Algoritmo Minimax

Situação Problema

Criação de uma inteligência artificial para o Jogo da Velha. Nós, humanos, temos a prioridade e começamos primeiro, com 'X', enquanto o computador jogará em seguida, com 'O'. Nesse caso, estamos falando de um jogo de dois jogadores e soma zero. São uma categoria específica de jogos nos quais dois participantes estão em competição direta, e o ganho de um jogador é exatamente compensado pela perda do outro. Em outras palavras, a soma total dos ganhos e perdas é sempre zero.

Modelagem Escolhida

Implementamos um jogo da velha simples, que é jogado selecionando-se o quadrado que o 'X' ou 'O' será colocado, através de um número de 1 a 0, sendo 1 o primeiro quadrado e 9 o último. Usamos o algoritmo Alpha-Beta-Minimax Search para que o computador encontre sempre a melhor jogada, em nossa implementaçãp, não é possível que a máquina seja vencida (nós não conseguimos) só é possível perder ou empatar (dar velha), como a árvore de jogo é pequena, com apenas 362,880 folhas (5,478 estados distintos) o jogo da velha é consideravelmente pequeno. Segue abaixo a implementação de nosso jogo da velha.

```
let board = [
    [' ', ' ', ' '],
    [' ', ' ', ' '],
    [' ', ' ', ' ']
];
```

Iniciamos o tabuleiro do jogo da velha como uma série de espaços que em seguida serão substituidos por 'X' ou 'O'

```
let currentPlayer = 'X';

function printBoard() {
    for (let i = 0; i < 3; i++) {
        console.log(board[i].join(' | '));
        if (i < 2) {
            console.log('---------');
        }
    }
}
```

Função que imprime o tabuleiro do jogo da velha no terminal

```
function checkWinner() {
    // Check rows
    for (let i = 0; i < 3; i++) {
```

```
        if (board[i][0] !== ' ' && board[i][0] === board[i][1] && board[i][1] ===
board[i][2]) {
            return true;
        }
    }

    for (let j = 0; j < 3; j++) {
        if (board[0][j] !== ' ' && board[0][j] === board[1][j] && board[1][j] ===
board[2][j]) {
            return true;
        }
    }

    if (board[0][0] !== ' ' && board[0][0] === board[1][1] && board[1][1] ===
board[2][2]) {
        return true;
    }

    if (board[0][2] !== ' ' && board[0][2] === board[1][1] && board[1][1] ===
board[2][0]) {
        return true;
    }

    return false;
}
```

Função que checa se há algum vencedor no jogo da velha

Implementação do Algoritmo Minimax

De acordo com o pseudo código proposto:

Algoritmo 03 - Minimax (versão alfa-beta)

```
function ALPHA-BETA-SEARCH(game, state) returns an action
   player ← game.TO-MOVE(state)
   value, move ← MAX-VALUE(game, state, −∞, +∞)
   return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
   if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
   v ← −∞
   for each a in game.ACTIONS(state) do
      v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
      if v2 > v then
         v, move ← v2, a
         α ← MAX(α, v)
      if v ≥ β then return v, move
   return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
   if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
   v ← +∞
   for each a in game.ACTIONS(state) do
      v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
      if v2 < v then
         v, move ← v2, a
         β ← MIN(β, v)
      if v ≤ α then return v, move
   return v, move
```

Heurística do Algoritmo Minimax

Usamos o algoritmo Alpha-Beta-Minimax Search, onde **para cada jogada**, o computador cria uma simulação de árvore de jogo, do jogador MAX (computador) contra o jogador MIN (oponente), que recebem seus nomes pois, caso o MAX chegue e uma vitória em uma das folhas da árvore de jogo, o computador considera a pontuação como +1, caso o jogador MIN ganhe, o computador considera a pontuação como -1 e caso haja empate, a pontuação é 0. segue abaixo nossa implementação da função Minimax

> Nossa função Minimax implementa as jogadas de MIN e MAX dentro do proprio fluxo, diferente do pseudocódigo

```
function minimax(tabuleiro, profundidade, player, alpha, beta) {
    if (checarVencedor()) {
        return player === COMP ? -1 : 1;
```

```
    } else if (isTabuleiroCheio()) {
        return 0;
    }

    if (player === COMP) {
        let maxEval = -Infinity;
        for (let i = 0; i < 3; i++) {
            for (let j = 0; j < 3; j++) {
                if (tabuleiro[i][j] === ' ') {
                    tabuleiro[i][j] = 'O';
                    let aval = minimax(tabuleiro, profundidade + 1, HUMANO, alpha,
beta);
                    tabuleiro[i][j] = ' ';
                    maxEval = Math.max(maxEval, aval);
                    alpha = Math.max(alpha, aval);
                    if (beta <= alpha) {
                        break;
                    }
                }
            }
        }
        return maxEval;
    } else {
        let minEval = Infinity;
        for (let i = 0; i < 3; i++) {
            for (let j = 0; j < 3; j++) {
                if (tabuleiro[i][j] === ' ') {
                    tabuleiro[i][j] = 'X';
                    let aval = minimax(tabuleiro, profundidade + 1, COMP, alpha,
beta);
                    tabuleiro[i][j] = ' ';
                    minEval = Math.min(minEval, aval);
                    beta = Math.min(beta, aval);
                    if (beta <= alpha) {
                        break;
                    }
                }
            }
        }
        return minEval;
    }
}
```

> A busca alfa–beta atualiza os valores de Alpha e Beta à medida que avança e poda os ramos restantes em um nó (ou seja, encerra a chamada recursiva) assim que o valor do nó atual é conhecido por ser pior que o valor atual de Alpha ou Beta para MAX ou MIN, respectivamente.
> Passamos o valor -Infinito para Alpha e Infinito para Beta e em seguida alteramos os valores de Alpha e Beta a cada interação.
> Caso seja jogada do MAX, Alpha recebe max(Alpha, aval) onde aval é o valor obtido ao fim da árvore de jogo.

> Caso seja jogada do MIN, Beta recebe min(Beta, aval) onde aval é o valor obtido ao fim da árvore de jogo.

Segue abaixo a função de jogada da Máquina, que utiliza a pontuação resultante da função minimax para escolher qual jogada fazer.

```
function jogadaMaquina() {
    let bestScore = -Infinity;
    let move;
    for (let i = 0; i < 3; i++) {
        for (let j = 0; j < 3; j++) {
            if (tabuleiro[i][j] === ' ') {
                tabuleiro[i][j] = 'O';
                let pontuacao = minimax(tabuleiro, 0, HUMANO, -Infinity,
Infinity);
                tabuleiro[i][j] = ' ';
                if (pontuacao > bestScore) {
                    bestScore = pontuacao;
                    move = { i, j };
                }
            }
        }
    }
    tabuleiro[move.i][move.j] = 'O';

    if (checarVencedor()) {
        imprimirTabuleiro();
        console.log(`Jogador ${currentPlayer} ganha!`);
        rl.close();
    } else if (isTabuleiroCheio()) {
        imprimirTabuleiro();
        console.log('É um empate');
        rl.close();
    } else {
        trocarJogador();
        jogadaHumano();
    }
}
```

Casos de Teste, Complexidade do Algoritmo e Discussão

Casos de Teste

Poderíamos mostrar diversos casos de teste, mas como os resultados são sempre vitória da máquina ou empate, optamos por mostrar somente alguns exemplos de cada caso.

Vitória da Máquina

```
Jogador X, digite sua jogada (1-9): 2
O | X |
---------
  |   |
---------
  |   |

Jogador X, digite sua jogada (1-9): 8
O | X |
---------
  | O |
---------
  | X |

Jogador X, digite sua jogada (1-9): 9
O | X |
---------
  | O |
---------
O | X | X

Jogador X, digite sua jogada (1-9): 4
O | X | O
---------
X | O |
---------
O | X | X
Jogador O ganha!
```

```
Jogador X, digite sua jogada (1-9): 2
O | X |
---------
  |   |
---------
  |   |

Jogador X, digite sua jogada (1-9): 8
O | X |
---------
  | O |
---------
  | X |

Jogador X, digite sua jogada (1-9): 9
O | X |
---------
  | O |
---------
O | X | X

Jogador X, digite sua jogada (1-9): 3
O | X | X
---------
O | O |
---------
O | X | X
Jogador O ganha!
```

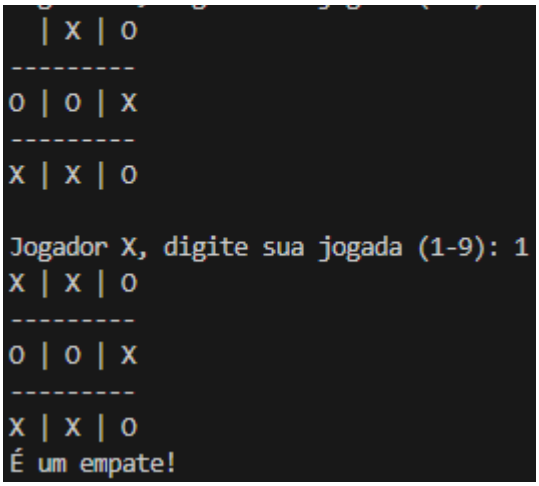
```
Jogador X, digite sua jogada (1-9): 1
X |   |
---------
  | O |
---------
  |   |

Jogador X, digite sua jogada (1-9): 2
X | X | O
---------
  | O |
---------
  |   |

Jogador X, digite sua jogada (1-9): 4
X | X | O
---------
X | O |
---------
O |   |
Jogador O ganha!
```

Empate

```
Jogador X, digite sua jogada (1-9): 3
  |   | X
---------
  | O |
---------
  |   |

Jogador X, digite sua jogada (1-9): 6
  |   | X
---------
  | O | X
---------
  |   | O

Jogador X, digite sua jogada (1-9): 1
X | O | X
---------
  | O | X
---------
  |   | O

Jogador X, digite sua jogada (1-9): 8
X | O | X
---------
O | O | X
---------
  | X | O

Jogador X, digite sua jogada (1-9): 7
X | O | X
---------
O | O | X
---------
X | X | O
É um empate!
```

```
Jogador X, digite sua jogada (1-9): 4
O |   |
---------
X |   |
---------
  |   |

Jogador X, digite sua jogada (1-9): 5
O |   |
---------
X | X | O
---------
  |   |

Jogador X, digite sua jogada (1-9): 8
O | O |
---------
X | X | O
---------
  | X |

Jogador X, digite sua jogada (1-9): 3
O | O | X
---------
X | X | O
---------
O | X |

Jogador X, digite sua jogada (1-9): 9
O | O | X
---------
X | X | O
---------
O | X | X
É um empate!
```

```
Jogador X, digite sua jogada (1-9): 6
  |   | O
---------
  |   | X
---------
  |   |

Jogador X, digite sua jogada (1-9): 2
  | X | O
---------
O |   | X
---------
  |   |

Jogador X, digite sua jogada (1-9): 8
  | X | O
---------
O | O | X
---------
  | X |

Jogador X, digite sua jogada (1-9): 7
```



Complexidade do Algoritmo

O algoritmo minimax realiza uma exploração completa em profundidade da árvore de jogo. Se a profundidade máxima da árvore for ***d*** e houver movimentos legais em cada ponto, então a complexidade de tempo do algoritmo minimax é $O(b^d)$.

A complexidade exponencial torna o MINIMAX impraticável para jogos complexos. Por exemplo, o xadrez tem um fator de ramificação de cerca de 35, e a profundidade média do jogo é de cerca de 80 jogadas, tornando inviável a busca de todos os estados.
No entanto, o MINIMAX serve como base para a análise matemática de jogos. Ao aproximar a análise minimax de várias maneiras, podemos derivar algoritmos mais práticos.

Discussão

O algoritmo minimax foi extremamente efetivo em nosso projeto, a jogada do computador é quase instantânea e não fomos capazes de derrotá-lo sequer uma vez. Como idea futura, seria interessante criar um front end, com reconhecimento de clicks, para tornar o jogo mais "User Friendly", devido ao curto prazo, não foi possível. Esse trabalho serviu para aumentar nossos conhecimentos de Teoria dos Jogos, Javascript e Busca Adversária.