Problem 3: Adversarial search

Grupo: P2 G4

Trabalho realizado por:

Afonso Guerreiro: a75660

Gonçalo Figueiredo a71353

Descrição do problema:

Ora este problema consiste em implementar um de dois algoritmos de adversarial search à nossa escolha, (MCTS - Monte carlo tree search ou Minimax com cortes alpha beta), e aplicá-lo ao jogo n,m,k.

Aqui está uma descrição do jogo, de acordo com a wikipedia[1]:

“An m,n,k-game is an abstract board game in which two players take turns in placing a stone of their color on an m-by-n board, the winner being the player who first gets k stones of their own color in a row, horizontally, vertically, or diagonally. Thus, tic-tac-toe is the 3,3,3-game (…)”

Descrição do algoritmo usado:

Para resolver este problema optamos por implementar o algoritmo MCTS, este algoritmo é composto essencialmente por 4 fases: selection, expansion, simulation e backpropagation. Também é importante termos noção do que é o UCB (Upper confidence bound), uma vez que este algoritmo utiliza essa expressão para encontrar um equilíbrio entre a exploitation e exploration, ou seja, para ir explorando os nodes promissores, mas não ignorando os nodes ainda não explorados, uma vez que pode haver resultados mais promissores em nodes ainda não explorados. A fórmula matemática para este UCB[2] é:

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$

Onde wi = numero de vitórias para o node considerado após i movimentos,

ni = numero de simulações para o node considerado após i movimentos,

Ni = numero de simulações para o node pai do node considerado após i movimentos,

c = parâmetro de exploração, raiz de 2, no nosso caso.

Vamos então explicar cada uma destas fases individualmente para uma melhor perceção do algoritmo.

Selection: Nesta fase o algoritmo começa no Node de raiz fornecido, e vai ver qual dos nodes filhos do node raiz é que tem um valor UCB mais elevado. Este processo de seleção vai-se repetindo ate se chegar a um node folha. Quando se chega a um node folha, verifica-se se este node já foi visitado alguma vez ou não, se é a primeira vez a ser visitado, passa-se para a fase da simulation, caso contrario passa-se para a fase expansion.

Expansion: Nesta fase o algoritmo vai adicionar os nodes filhos do node selecionado, à árvore e de seguida seleciona-se um desses filhos aleatoriamente, isto porque o UCB score dos filhos é inicialmente igual entre todos eles, e passa-se para a fase da simulation.

Simulation: Nesta fase o algoritmo vai simular jogadas aleatórias a partir do node dado no input ate ao jogo chegar ao estado terminal. De seguida apenas retorna um certo valor, no nosso caso metemos 3 diferentes para os 3 estados terminais possíveis. 1 caso o estado terminal corresponda a uma vitória, 0 caso o estado terminal corresponda a um empate e -1.75 caso corresponda a uma derrota. De seguida passa-se para a fase da backpropagation.

Backpropagation: Por fim nesta última fase o algoritmo vai atualizar os valores de winScore e de VisitCount desde o node folha selecionado ate ao node raiz.

Depois o que algoritmo faz é repetir essas fases um certo numero de vezes, ou então durante um certo tempo, no nosso caso optamos por meter um limite temporal de 1,25 segundos.

Unit Tests

Ver os ficheiros: BoardTest.java e CarloAgentTest.java

Opções de design:

Para este projeto optamos por usar o design: “Strategy design” isto porque há inúmeros problemas que podem utilizar o nosso algoritmo MCTS e utilizando o Strategy design pattern, em vez de se implementar um algoritmo de adversarial search para um problema/jogo em específico, implementa-se um algoritmo de adversarial search geral, onde não é preciso se mexer independentemente do problema a ser resolvido. Isto consegue-se fazer através do uso de uma interface, neste projeto o nome da nossa interface é Ilayout, e se for necessário explorar mais um problema da natureza adversarial Search, basta fazer uma nova classe que implemente o Ilayout e os algoritmos da classe CarloAgent, neste caso, vão funcionar para a nova classe.

Código

Ver pasta “src”.

Javadoc

Ver pasta “doc”.

UML

Para ver com mais detalhe, ver pasta uml.

Resultados, análise e discussão

Os resultados deste algoritmos são muito bons, isto porque o nosso algoritmo, mesmo com um tempo de processamento de 1,25 segundos, consegue ganhar ou no pior caso empatar contra um ser humano normal ou contra um agente que apenas faz jogadas random. É claro que aumentado o tempo de processamento do algoritmo os resultados iram ser melhores, mas como no futuro nós queremos utilizar este algoritmo numa situação onde estamos limitados temporalmente, considerámos 1,25 segundos um tempo razoável.

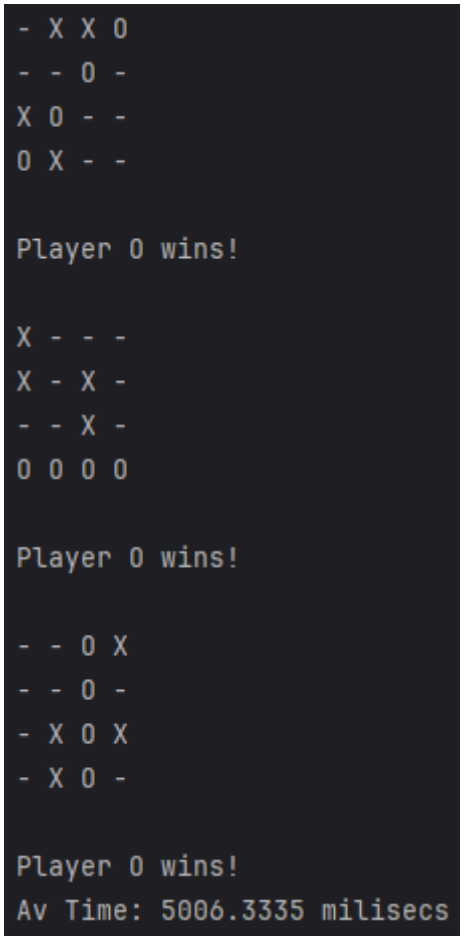
Algo interessante deste algoritmo é que quanto mais avançado o jogo estiver melhor é o seu desempenho, uma vez que há menos estados a ser explorados, daí também termos optado por uma limitação temporal em vez de uma limitação de iterações.

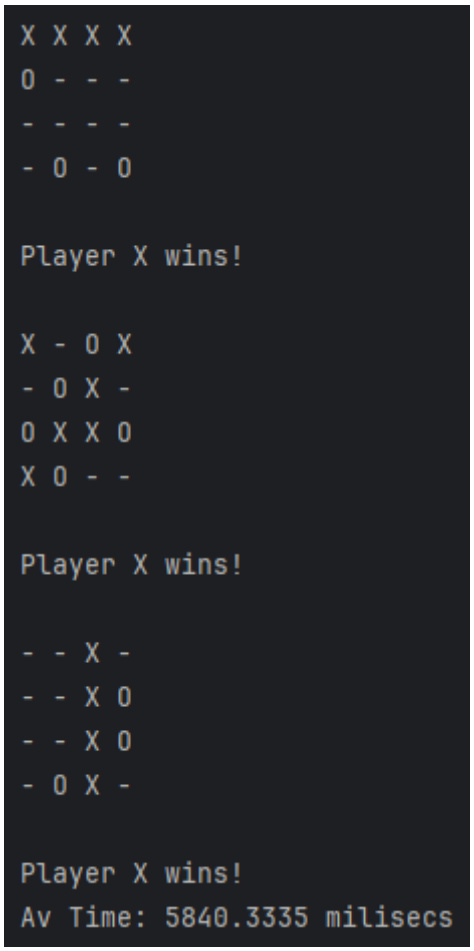
.

Resultados de 2 jogos entre um agente que utiliza MCTS e outro que joga aleatóriamente

Primeiro caso (esquerda) : agente aleatório = X , agente MCTS = O.

Segundo caso (direita) : agente MCTS = X, agente aleatório = O.





Conclusão e comentários

Com este trabalho aprimorámos o nosso conhecimento relativamente a algoritmos de procura adversarial, mais especificamente sobre o algoritmo MCTS, e também sobre o jogo n,m,k.

Um vídeo que achamos interessante e que ajudou muito na implementação do algoritmo: https://www.youtube.com/watch?v=UXW2yZndl7U&t=803s

Bibliografia

1 – https://en.wikipedia.org/wiki/M,n,k-game

2 - https://en.wikipedia.org/wiki/Monte_Carlo_tree_search