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

Bruno Susana, nº61024

## Resumo

## Introdução

Este trabalho tem como objetivo aprofundar os nossos 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, tem como objetivo colocar três peças iguais em linha, quer horizontal, vertical ou diagonal.

Devido à simplicidade do tic tac toe, é frequentemente usado como uma ferramenta [pedagógica](https://en.wikipedia.org/wiki/Pedagogical) para ensinar os conceitos de bom [desportivismo](https://en.wikipedia.org/wiki/Sportsmanship) e o ramo da [inteligência artificial](https://en.wikipedia.org/wiki/Artificial_intelligence) que trata da busca em [árvores](https://en.wikipedia.org/wiki/Game_tree) de [jogo](https://en.wikipedia.org/wiki/Game_tree) . É simples escrever um [programa de computador](https://en.wikipedia.org/wiki/Computer_program) para jogar o tic tac toe perfeitamente ou enumerar as 765 posições essencialmente diferentes ou os 26.830 jogos possíveis [até](https://en.wikipedia.org/wiki/Up_to) 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](https://en.wikipedia.org/wiki/Futile_game)



Figura -jogo tic tac toe

**Monte Carlo Tree Search**, ou MCTS, é um algoritmo utilizado para fazer busca em árvores que são particularmente grandes. Ao empregar a "aleatoriedade direcionada" comum aos métodos de Monte Carlo, o algoritmo resolve algumas das dificuldades inerentes ao percurso de árvores com muitos nós.

Em que situações um algoritmo como o MCTS é bem-vindo? Uma das aplicações mais famosas é na área de inteligência artificial para jogos. Nessa área, é comum ter como objetivo a tarefa de "buscar a melhor jogada, dentro de todas as jogadas possíveis". Esse problema em si já é uma busca, mas se representarmos todas as decisões possíveis do jogo como uma árvore, o problema acaba especificamente numa busca em árvore.

O jogo do galo é um caso trivial: são apenas 9 espaços, e a cada turno o jogador tem menos decisões possíveis para analisar. Nesse caso, é possível construir uma árvore com todos os tabuleiros possíveis, e simplesmente consultá-la toda a vez que formos fazer uma jogada.

Felizmente, o que o MCTS nos permite fazer é justamente obter uma boa decisão sem ter que expandir a árvore inteira.

**Funcionamento**

De forma bem informal, o que o algoritmo faz é muito parecido com o que muitos de nós fazemos enquanto estamos pensando na próxima jogada: analisar aos poucos a árvore de jogo procurando as decisões mais promissoras. É como se ele estivesse falando: "eu posso jogar um X aqui, mas o oponente vai jogar a O aqui, e eu vou perder... então é melhor eu não jogar o X aqui... onde eu vou jogar então? vamos supôr que eu jogue aqui..." e assim por diante. Basicamente, vamos continuando fazendo isso até atingir o critério de paragem que nós escolhemos - como um tempo máximo de execução, ou um número fixo de iterações.

O mais importante de entender de cara é que o objetivo do algoritmo é **construir uma árvore de jogo**. A cada iteração o algoritmo vai 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 vai focar a construção nos caminhos que contém as decisões mais promissoras.

Mas o que significa uma decisão ser *promissora*? Significa duas coisas diferentes:

1. tomar essa decisão agora levará a mais vitórias no futuro (é uma *boa* decisão), ou
2. essa decisão não foi analisada muito a fundo (tem *potencial inexplorado*).

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

Mais especificamente, cada nó de decisão possui dois atributos:

1. *valor*: um número que representa o quão atrativa é aquela decisão. Um valor alto significa que a decisão é boa.
   * no exemplo do jogo da velha, podemos modelar isso como o *saldo de vitórias* atingidas caso o jogador tome essa decisão. Por exemplo, se uma vitória vale 1, um empate vale 0,5 e uma derrota vale 0, evidentemente uma decisão boa teria um saldo positivo e uma decisão ruim teria um saldo negativo.
2. *visitas*: um número que representa quantas vezes aquela decisão já foi analisada. Se um nó tiver um número de visitas baixo (quando comparado ao resto da árvore, claro), significa que não demos muita atenção àquela decisão: isto é, ela tem potencial inexplorado.

**Etapas**

O algoritmo pode ser dividido em 4 etapas:

1. seleção
2. expansão
3. simulação
4. retropropagação

### 1. seleção

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

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

### 2. expansão

Na etapa anterior, selecionamos o nó mais promissor que ainda pode ser expandido. Agora, vamos adicionar 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 a mesma coisa: adicionar nós à árvore.

### 3. simulação

Existem dois tipos de situação em que essa etapa começa.

1. o nó atual é um nó recém-expandido, ou
2. o nó atual é um nó terminal (a etapa de expansão foi pulada).

Nesta etapa, o nó recém-expandido irá fazer uma simulação semi-aleatória até chegar a um nó terminal, em algumas situações caso um no tenha um filho que seja no final ele ira escolhê-lo.

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

### 4. 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:

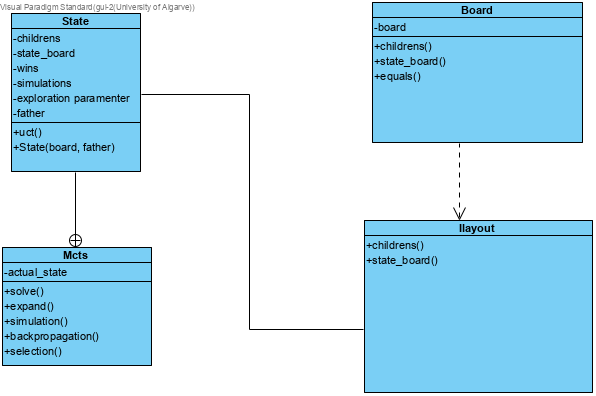
1. o atributo *visitas* deve ser incrementado em 1;
2. o atributo *valor* deve ser incrementado de acordo com o resultado da simulação (p. ex.: vitória 1, derrota 0, empate 0,5)

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

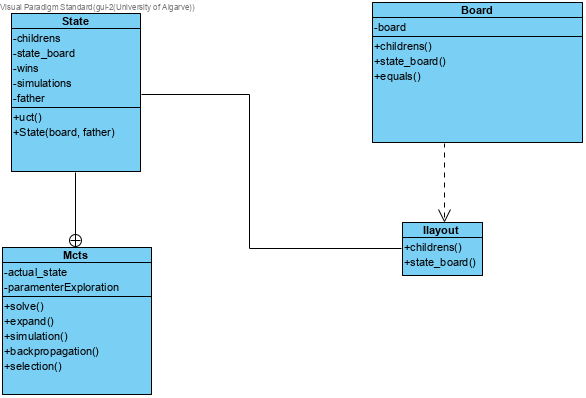
Obviamente, fazer uma única iteração não é bom o suficiente - o ideal é fazermos milhares, para que possamos obter a melhor decisão possível. Como dissemos anteriormente, temos que estabelecer algum critério de paragem (p. ex.: número de iterações), e assim que esse critério for atingido, consideramos nossa árvore como "finalizada".

**All design options taken**

A estrutura pedida para o problema era que foi usada no tutorial 1, consistia na seguinte imagem:



Durante o desenvolvimento, houve algumas alterações nas funções e paramentos, no caso do parâmetro de exploração, visto que é um atributo importante no algoritmo e fazia mais sentido estar na classe Mcts do que State.



**Desenvolvimento do projeto**

Inicialmente, ao procurar na internet a melhor opção foi usar o MCTS e selecionar sempre o nó com o melhor UCT, implementamos essa estratégia e até tivemos resultados relativamente bons, no entanto tivemos que alterar para o minimax, como era pedido no problema.

Em segundo, design foi implementado, que ia ao encontro do enuciado do problema, consistia em dar valores as vitorias, empates e derrotas, 1, 0 e -1, respetivamente. Esta técnica é usada em muitos jogos multiplayer que usam o algoritmo MCTS. Esta forma iria sem útil no sentido que iriamos influenciar na escolha do UCT menor, visto que caso encontrasse uma derrota, iria mais vezes visita-lo tornado o nó pai, menos promissor, mas devido alguns erros de calculo, ou de implementação não deve sucesso.

Depois, dessa implementação, foi alterado os valores das vitorias, 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 fosse do oponente ou não. Desta forma os resultados melhoram mas ainda assim não eram os desejados.

Nesta fase também foram feitas algumas alterações