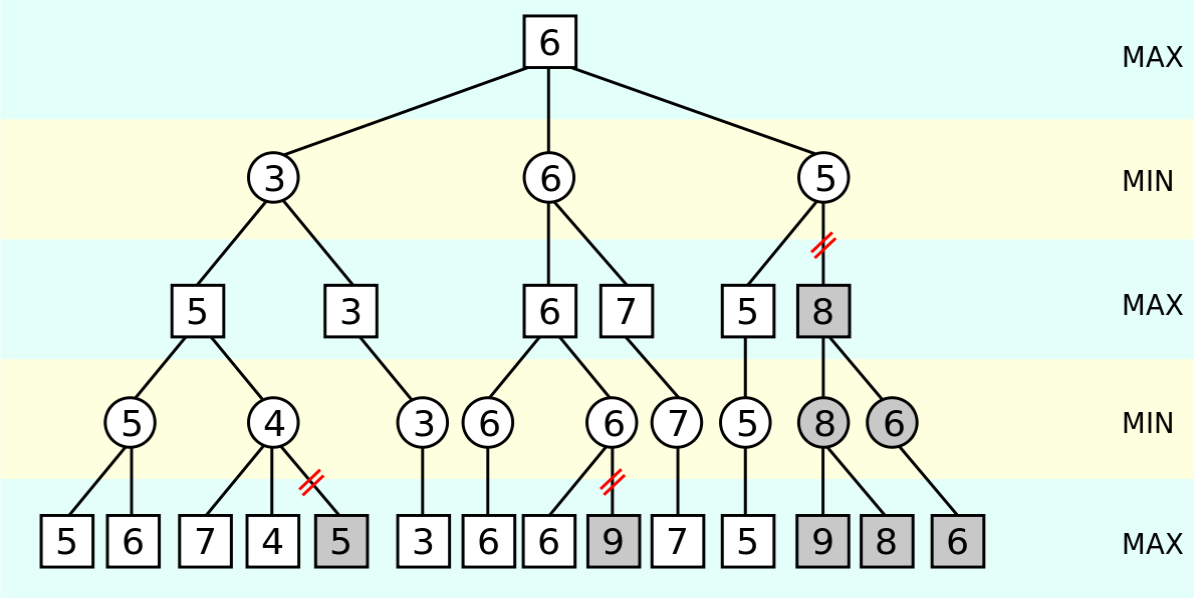


Fig.2

Monte Carlo Tree Search

O Monte Carlo Tree Search (MCTS) é um algoritmo de busca em árvore que é frequentemente utilizado em jogos cujo espaço de estados seja muito grande. Por exemplo: Battleship Poker e Go

O algoritmo é constituído por quatro passos diferentes:

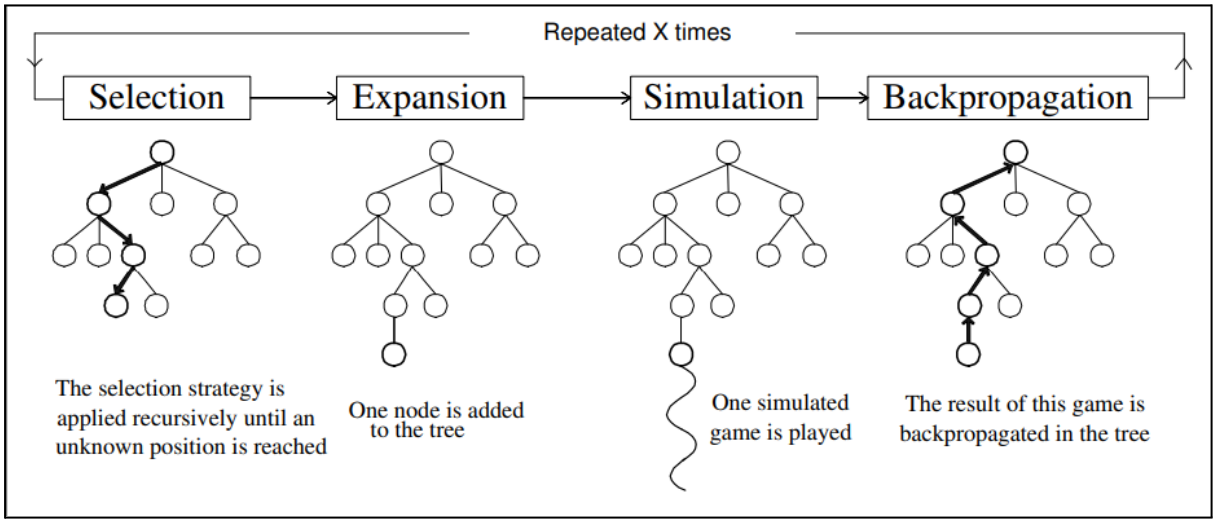


Fig.3

É baseado em uma exploração aleatória do espaço de busca. Usando os resultados de explorações anteriores, o algoritmo gradualmente constrói uma árvore de jogo na memória e torna-se sucessivamente melhor na estimativa precisa dos valores dos movimentos mais promissores. MCTS é aplicável se pelo menos as seguintes três condições forem satisfeitas: (1) as pontuações do jogo são limitadas, (2) as regras são conhecidas (informação completa) e (3) as simulações terminam relativamente rápido (o comprimento do jogo é limitado).

Connect Four

Connect Four, também conhecido como Quatro em Linha, é um jogo de estratégia para dois jogadores criado por Howard Wexler e Ned Strongin em 1974. O objetivo do jogo é "conectar", ou seja, colocar quatro peças da mesma cor, numa grade de seis linhas por sete colunas, seguidas uma atrás da outra tentando sempre impedir que o nosso oponente consiga fazer o mesmo antes de nós, tanto vertical, horizontal ou diagonalmente.

Regras:

Cada jogador escolhe uma cor, normalmente vermelho ou amarelo, de seguida, vão se alterando, colocando uma peça da sua cor em uma das colunas da grade. O objetivo é conectar quatro peças da mesma cor em uma linha, seja vertical, horizontal ou diagonal. O jogo termina quando um jogador consegue conectar quatro peças ou quando a grade fica completamente preenchida sem que nenhum jogador tenha conectado quatro peças.

Funções de avaliação:

Uma função de avaliação no Connect Four é um valor que mede o quão favorável é a posição atual do jogo para um determinado jogador. É usada, por isso, pelos algoritmos que exploram o jogo para decidir qual jogada fazer a seguir. No nosso caso, a nossa função de avaliação é disponibilizada no enunciado do trabalho e é do seguinte formato:

```
-50 for three Os, no Xs,
-10 for two Os, no Xs,
- 1 for one O, no Xs,
  0 for no tokens, or mixed Xs and Os,
  1 for one X, no Os,
 10 for two Xs, no Os,
 50 for three Xs, no Os.
```

Fig.4

Implementação dos algoritmos no Connect Four

Descrição da Implementação

Board

Para recriar-mos as regras do jogo, decidimos criar uma classe **Board** que guarda o estado do tabuleiro numa matriz, o jogador que está a fazer a jogada numa *flag* (0 para a pessoa, 1 para o computador) e a quantidade de peças numa coluna num array *max*.
Para além disso, criamos mais cinco funções:

- **add()** - adiciona a respectiva peça (‘O’ ou ‘X’) à coluna desejada
- **complete()** - retorna *True* se a coluna selecionada estiver cheia e não der para colocar mais peças, caso contrário retorna False
- **points()** e **contar()** - calcula o valor de pontos do estado atual do tabuleiro para ser utilizado pelos algoritmos
- **printBoard()** - é o que imprime o tabuleiro no terminal

MiniMax

O nosso MiniMax foi implementado da seguinte forma:
Criamos uma função **full()** que verifica se o tabuleiro está cheio, uma outra função **winner()** que sempre que é colocada uma peça, verifica as posições em todas as direções para ver se alguém ganhou, estando estas duas funções nos 3 algoritmos feitos, e por fim a própria função recursiva do MiniMax chamada **recursive()** que aplica o algoritmo em si, explicado anteriormente, e retorna qual a melhor peça para o computador jogar.

Alpha-Beta Pruning

Para implementar os cortes Alpha-Beta bastou-nos apenas adicionar dois inteiros ao já criado algoritmo MiniMax:
Alpha - Melhor valor encontrado para o jogador que maximiza.
Beta - Melhor valor encontrado para o jogador que minimiza.
E fazer as condições dos cortes:

- Num nível de máximo: Se alpha ≥ que o beta do nó pai dá break
- Num nível de mínimo: Se beta ≤ que o alpha do pai dá break

Monte Carlo Tree Search

Para implementar o Monte Carlo Tree Search criamos em primeiro lugar uma função com o algoritmo base dele (mcts), em que retorna um número inteiro representando a coluna onde o próximo movimento deve ser feito.

Essa função mcts vai chamar algumas funções auxiliares:

A função para selecionar o nó a expandir, em que vai usar o UCT (Upper Confidence Bound applied to trees).

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$ Fig.5

- w_i = número de vitórias após a jogada i.
- n_i = número de simulações após a jogada i.
- c = parâmetro de exploração (equivalente a $\sqrt{2}$)
- t = total número de simulações para o nó pai

A função que faz a expansão, em que, pega num Nó e expande-o criando todos os possíveis nós filhos. Retorna o primeiro nó criança.

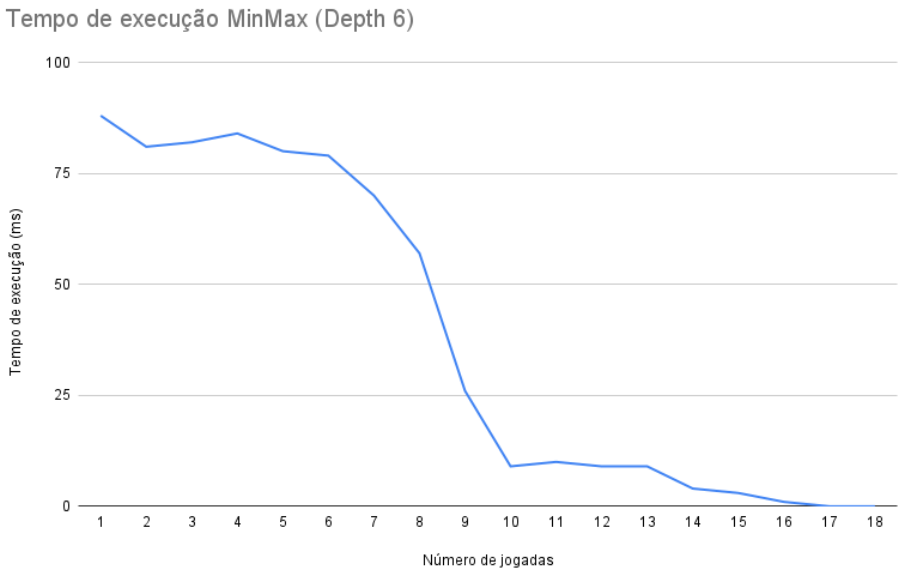
Uma função que faz a simulação, onde seleciona aleatoriamente uma coluna válida para o próximo movimento.Se esse movimento não originar uma vitória, o processo repete-se até ser encontrado uma.

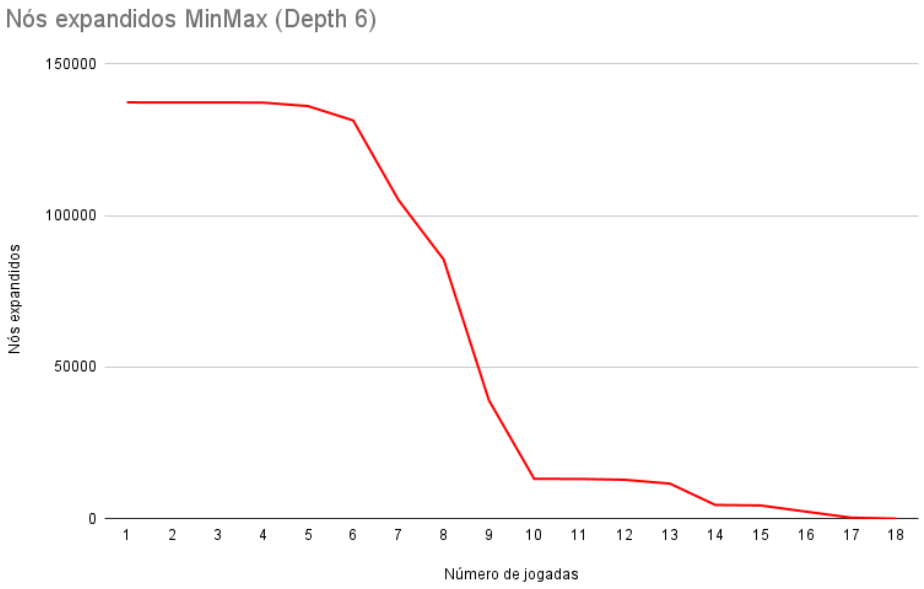
Implementações:

Pusemos um depth-limit de 6 para os testes do algoritmo, sendo possível alterar na chamada da função recursiva para a jogada do computador.

Input: 4 3 5 3 3 5 4 5 3 5 2 1 6 7 7 1 1 6

Min-Max

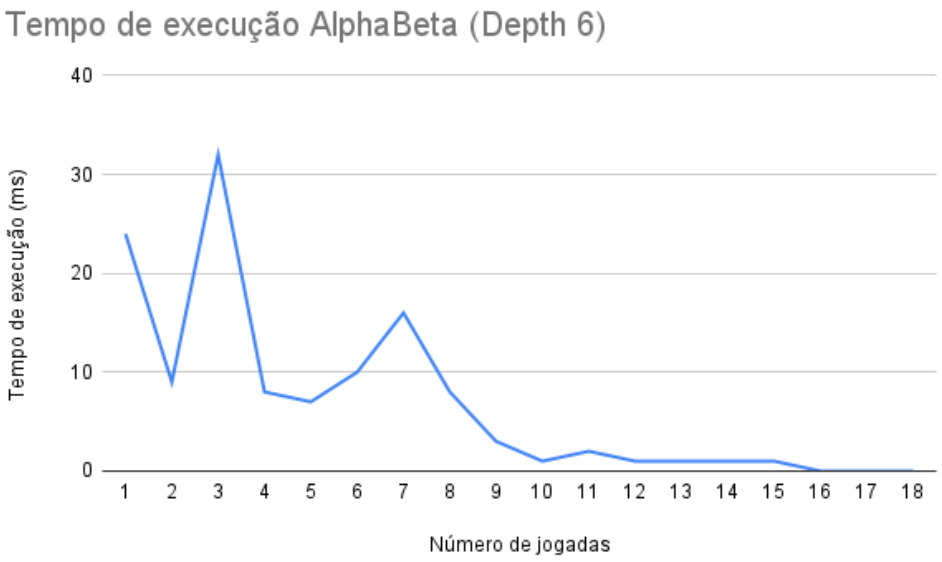


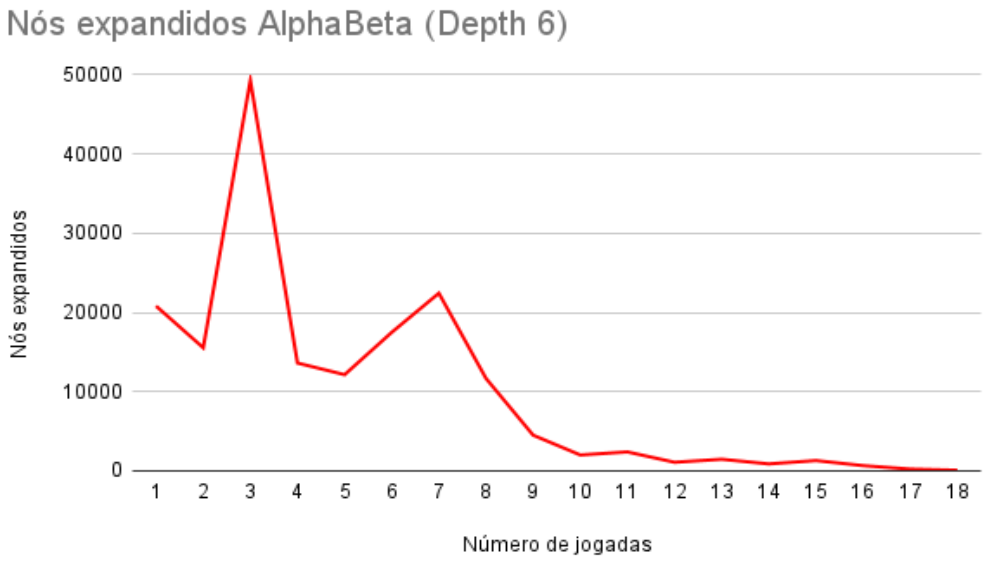


Alpha-Beta

Pusemos um depth-limit de 6 para os testes do algoritmo, sendo possível alterar na chamada da função recursiva para a jogada do computador.

Input: 4 3 5 3 3 5 4 5 3 5 2 1 6 7 7 1 1 6

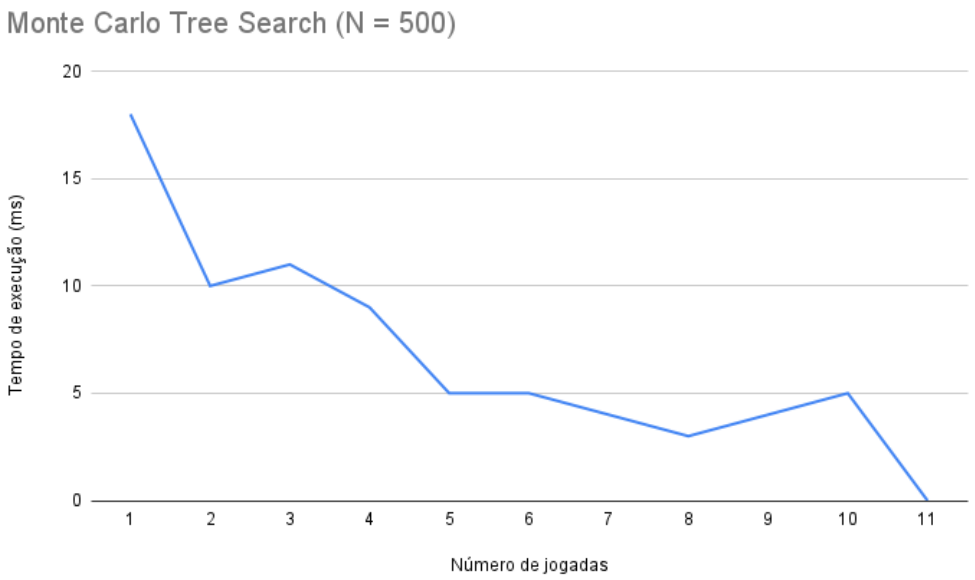


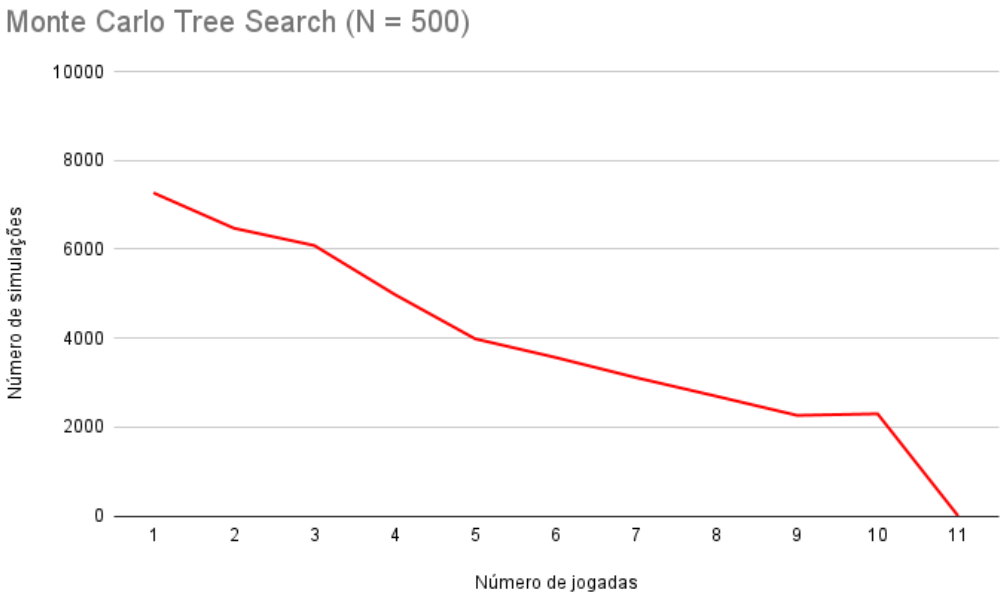


Monte Carlo Tree Search

Pusemos um limite de 500 repetições para os testes do algoritmo, sendo possível alterar no início da função MCTS.

Input: 4, 5, 4, 5, 6, 3, 6, 2, 5, 6, 4





Comentários Finais e Conclusão

Com este trabalho conseguimos implementar o MinMax, AlphaBeta e Monte Carlo Tree Search para o jogo do Connect Four. São algoritmos que têm as suas vantagens e desvantagens em comparação uns com os outros.

Usando o MinMax, o computador vai escolher, em geral, a jogada que mais o beneficia, assumindo que o oponente jogará também a melhor jogada possível. Com os testes, concluímos que é um bom algoritmo para jogos pequenos, sendo que poderá ser computacionalmente caro em jogos cujo espaço de estados seja muito elevado.

Já com o AlphaBeta, uma melhoria do MinMax, vai reduzir o número de estados expandidos, excluindo ramos que não sejam relevantes. Em comparação com o MinMax, essa mudança vai fazer com que o AlphaBeta seja melhor em termos de eficiência, tanto em tempo de execução como em nós explorados.

Por fim, utilizando o Monte Carlo Tree Search (MCTS), o computador vai fazer uma busca com base em simulações aleatórias para encontrar o melhor movimento. Nos testes que fizemos, geralmente era o algoritmo mais eficiente, no entanto em alguns casos não foi tão eficaz em alcançar a vitória como os outros dois algoritmos.

Referências bibliográficas

- https://project.dke.maastrichtuniversity.nl/games/files/phd/Chaslot_thesis.pdf
- Slides das aulas teóricas da unidade curricular Inteligência Artifical (2022/2023) (21/04/2023).
- https://pt.wikipedia.org/wiki/Poda_(computa%C3%A7%C3%A3o)
- https://www.mathsisfun.com/games/connect4.html
- Russel,S. & Norvig,P. (2021). Artificial Intelligence: A Modern Approach, Global Edition(4th ed.).Pearson. (18/04/2023)
- Fig.1-https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/
- Fig.2- https://pt.wikipedia.org/wiki/Poda_(computa%C3%A7%C3%A3o)
- Fig.3-https://project.dke.maastrichtuniversity.nl/games/files/phd/Chaslot_thesis.pdf
- Fig.4 - pdf sobre implementação do trabalho fornecido pelos docentes da unidade curricular Inteligência Artifical (2022/2023)
- Fig. 5 - https://www.baeldung.com/java-monte-carlo-tree-search