Relatório de Inteligência Artificial

2022/23

André Serra - 201804979

Bernardo Salgado – 202004493

Henrique Dias - 202108140

Miguel Ferreira – 202104553

Índice

Introdução 3
Minimax 3
Alpha-Beta Pruning 4
Monte Carlo Tree Search (MCTS) 7
Quatro em linha 8
Funções de avaliação 8
Implementação 9
Estatísticas 10
Conclusão e comentários finais 11
Referências 11

Introdução

Os jogos mais populares em inteligência artificial são jogos de dois jogadores, determinísticos, de turno, jogos de soma zero com informações perfeitas (como jogo do galo, quatro em linha, xadrez e etc.). São ambientes com dois jogadores que jogam alternadamente e onde os valores de utilidade de fim de jogo são sempre iguais e opostos. Ou seja, se um deles ganha o seu oponente perde obrigatoriamente.

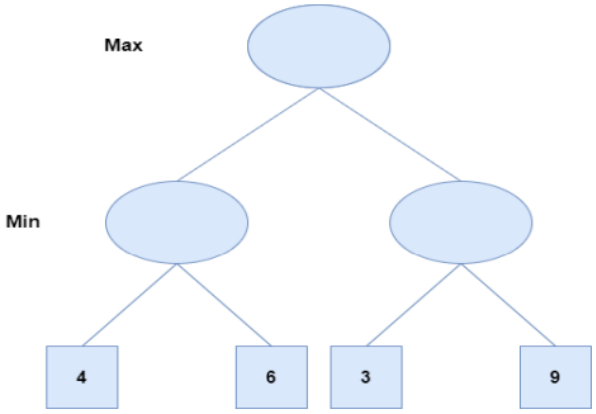
Minimax

O princípio fundamental do minimax é levar em conta todos os movimentos possíveis que um jogador pode fazer e determinar para cada movimento o maior ganho possível que o oponente (neste caso o computador) pode ter.

Neste algoritmo o jogador que é ajudado, é o MAX, pois o algoritmo tenta maximizar as suas possibilidades de vitória, e o seu adversário (o computador) é o MIN, visto que o objetivo do algoritmo é diminuir (minimizar) a sua probabilidade de ganhar.

No minimax cada tabuleiro tem um valor agregado, se o MAX estiver em vantagem o valor deste tabuleiro é positivo, se o contrário acontecer então o valor vai ser negativo, estes valores são calculados por heurísticas únicas para cada jogo.

Por exemplo:

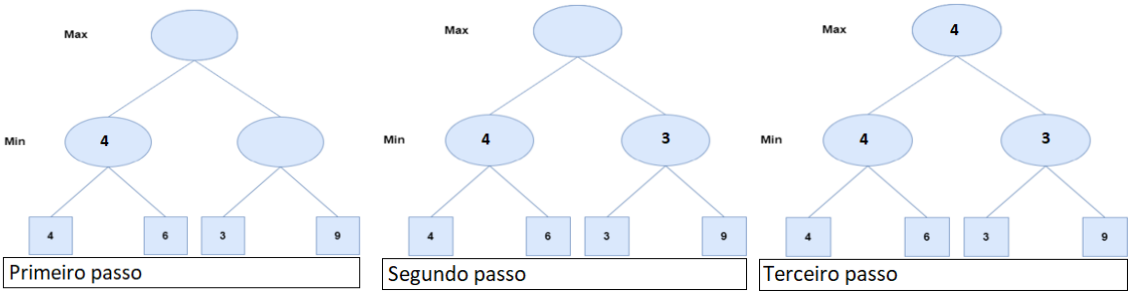


O minimax usa backtracking, ou seja, tenta todos os movimentos e depois retrocede para tomar a melhor decisão.

Primeiro passo: O MAX vai para a esquerda, depois o MIN vai ter que decidir entre 4 e 6 e uma vez que 4 < 6 ele escolhe o 4.

Segundo passo: O MAX vai agora para a direita, logo o MIN vai escolher entre 3 e 9 e como 3 < 9 o minimizador vai escolher o 3.

Terceiro passo: Visto que os MINs já escolheram os seus valores, agora é a vez do MAX e este vai optar pelo 4 porque 4 > 3.



Alpha-Beta Pruning

O Alpha-Beta Pruning é considerado uma otimização do algoritmo minimax.

Alguns dos ramos do jogo não vão ser escolhidos se o adversário estiver a jogar de forma ótima.

Agora temos 2 parâmetros, chamados de alpha e beta.

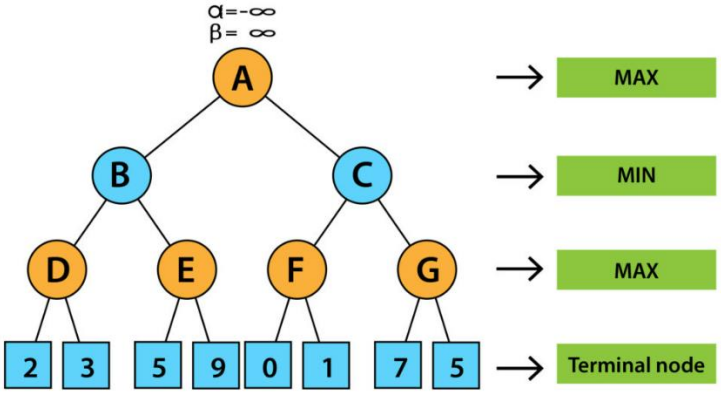
Alpha: é usado apenas pelo maximizador, e representa o maior valor encontrado até agora, o seu valor inicial é de -∞.

Beta: é usado apenas pelo minimizador, e que representa o menor valor encontrado até agora, o seu valor inicial é ∞.

A condição alpha >= beta é utilizada para descobrir que ramos são desnecessários.

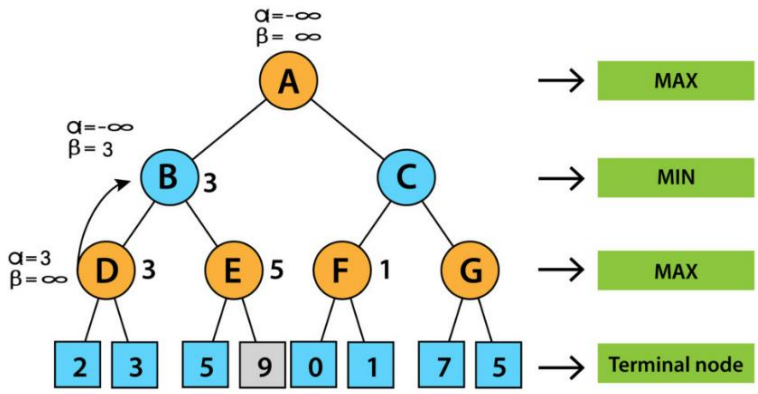
Por exemplo:

Primeiro passo: Vamos começar com o movimento inicial. Inicialmente definiremos os valores alpha e beta como o pior caso, ou seja, α = -∞ e β= +∞.



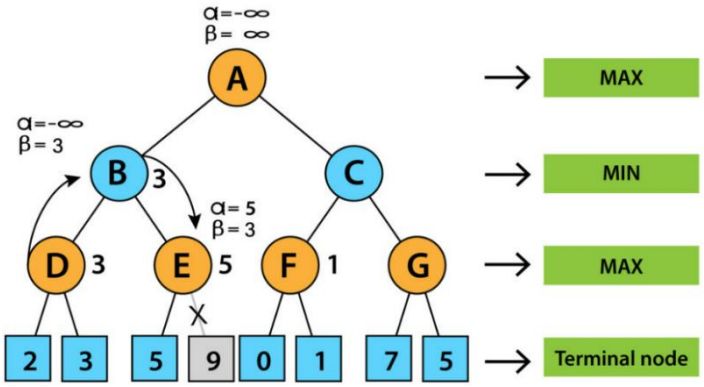
Segundo passo: Como o valor inicial de alpha é menor que beta, não o removemos (não fazemos pruning). Agora é a vez do MAX. Assim, no nó D, o valor de alpha será calculado. O valor de alpha no nó D será o máximo entre 2 e 3. Portanto, o valor de alpha no nó D será 3.

Terceiro passo: Agora o próximo movimento será no nó B (MIN). Portanto, no nó B, o valor de alpha beta será min (3, ∞). Logo, no nó B os valores serão alpha= – ∞ e beta será 3.



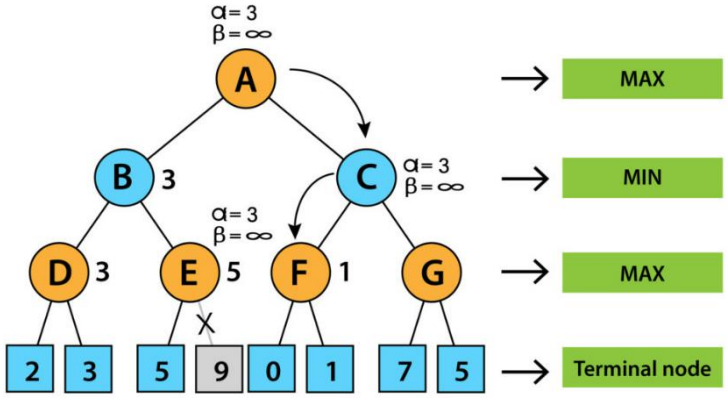
No próximo passo, os algoritmos percorrem o próximo sucessor do Nó B, que é o nó E, e os valores de α= -∞ e β= 3 também são passados.

Quarto passo: Agora é a vez do MAX. Então, no nó E vamos procurar MAX. O valor atual de alpha em E é – ∞ que será comparado com 5. Portanto, MAX (- ∞, 5) será 5. Logo, no nó E, alfa = 5, Beta = 5. Agora, como podemos ver esse alpha é maior que o beta, que está satisfazendo a condição de remoção, para que possamos fazer pruning do nó filho à direita de E, o algoritmo não será percorrido e o valor no nó E será 5.

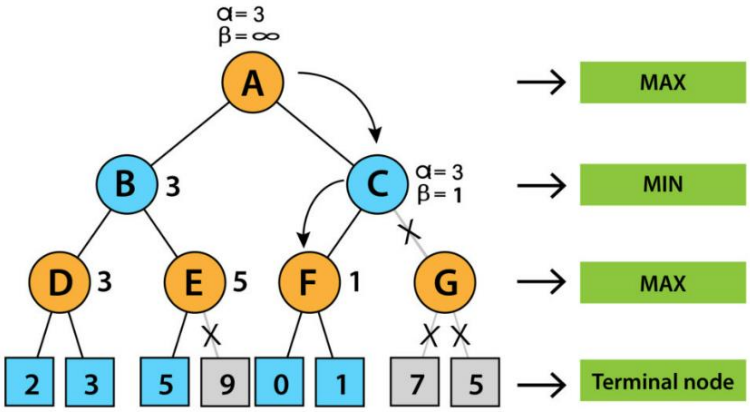


Quinto passo: Neste passo, o algoritmo chega novamente ao nó A através do nó B. No nó A, o alpha será alterado para o valor máximo de MAX (- ∞, 3). Portanto, agora o valor de alpha e beta no nó A será (3, + ∞), respetivamente, e será transferido para o nó C. Esses mesmos valores serão transferidos para o nó F.

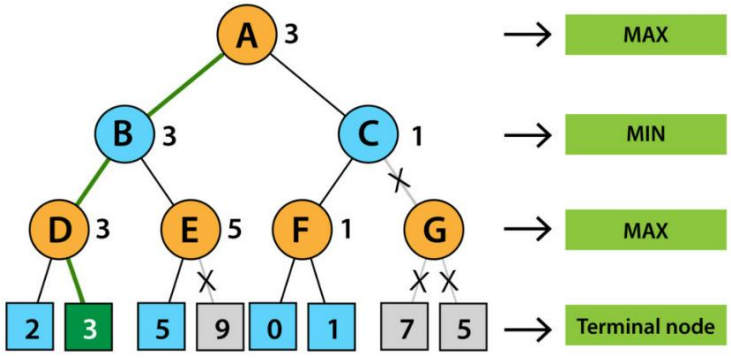
Sexto passo: No nó F, o valor de alpha será comparado ao ramo esquerdo que é 0. Então, MAX (0, 3) será 3 e comparando com o filho direito que é 1, e MAX (3,1) = 3 α ainda permanece 3, mas o valor do nó de F se tornará 1.



Sétimo passo: Agora o nó F retornará o valor do nó 1 para C e irá comparar com o valor beta em C. É a vez de MIN, MIN (+ ∞, 1) será 1. Agora no nó C, α= 3 e β= 1 e alfa é maior que beta, o que novamente satisfaz a condição de pruning. Assim, o próximo sucessor do nó C, ou seja, G também será alvo de pruning.



Agora, C retornará o valor do nó para A e o melhor valor de A será MAX (1, 3) que é 3.



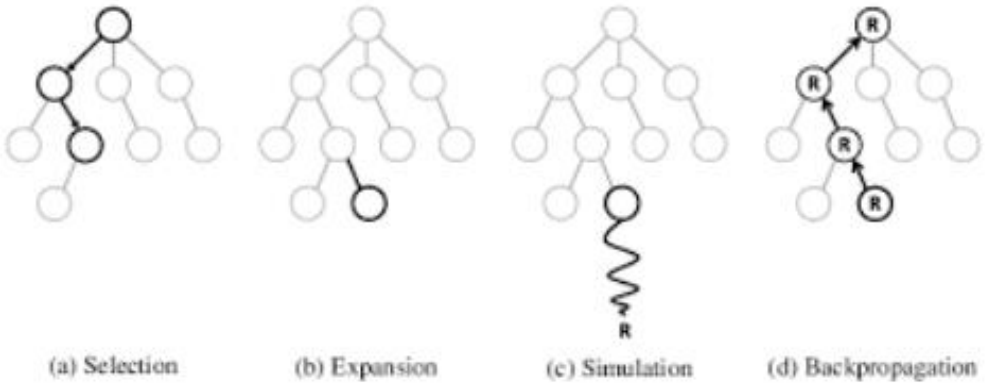
Esta é a árvore final que mostra os nós que são computados e os nós que não são computados. Assim sendo, para este exemplo, o valor ideal do maximizador será 3. (1)

Monte Carlo Tree Search (MCTS)

O Monte carlo tree search (MCTS) é um algoritmo de procura que usa procura heurística e probabilística para chegar ao objetivo de forma mais eficaz. Quando um algoritmo segue o melhor caminho, este pode não ser a solução ótima, por isso o MCTS avalia todas as outras opções. Isto é feito durante a fase de aprendizagem no lugar da solução ótima atual, processo conhecido como exploration-exploitation trade-off.

Minimax e alpha-beta podem não conseguir resolver alguns jogos cujo espaço de estados seja muito grande, como por exemplo o Monopoly e o Backgammon.

MCTS ajuda a resolver estes tipos de jogos através destes 4 passos.



Seleção: Neste processo, o algoritmo MCTS percorre a árvore atual do nó raiz usando uma estratégia específica. A estratégia usa uma função de avaliação para selecionar nós de forma otimizada com o maior valor estimado. O MCTS usa a fórmula Upper Confidence Bound (UCB) aplicada às árvores como estratégia no processo de seleção para atravessar a árvore. A UCB é o número de vitórias a dividir pelo número de vezes que o algoritmo passou por aquele nó.

Expansão: Um novo nó filho é adicionado à árvore para aquele nó que foi alcançado de forma otimizada durante o processo de seleção.

Simulação: Uma simulação é realizada escolhendo movimentos ou estratégias até que um resultado ou estado predefinido seja alcançado.

Backpropagation (retro propagação): Depois de determinar o valor do nó recém-adicionado, a restante árvore deve ser atualizada. Assim, é realizado o processo de backpropagation, onde ele a faz do novo nó para o nó raiz. Durante o processo, o número de simulações armazenadas em cada nó é incrementado. Além disso, se a simulação do novo nó resultar em uma vitória, o número de vitórias também será incrementado. (2)

Quatro em linha

O 4 em linha é um jogo de estratégia que se desenrola num tabuleiro de 6 por 7, ou seja, 6 linhas e 7 colunas.

Ambos os jogadores jogam as suas peças no tabuleiro (inicialmente vazio), uma peça por jogada alternando os jogadores.

É jogado usando 42 peças, cada um com peças coloridas diferentes (geralmente 21 fichas vermelhas para um jogador e 21 fichas amarelas para o outro).

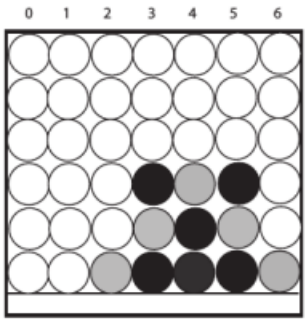
Um movimento consiste num jogador deixar cair uma das suas fichas numa coluna à sua escolha, caindo no fundo da coluna ou caindo em cima duma outra peça (colocada nessa coluna anteriormente).

O jogo termina quando uma das condições seguintes se verificar:

- Um dos jogadores coloca quatro ou mais peças numa linha contínua vertical, horizontal ou diagonalmente, ou seja, vence.

- Todas as casas do tabuleiro estão ocupadas e nenhum jogador satisfaz a condição anterior de vitória. Neste caso o jogo termina em empate.

Exemplo de um jogo de quatro em linha:



Funções de avaliação

Modifica-se o minimax ou o alpha-beta substituindo a função de utilidade por uma função de avaliação heurística e substitui-se o teste terminal por um teste de corte.

Para o este jogo é utilizada uma heurística que dá uma pontuação a todos os segmentos de 4 peças. Essa pontuação é dada por:

-Se existem 4 peças do computador: 512 pontos (computador venceu);

-Se existem 3 peças do computador: 50 pontos;

-Se existem 2 peças do computador: 10 pontos;

-Se existe só 1 peça do computador: 1 ponto;

-Se existem peças dos dois jogadores: 0 pontos;

-Se existe só peça do jogador: -1 ponto;

-Se existem 2 peças do jogador: -10 pontos;

-Se existem 3 peças do jogador: -50 pontos;

-Se existem 4 peças do jogador: -512 pontos (jogador venceu). (3)

Implementação

Usamos uma classe Game, com 3 atributos: matriz de char ('O' e 'X') game 6 por 7, char player e char winner. Esta classe tem várias funções, o playerSwitcher que serve para trocar de jogador após cada jogada, o play que serve para começar o jogo, entre outras.

Usamos um limitador de profundidade no minimax e outro no alphabeta, o max_depth no minimax é 7 e no alphabeta é 11.

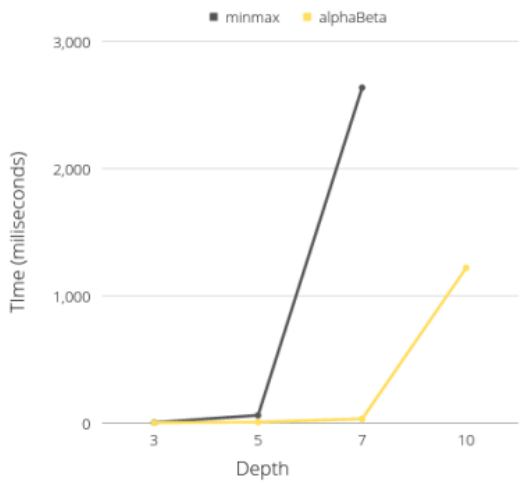
A MaxSearch chama a função max depois elas veem se é ou não possível jogar na coluna em questão, se sim criamos um nó filho com um estado a seguir ao do nó pai e chamamos a função min, se não for possível jogar naquela coluna o valor do nó passa a ser 0 para nunca ser escolhido. Depois de fazer a recursividade e, ambas as funções criamos novamente um nó que guarda os dados do melhor filho.

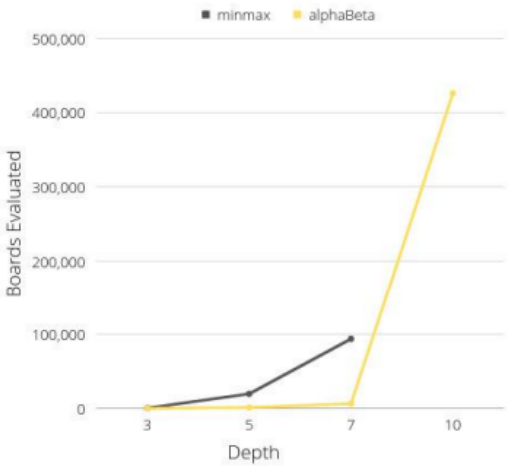
As funções max e a min não guardam nada enquanto não chegarem ao max_depth ou a uma vitória de qualquer um.

Para o MCTS usamos uma priority queue que se basiea na ucb, o primeiro node da priority queue é escolhido e expandido para todos os filhos, seguidamente faz 4000 simulações a partir de cada filho, aumentando o contador que guarda o número de vezes que passa pelo nó. Se chegar a um estado final do jogo, ele faz backpropagation, verifica qual dos filhos é que tem maior ucb e faz essa jogada.

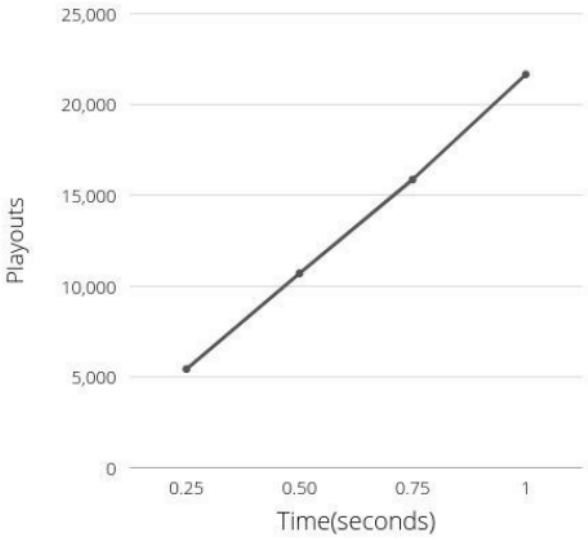
Estatísticas

Minimax e alphabeta:





MCTS:



Conclusão e comentários finais

Os algoritmos minimax e alphabeta são algoritmos que verificam todo o espaço de jogadas e por isso são considerados bons algoritmos quando um tabuleiro tem poucas jogadas possíveis, ou seja, quando o grau de ramificação da árvore é baixo. E como o grau de ramificação do quatro em linha é sete, a arvore torna-se muito grande. Isto faz com que o minimax só consiga ver até sete de jogadas em tempo normal, ou seja, passando das sete jogadas já demora muito tempo, uma vez que o tempo de execução aumenta exponencialmente. Mesmo o alphabeta, que tem uma performance bastante melhor consegue ver apenas dez ou onze jogadas antes que o tempo de execução aumente drasticamente.

O algoritmo MCTS faz muitas simulações do jogo e, ao contrário dos outros dois algoritmos, não tem problemas quando o grau de ramificação é muito grande, porque ele escolhe um nó (de cada vez) para desenvolver até ao fim através de jogadas aleatórias. Logo, podemos dizer que o número de simulações é diretamente proporcional ao tempo de execução.

Referências

(1) https://www.mygreatlearning.com/blog/alpha-beta-pruning-in-ai/
(2) https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/
(3) Moodle de inteligência artificial
(4) Artificial Intelligence: A Modern Approach, EBook, Global Edition