UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

Licenciatura em Engenharia Informática

Unidade Curricular de

Inteligência Artificial

(3º ano/1semestre – 2020 / 2021)

TIC TAC TOE

Relatório Técnico – Mini Projeto

Docentes:

José Valente de Oliveira

Discentes:

PL1 Grupo nº5

Guilherme Correia, nº61098

Henrique Cruz, nº61099

Ricardo Rosa nº 62461

Conteúdo

Introdução

Este trabalho tem como objetivo aprofundar os conhecimentos sobre inteligência artificial, implementando o algoritmo **Monte Carlo Tree Search** (MCTS) ao jogo do galo. Este jogo envolve dois jogadores e é formado por um tabuleiro 3 por 3 e tem como objetivo colocar três peças iguais em linha, quer horizontal, vertical ou diagonal.

Devido à simplicidade do jogo do galo, é frequentemente usado como uma ferramenta pedagógica para ensinar os conceitos de bom desportivismo e o ramo da inteligência artificial que trata da busca em árvores de jogo . É simples escrever um programa de computador para jogar o jogo do galo perfeitamente ou enumerar as 765 posições essencialmente diferentes ou os 26.830 jogos possíveis até rotações e reflexões neste espaço. Se jogado de forma ideal por ambos os jogadores, o jogo termina sempre em empate, tornando o jogo do galo um jogo fútil.

Figura 1-jogo tic tac toe

Uml - diagrama de classes

UML inicial de acordo com a estrutura utilizado no tutorial 1:

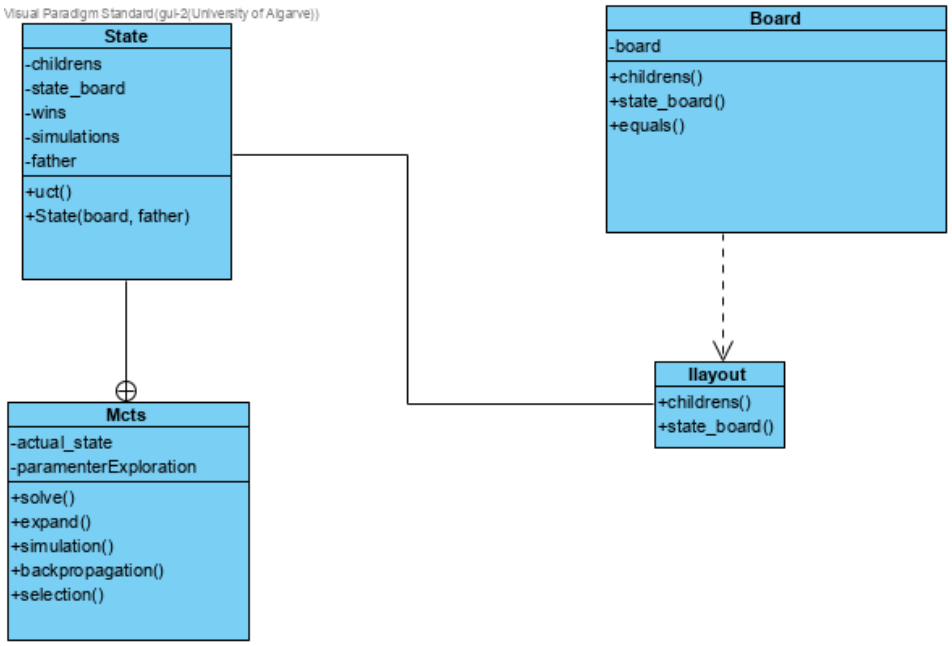

Figura 2-uml inicial

Classe State

Utc()- calcula o valor limite confiança superior usando a fórmula

State()- construtor da classe, precisa da configuração e do seu pai

Classe MCTS

Expand()- expande um nó

Simulation()- simula um conjunto de nós

Backpropagation()- retropropaga um resultado ate ao topo da arvore

Selection()- seleciona um nó

Classe Board

Children()-gera os filhos de uma board

State_board()-devolve o resultado de uma board

Durante o desenvolvimento do programa foi necessário adicionar mais variáveis e métodos para ajudar a finalizar o projeto.

UML final (objectaid):

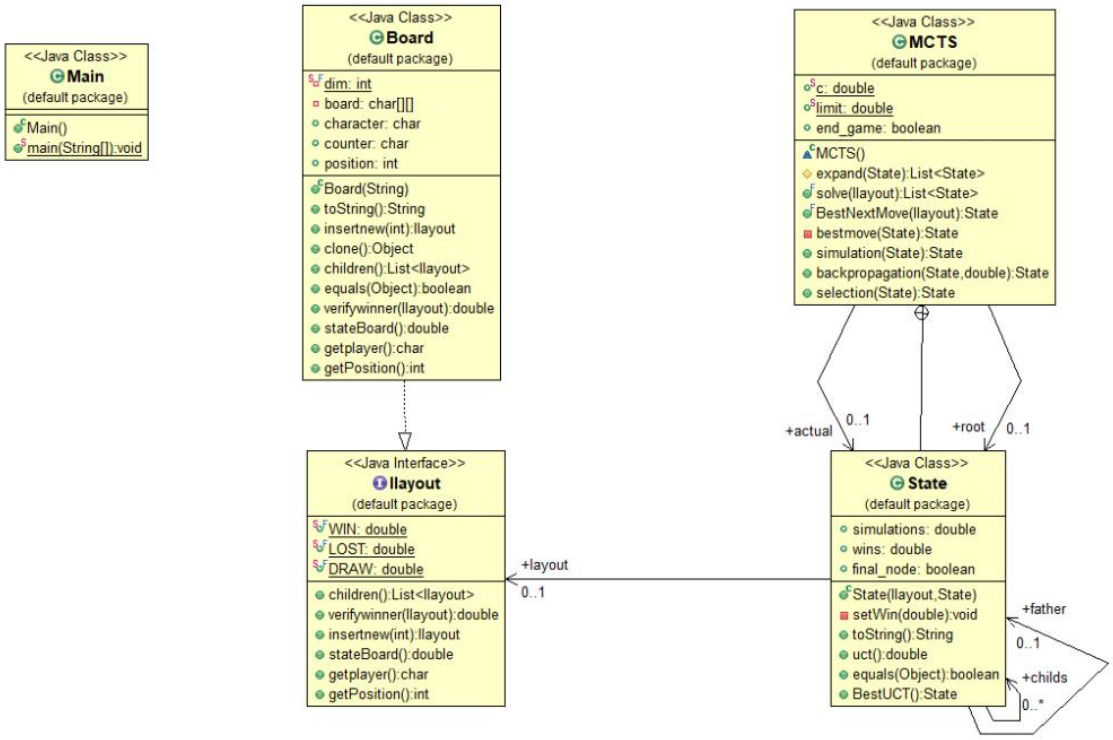

Figura 3-uml implementação

Implementação

Algoritmo [1]

Monte Carlo Tree Search (MCTS) é uma técnica de pesquisa na área de inteligência artificial. É um algoritmo de busca probabilístico e heurístico que combina as implementações clássicas de busca em árvore com princípios de aprendizagem máquina.

Na pesquisa em árvore, existe sempre a possibilidade de que a melhor ação atual não seja realmente a mais ótima. Em tais casos, o algoritmo MCTS torna-se útil, pois continua a avaliar outras alternativas periodicamente durante a fase de aprendizagem, executando-as, em vez da estratégia ótima percebida atual. Ele explora as ações e estratégias consideradas as melhores até agora, mas também deve continuar a explorar o espaço local de decisões alternativas e descobrir se elas poderiam substituir as melhores atuais.

A exploração ajuda a explorar e descobrir as partes inexploradas da árvore, o que pode resultar na descoberta de um caminho mais ideal. Em outras palavras, podemos dizer que a exploração expande a largura da árvore mais do que sua profundidade. A exploração

pode ser útil para garantir que o MCTS não ignore nenhum caminho potencialmente melhor.

O mais importante de entender é que o objetivo do algoritmo é **construir uma árvore de jogo**. A cada iteração o algoritmo irá jogar um nó novo na árvore, referente a uma decisão que pode ser tomada em algum momento do jogo. O segredo é que o MCTS irá focar na construção nos caminhos que contém as decisões mais promissoras.

Funcionamento [4]

No MCTS, os nós são os blocos de construção da árvore de pesquisa. Esses nós são formados com base no resultado de uma série de simulações. O processo de pesquisa em árvore pode ser dividido em quatro etapas distintas: seleção, expansão, simulação e retropropagação.

Uma decisão ser promissora:

- tomar essa decisão agora levará a mais vitórias no futuro;
- essa decisão não foi analisada muito a fundo.

Esses vão ser os dois fatores analisados na hora de escolher qual decisão será expandida

Mais especificamente, cada nó de decisão possui dois atributos:

- valor: um número que representa o quão atrativa é aquela decisão. Um valor alto significa que a decisão é boa;
- visitas: um número que representa quantas vezes aquela decisão já foi analisada. Se um nó tiver um número de visitas baixo significa que não demos muita atenção àquela decisão: isto é, ela tem potencial inexplorado.

Seleção [11]

Neste processo, o algoritmo percorre a árvore atual do nó raiz usando uma estratégia específica. A estratégia usa uma função de avaliação heurística para selecionar de forma otimizada os nós com o maior valor estimado. O MCTS usa a fórmula “Upper confidence Bound” (UCB) aplicada a árvores como estratégia no processo de seleção para atravessar a árvore.

Na etapa de seleção, o algoritmo irá partir da raiz e percorrer a árvore descendo pelos nós e alternativamente escolhe o nó de maior UCB e de seguida o de menor UCB (minimax), até chegar em um nó que satisfaça uma das seguintes 2 condições:

- o nó ainda pode ser expandido;
- o nó é um estado terminal (vitória, derrota ou empate).

Fórmula do UCB

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

Figura 4-formula do UCB

ni-número de simulações do nó atual

t-número de simulações do nó pai

wi-nº de vitórias do nó atual

c-parâmetro de exploração

Expansão

Na etapa anterior, foi selecionado o nó mais promissor que ainda pode ser expandido. De seguida, foi adicionado os seus nós filhos à árvore.

Existem diversas formas de fazer isso, e essa é uma decisão de design do algoritmo. Um método possível é escolher aleatoriamente um dos filhos possíveis e adicioná-lo à árvore. Outro é adicionar todos os filhos possíveis simultaneamente.

Independente de qual for o método de expansão escolhido, no fim a etapa fará sempre mesmo: adicionar nós à árvore.

Simulação

Existem dois tipos de situação em que essa etapa começa.

- o nó atual é um nó recém-expandido:
- o nó atual é um nó terminal.

Nesta etapa, o nó recém-expandido irá fazer uma simulação aleatória até chegar a um nó terminal.

No segundo ponto é avaliado o estado do nó e avança para próxima etapa.

Retropropagação

A etapa de retropropagação é responsável por propagar os resultados de uma simulação para os antecessores do nó simulado. Portanto:

Isso significa iterar por todos os nós descendentes do nó recém-expandido, e fazer duas atualizações:

- o atributo visitas deve ser incrementado em 1;
- o atributo valor deve ser incrementado de acordo com o resultado da simulação.

Depois dessas 4 etapas, a iteração do algoritmo chegou ao fim.

Obviamente, fazer uma única iteração não é bom o suficiente, o ideal é fazer milhares, para que se possa obter a melhor decisão possível. Como foi dito anteriormente, temos que estabelecer algum critério de paragem (ex.: número de iterações), e assim que esse critério for atingido, é considerado a árvore como "finalizada".

Algoritmo implementado (Pseudo Code)

```
actua->State
limite,res->inteiro

bestmove(Board)
    actual->inicializar no raiz
    limite-> define o user
    while(n simluacoes do no raiz< limite)
        if(actual ter filhos)
            actual->selection(actual)
        if(actual nao é nó final)
            filhos no actual->expand(actual)
        simulation(actual)

selection(State)
    while(actual tiver filhos)
        if(maximizar uct)
            actual->filho com maior uct
        if(minimizar uct)
            actual->filho com menor uct
    return actual

expand(State)
    return lista de movimentos possiveis

simulation(State)
    for(filhos do actual)
        res->resultado das escolhas semi-aleatorias
        backpropagation(filho,res)

backpropagation(State,inteiro)
    while(actual nao for o no raiz)
        actual vitoria->soma o resultado
        actual simulaçaoes-> incrmenta
        actual-> pai do actual
```

Figura 5-pseudo-codigo

Desenvolvimento

Inicialmente foi implementado, na classe Board o equals, que verificava as simetrias, desta forma iria reduzir o número de filhos e consequentemente o programa seria mais rápido pois a árvore teria menos nós. Segue um exemplo dos nós gerados da raiz utilizando a implementação descrita: [9]

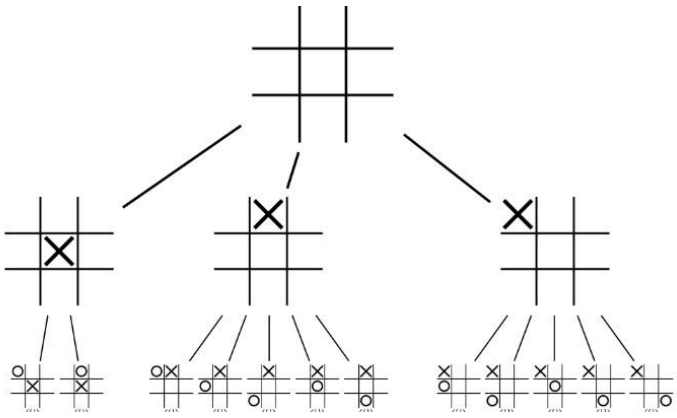

Figura 6-arvore do jogo

Uma das duas estratégias que foi implementada, consistia em dar valores às vitórias, empates e derrotas, 1, 0 e -1, respetivamente. Esta técnica é usada em muitos jogos multiplayer que usam o algoritmo MCTS. Usando esta estratégia em conjunto com a tree policy que é usada, que consistia em escolher o maior UCT nos casos que era para maximizar e nos casos de minimizar escolhia o menor, mas desta forma influenciava a escolha dos nós após algumas interações passado a escolher só 1 no (o que tem mais simulações).

Nesta estratégia durante a fase de retropagação foi trocado o sinal, se houvesse uma vitória ou derrota de um nó, ou seja, imaginando que o "X" ganhou, durante esta fase todos os nós em que o "X" jogasse daríamos o valor de 1, enquanto que quando houvesse um nó em que fosse a "O" jogar daríamos o valor de -1. [5]

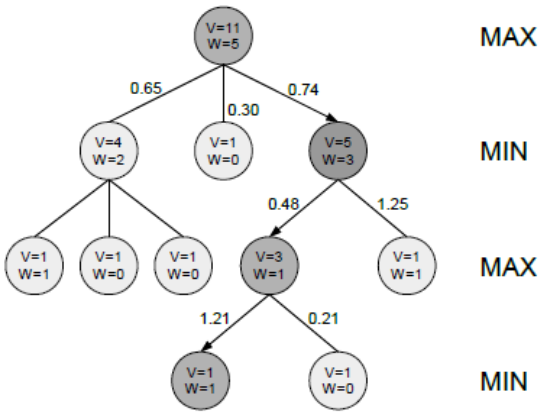

Figura 7-tree policy

Depois, dessa implementação, foi alterado os valores das vitórias, derrotas e empates para 1, 0 e 0.5, respetivamente. Assim seria mais fácil de implementar em relação ao anterior, a ideia continua a ser mesma, mas neste caso não existiria troca de sinal, seria

retropropagado o valor independentemente se o nó é um movimento do jogador ou do oponente.

No entanto não se conseguiu chegar a um resultado perfeito. Como tinha sido comunicado com o professor, no momento de criação da árvore, quando é o MIN a jogar, numa situação em que os movimentos em escolha obtiveram vitória, escolhendo o primeiro movimento do conjunto e realizando as outras fases do MCTS (expansão, simulação e retroprapagação), mas numa segunda escolha desses movimentos, o MIN irá escolher o mesmo movimento visto que, a função UCB contém o termo de exploração ($c*\sqrt{\frac{ln(T)}{n_i}}$), este que quantas mais simulações tiver menor será o parâmetro de exploração, com isto e usando uma simulação que para quando um dos filhos é um nó terminal leva que em certas situações ele escolha movimentos que não são os melhores para o oponente e leva a que a estatística dos nós não convirja para o nó que seria a melhor opção, resultando numa árvore com ramos mal explorados.

De seguida foi alterado a simulação para que caso não se encontra um nó final, dos nós filhos que estavam no momento a ser estudado no percurso da simulação, iria ver se algum dos netos era um nó final, neste caso escolheria um nó que não tivesse filhos que terminassem a simulação. Permitido assim que durante a simulação, realiza-se jogadas parecidas a um jogador racional para ambos os lados.

Por fim a segunda estratégia, consiste em aumentar o valor do UCB, isto é aumentar a confiança dos nós mais promissores, como é esperado usado o MCTS, e em vez na fase de minimizar iriamos maximizar UCB no ponto de vista do oponente, isto é, iriamos assim estar a minimizar a possibilidade de o atual jogador, isto resume-se a seguinte formula: [10]

$$UCTmax + UCTmin = 0$$

Resultando assim, que a construção da árvore e os valores das simulações fosse convergir para os melhores nós, caso fosse oponente ou não. Permitido assim, que as somas dos UCBs resultasse em 0, simbolizado assim que ambos os jogadores, caso estivessem a jogar de forma racional, resultaria num empate.

Para critério de escolha do movimento a meter em jogo é usado o que tem mais visitas. [3]

Testes

Os testes foram importantes para o desenvolvimento e levantamento dos resultados, pois permitiram ver os pontos em que o nosso programa falhava, no casos das escolhas dos nós que estão no conjunto de funções *testBestandWorstUCT*, o caso do *backpropagantion* foi importante para ver como eram distribuídos os resultados ate a raiz, estão no conjunto de funções *testback,* nas simulações foi importante para ver se o algoritmo chegava a um resultado, estão no conjunto *testsimulation,* e por fim foi feita uma função para ver o nº de vezes que o pedia usando a 1º estratégia contra ele própria e contra a outra, ou vice-versa, é a função *testPrecision.*

Estes testes encontram-se no ficheiro anexo tests.java.

Resultados

1º versão do projeto

Parâmetro de exploração

No seguinte gráfico é mostrado o resultado no intervalo de valores [0,2] usado na primeira estratégia, onde foi usado 1000 simulações, é o valor mais adequado, para cada um dos jogadores, e para cada um dos valores do parâmetro de exploração foram feitos 1000 jogos.

Figura 8-grafico parâmetro exploração/vitórias

Com este gráfico é possível observar que quanto maior for o valor do parâmetro pior resultados obterá, isto acontece devido as várias explorações que precisa fazer, e usando a estratégia 1, leva que os nos passem a seguir diferentes. Mas se o valor do parâmetro for muito próximo de 0 piores são os resultados. Por isso o resultado que pareceu melhor foi um resultado entre 0.1 e 0.2, que esta próximo de 0.18.

Número de simulações

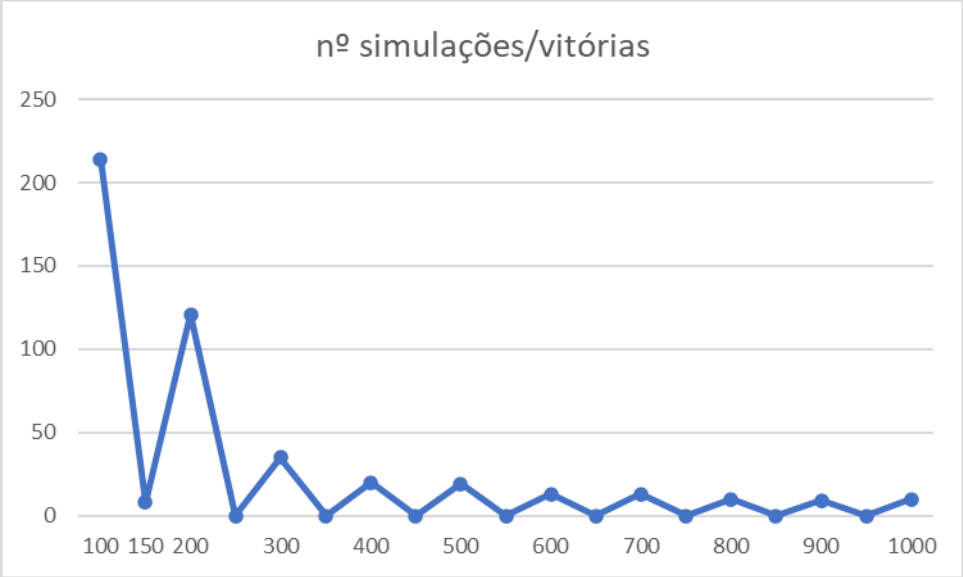

Figura 9-gráfico nº simulações/vitórias

Quando ao número de simulações, podemos concluir que os resultados foram próximos do esperados visto que ao longo do número de simulações o valor das vitorias diminui significado assim que o bot vai jogando melhor. Existe uns pontos mais baixos porque por vezes precisa de poucas simulações, porque na fase do minimizar ele escolhe um no e as vezes pode ser o correto.

2º versão do projeto

Parâmetro de exploração

No seguinte gráfico é mostrado o resultado de usar um intervalo de valores [0,2] usado na primeira estratégia, onde foi usado vários valores para o nº de simulações para cada um dos jogadores, e para cada um dos valores do parâmetro de exploração foi feito 1000 jogos.

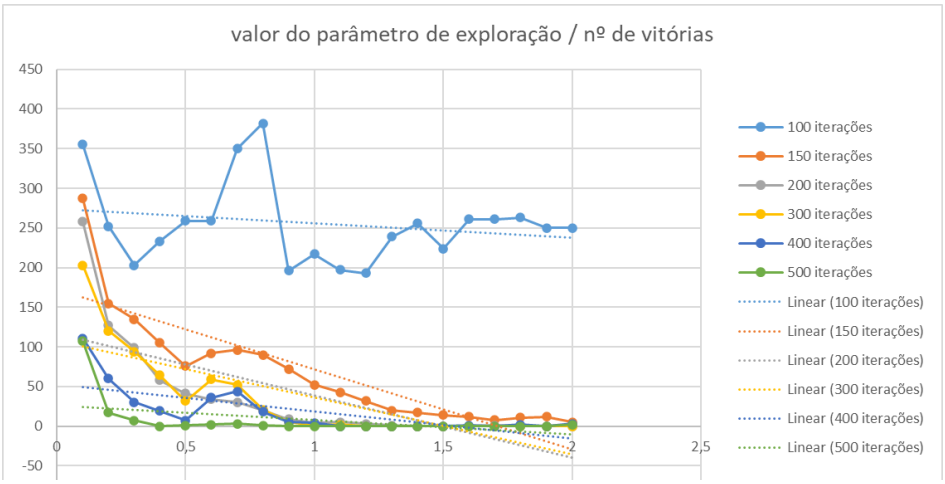

Figura 10-grafico parâmetro exploração/nº vitórias

Número de simulações

A partir deste gráfico é possível concluir que quanto mais iterações melhores são os resultados do "bot" o que era de esperar, pois a árvore é mais informada quanto mais iterações forem realizadas. Também é possível verificar que o valor do parâmetro de exploração (c) é melhor a partir do valor 1.

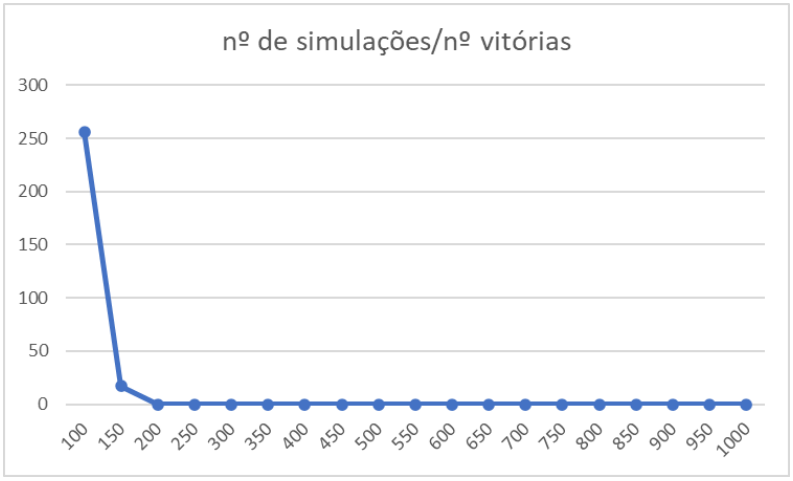

Figura 11-gráfico nº simulações/nº vitorias

Com estes resultados, concluímos que a 2º versão é melhor porque teve resultados mais coerentes.

Eficiência

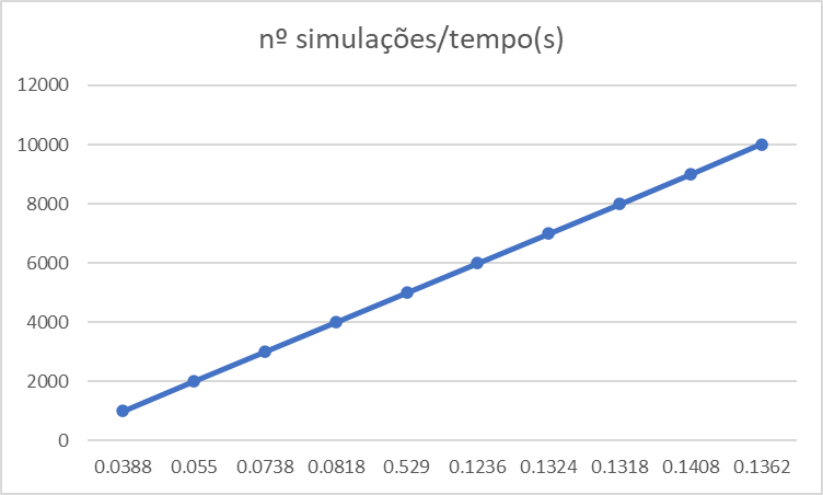

Figura 12-gráfico de eficiência

O tempo de execução parece ser bom, apesar de aumentar de forma linear.

Melhorias

A eficácia poderia ter sido melhorada, provavelmente pode haver comparações que são feitas, que podiam ser eliminadas de forma a acelerá-lo.

Conclusão

Com este trabalho foi possível concluir que o MCTS é um algoritmo poderoso. E que não basta abranger alguns pontos, cruciais no jogo mas obter bons resultados, visto que nessas situações iriamos estar a por em causa o uso do algoritmo.

À primeira vista, é difícil confiar que um algoritmo baseado em escolhas aleatórias pode levar a IA inteligente. No entanto, a implementação cuidadosa do MCTS pode realmente fornecer uma solução que pode ser usada em muitos jogos, bem como em problemas de tomada de decisão.

Apesar de ambas as versões terem obtido bons resultados, a 1º versão produzia uma árvore incorreta e os resultados obtidos não foram dos bons quanto os da 2º versão, este pelo contrário produz uma arvore que vai ao encontro do que pedido pelo Mcts e mostra resultados esperados.

Bibliografia

1) https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/
2) https://towardsdatascience.com/monte-carlo-tree-search-158a917a8baa
3) https://ai.stackexchange.com/questions/16905/mcts-how-to-choose-the-final-action-from-the-root
4) https://en.wikipedia.org/wiki/Monte_Carlo_tree_search
5) https://gamedev.stackexchange.com/questions/54294/monte-carlo-tree-search-for-simultaneous-multiplayer-game/55482
6) https://pdfs.semanticscholar.org/ccc2/ea7c4df7827ecd3118b0e4da7829201d28aa.pdf
7) https://en.wikipedia.org/wiki/Tic-tac-toe
8) https://www.cs.swarthmore.edu/~mitchell/classes/cs63/f20/reading/mcts.html
9) Slide adversarial search, José Valente Oliveira,IA 2020-2021
10) https://vgarciasc.github.io/mcts-viz/
11) Slide MCTS, José Valente Oliveira, IA 2020-2021