MINISTÉRIO DA EDUCAÇÃO
Instituto Federal de Educação, Ciência e Tecnologia do Sul de Minas
IFSULDEMINAS - Campus Poços de Caldas
Avenida Dirce Pereira Rosa, 300. Poços de Caldas/MG. CEP 37713-100 Fone: (35) 3713-5120

Matheus Paudarco da Silveira

Inteligência Artificial - Jogo da velha implementando o algoritmo MiniMax

O presente documento detalhará o desenvolvimento do jogo da velha, será relatado as tecnologias aplicadas em sua programação, as estruturas de dados utilizadas, a complexidade de cada função e da aplicação em geral, e demais informações que o autor julgar necessário.

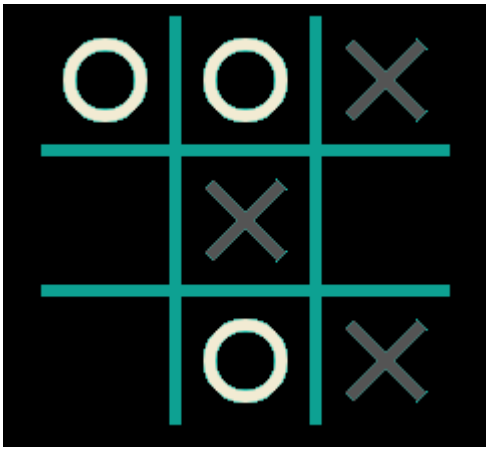
O *game* foi desenvolvido na linguagem de programação Java, onde o código utilizou o paradigma da programação orientada a objetos.

O algoritmo funciona da seguinte maneira:

Há uma interface gráfica que é o *main* da aplicação, esta, desenhada no *software* Netbeans. Na interface é colocados elementos para se obter do usuários as seguintes informações: nível de dificuldade e quem começará jogando (máquina ou usuário).

Se for o usuário, ele deve clicar onde irá jogar, e no botão que ele clicar há a função de fazer o movimento dele, marcando na interface e realizando o movimento de fato no "*backend"* da aplicação. Após isso, é chamado a função de análise de possibilidade onde é construída a árvore de todos os movimentos possíveis dentro do nível de dificuldade escolhido, lembrando que o máximo é 9 (mais difícil) e o mínimo é 1 (mais fácil). Lembrando que nos níveis finais, deve ser impossível vencer a máquina. Após a máquina realizar seu movimento, volta a ser a vez do jogador e assim tudo citado acima acontece novamente até que o tabuleiro fique sem novas possibilidades de jogadas ou alguém vença.

Algumas coisas são importantes salientar, como por exemplo, utilizando apenas o algoritmo MiniMax para a decisão de como a máquina deverá jogar, o algoritmo toma decisões imprecisas, como por exemplo: Se o jogo já estiver decidido a favor da máquina, ele jogará em qualquer lugar disponível, ele não acabará com o jogo de uma vez.



Repare que onde quer que o X jogue, ele está com o jogo decidido a seu favor, e assim, jogaria em qualquer lugar das três casas disponíveis. Para resolver este problema,

foi implementada uma segunda heurística onde o programa avaliará, além do resultado final do jogo com o MiniMax, a quantidade restante de movimentos a serem feitos.

O algoritmo MiniMax retorna 1, porque é o valor que diz que o movimento analisado gera uma vitória em favor da máquina, a segunda heurística acrescenta a esse valor a quantidade restante de movimentos, nesse caso 3. Caso ele jogue na casa superior esquerda livre, isso geraria uma vitória com uma quantidade de movimentos restantes igual a 0, por isso, a máquina sempre fechará o jogo o mais rápido possível.

Vale ressaltar também a necessidade de se implementar um pequeno algoritmo para tratar “jogadas viciadas”. Isso ocorre quando duas casas possuem o mesmo valor de peso, e a máquina jogará na última a entrar na variável de controle de posição a ser jogada. Para resolver isso foi implementado um *ArrayList* de possíveis jogadas, onde após ser preenchido com as jogadas possíveis, um método da classe Random fará a seleção de um valor para ser jogado.

Foram utilizados no desenvolvimento as seguintes estruturas de dados:

- **Lista** – foi usada a classe *ArrayList* no Java para a sua implementação. Foi implementada uma lista no algoritmo para armazenar as possíveis jogadas a serem realizadas pela máquina.
- **Árvore** – a árvore é implementada através da estrutura de repetição *for* para armazenar cada tabuleiro de possibilidade. O nó raiz implementa a classe “Matriz”, onde há um atributo “arvorePossibilidades” que é um *ArrayList* que armazena os filhos deste nó.

Na classe TelaInicial, não há *“fors”* sendo executados, com isso a demora, ou melhor dizendo, a complexidade da execução da classe não mudaria de acordo com a variável, tendo sua complexidade sendo definida por uma constante, ou seja, O(1).
Na classe Interface, há um for sendo executado, este for percorrerá toda o *ArrayList* de possibilidades, sendo assim sua repetição irá variar de acordo com n.
Na mesma classe, há mais um *for*, este é responsável por reiniciar o jogo, sendo assim, ele percorre todo o tabuleiro, sua complexidade varia de acordo com n também.
É importante lembrar, o *for* que avalia as possibilidades, é executado por 9 vezes no primeiro nível, e cada uma das 9 jogadas geram 8 jogadas, que geram 7 e assim por diante. A complexidade geral da função de possibilidades é n! (n fatorial).