UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO

TRABALHO SOBRE MÉTODOS DE BUSCA MINMAX PARA JOGO DA VELHA

VANESSA DANTAS DE SOUTO COSTA

Natal-RN
2018

VANESSA DANTAS DE SOUTO COSTA

TRABALHO SOBRE MÉTODOS DE BUSCA MINMAX PARA JOGO DA VELHA

Este relatório é referente ao trabalho desenvolvido na Unidade I da disciplina Controle Inteligente, correspondente à avaliação da 1º unidade do semestre 2018.2 da Universidade Federal do Rio Grande do Norte, sob orientação do **Prof. Fábio Meneghetti Ugulino de Araújo.**

Professor: Fábio Meneghetti Ugulino de Araújo.

Natal-RN
2018

RESUMO

Este trabalho tem como objetivo explicitar a implementação utilizada para desenvolvimento do **J**ogo da Velha embasado na estratégia de algoritmo conhecido como **M**INMAX. Ainda, foi solicitado que aplicassemos outros tipos de estratégia para dificultar o nível do jogo.

Palavras-chave: Hard Rules, MINMAX, Jogo da Velha, Métodos de Busca

Sumário

1 INTRODUÇÃO

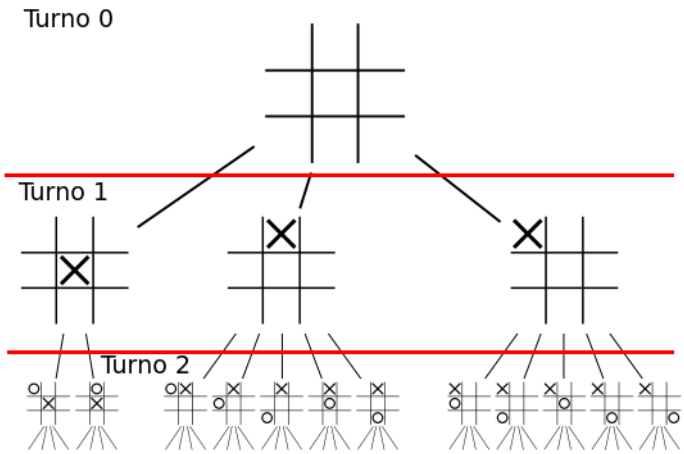
A inteligência artificial é um campo que está presente no nosso dia a dia, um exemplo disso, conforme abordado na disciplina, é em estratégias de jogos. Para tanto, podemos utilizar estratégias de busca em árvores para a implementação da inteligência do jogo. Alguns exemplos disso são o algorimo de A* e o algoritmo de busca MINMAX.

Nesse trabalho utilizaremos o algoritmo do MINMAX para implementação de um jogo da velha. Esse método de busca é relevante para o jogo da velha, pois permite a análise de várias possibilidades de ação, permitindo uma tomada de decisão embasada na maior chance de vitoria.

Com isso, esperamos gerar uma I.A. que possa competir ou superar a capacidade humana de acordo com a dificuldade desejada para o jogo.

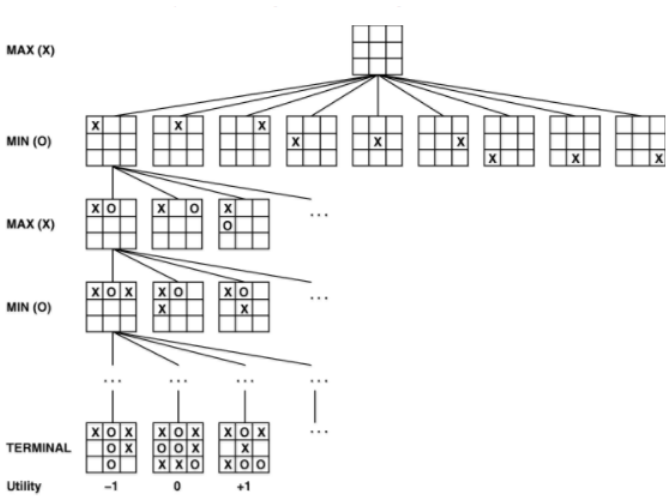
2 REFERENCIAL TEÓRICO

O MINMAX foi desenvolvido a priori para jogos no qual cada jogador pode ganhar, perder ou empatar, como é o caso do jogo da velha. O algoritmo minimax ajuda a encontrar a melhor jogada, ao se caminhar pelas opções válidas, a partir do fim do jogo. A cada passo, assume-se que o jogador maximizador está tentando maximizar as suas chances de ganhar, enquanto na próxima rodada o jogador minimizador está tentando minimizar as chances de isso acontecer (maximizando as chances de que ele próprio ganhe). O maximizador precisa escolher uma jogada que tem a maior dentre as menores pontuações que o minimizador pode fazer aquele ter.



MINMAX analiza a chances de cada jogador de vitoria.

Dada uma certa profundidade, o algorítmo do MINMAX irá caminhar pelos nós da árvore até a profundidade desejada checando os valores heuristicos de cada folha da seguinte maneira: O valor do nó pai é calculado à partir do menor(ou maior dependendo da heuristica adotada) valor dentre seus nós-filhos para ao final, escolher a situação que melhor se adeque ao resultado desejado.



MINMAX em nível maior de profundidade

3 METODOLOGIA

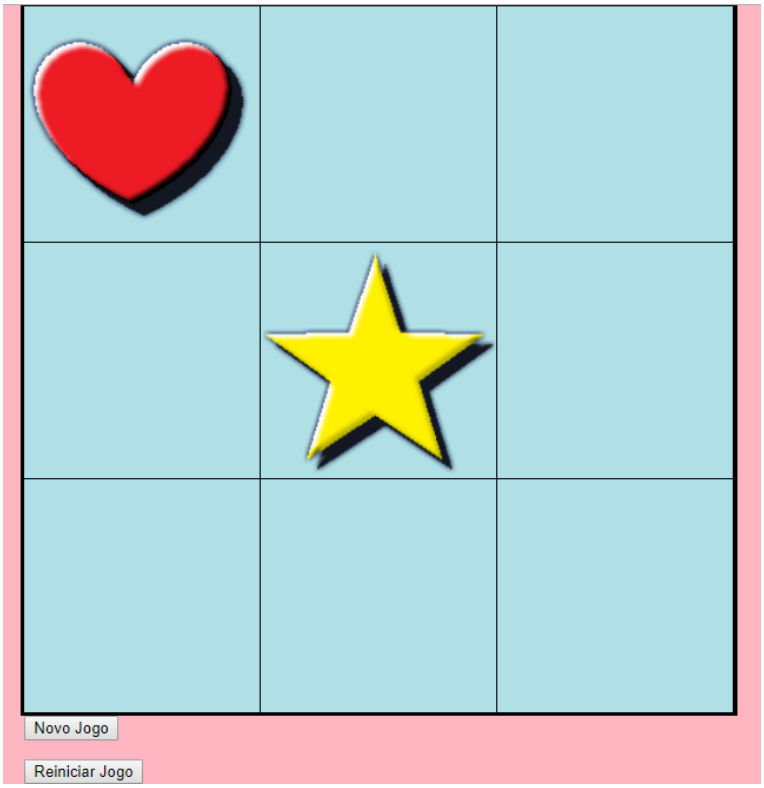
O Jogo da Velha criado em linguagem Javascript, HTML e css é composto por quatro telas. As 3 primeiras são referentes às opções iniciais de jogo *Escolher modo de jogo (implementamos apenas a opção jogador vs máquina), Escolher quem começa primeiro (o jogador ou o adversário), Escolher nível de dificuldade (3 níveis, fácil, médio e difícil).* A ultima tela é o jogo em da velha (tabuleiro com 9 casas), nesa.

escolha do tipo de jogo

escolha de quem começa a partida

O jogador 1 sempre utiliza a imagem do coração e o jogador 2 utiliza a estrela.

escolha da dificuldade



tela do jogo

3.1 Implementação do MINMAX

Para o MINMAX foi feita uma implementação que a cada nova jogada que o computador deve fazer ele procura a chance máxima de vitoria. Para tanto, ele percorre todo o array de elementos representando cada casa do tabuleiro e para casa posição verifica a chance de vitoria. Suas chances de vitoria são calculadas pela diferença entre o número de alinhamentos que ele pode formar versus o número de alinhamentos que o adversário pode formar. Veja o código:

```
1  function MinMax(tabuleiro,computador){
2
3  var casa = tabuleiro.split(',');
4
5  //esse vetor ser  a resultante de chances de alinhamento do
       computador - alinhamento do inimigo
6  var chancesPC = new Array(0,0,0,0,0,0,0,0,0);
7
8  //calcular chances de alinhamento do pc para posi  o i
9  var valor=(computador==1? 1:-1); //valor do computador
10
11 for(var i = 0; i<9 ; i ++){
12     if(casa[i]=="0"){//somente devemos calcular as chances para
           casas dispon veis (vazias)
13         casa[i]=valor;
14         chancesPC[i]=calcChances(casa.join(',') ,valor) -
               calcChances(casa.join(',') ,(-1)*valor);
15         casa[i]="0";//liberar a casa;
16     }
17     else{
18         chancesPC[i]=-10 ;//isso garante que casas ocupadas n o
               ser o selecionadas
19     }
20 }
21
22
23 //verificar a posi  o com melhor chance de alinhamento
24 var maior=0; //melhor representa a posi  o
25
26 //buscar a maior chance de vitoria
27 for(var i = 1; i<9 ; i++){
28     if( chancesPC[i]>chancesPC[maior] ){
29         maior=i;
30     }
31 }
```

```
32
33      return maior;
34  }
```

Para testar as chances de alinhamento, foi feita essa função que opera de modo a preencher o array de casas do tabuleiro com um valor, que pode ser 1 ou -1, dependendo se é desejado calcular as chances do computador ou do adversário, bem como quem é o primeiro e o segundo jogador.

Preenchida a matriz, é avaliada a chance de alinhamento nas linhas, depois nas colunas e por fim nas diagonais. Cada chance de alinhamento é contabilizada e retornada para a função que a chamou (no caso a função com o algoritmo do MINMAX).

```
1       function calcChances(casa, valor){
2       var cas = casa.split(',');
3       for(var i=0;i<9;i++){
4           if(cas[i]==0){
5               cas[i]=valor;
6           }
7       }
8
9       var chances=0;
10      var soma;
11      //chances de alinhamentos nas linhas
12      for(var i=0;i<3;i++){
13          soma=0;
14          for(var j=0;j<3;j++){
15              soma=soma+parseInt(cas[3*i+j]);
16          }
17          if(soma == 3*valor){
18              chances++;
19          }
20
21      }
22      //chances de alinhamentos nas colunas
23      for(var j=0;j<3;j++){
24          soma=0;
25          for(var i=0;i<3;i++){
26              soma=soma+parseInt(cas[3*i+j]);
27          }
28          if(soma== 3*valor){
29              chances++;
30          }
31      }
32      //chance de alinhamento na diag. princ
```

```
33      soma=0;
34      for(var i=0;i<3;i++){
35          soma=soma+parseInt(cas[3*i+i]);
36      }
37      if(soma== 3*valor){
38          chances++;
39      }
40
41      //chance de alinhamento na diag. sec
42      soma=0;
43      for(var i=0;i<3;i++){
44          soma=soma+parseInt(cas[2*i+2]); //3*(i+1)- 1 -i == 2*i+2
45      }
46      if(soma== 3*valor){
47          chances++;
48      }
49
50      //retorna o numero de alinhamentos possiveis
51      return chances;
52
53  }
```

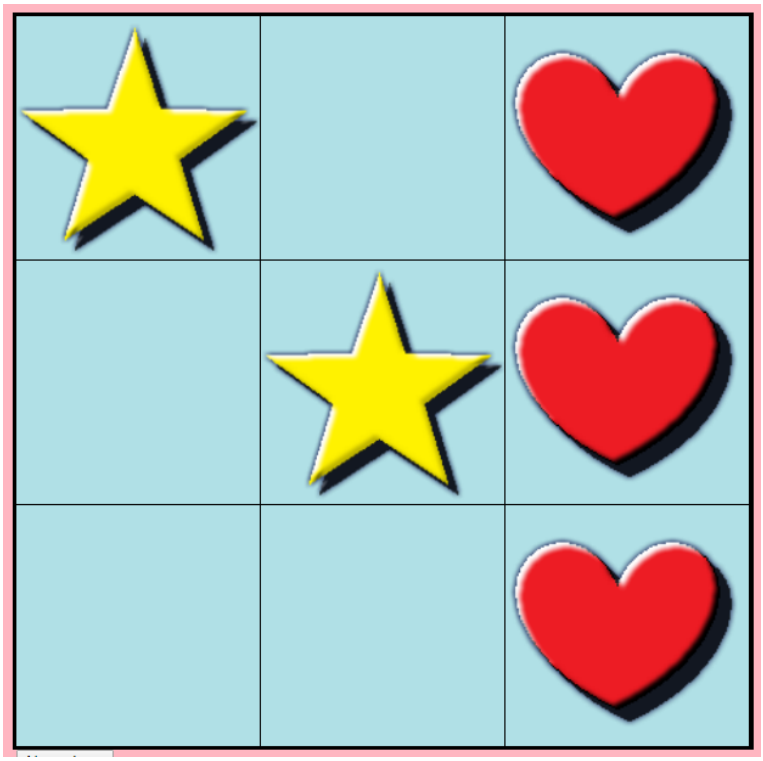
O algoritmo do MINMAX faz a decisão pela sua maior chance de vitória.

Além disso, o jogo foi feito de modo a ter 3 níveis de dificuldade: fácil, médio, difícil.

3.1.1 Nível Fácil

O nível fácil consiste no MINMAX puro. No entanto, é um algoritmo um tanto "burro", levando em consideração que trabalhamos apenas com 1 nível de dificudalde, portanto não é imbatível. Veja:

```
jogadaMinMax = MinMax(tab.join(','),computador);
```



vitoria do jogador no nível fácil

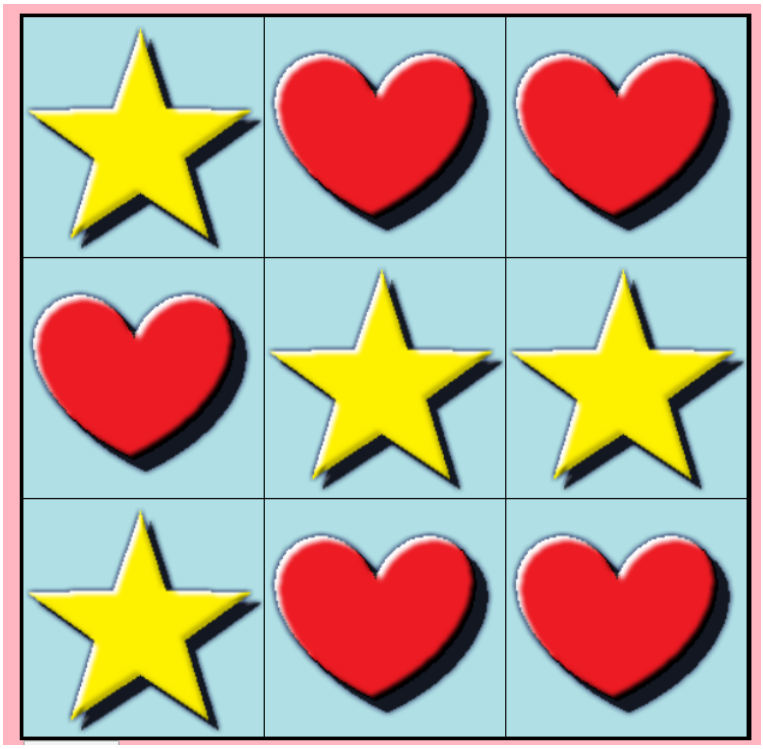
Nesse caso o jogador que escolheu iniciar, portanto era o coração que venceu a IA.

3.1.2 Nível Médio

Para o nível médio foi acrescentada 2 hard rules. Sempre que o computador pudesse vencer, ele faria a jogada para tanto. Caso não fosse possível, ele checa a possibilidade de vitoria do oponente e caso ele possa vencer, o computador joga de modo a bloquear a vitória do oponente.

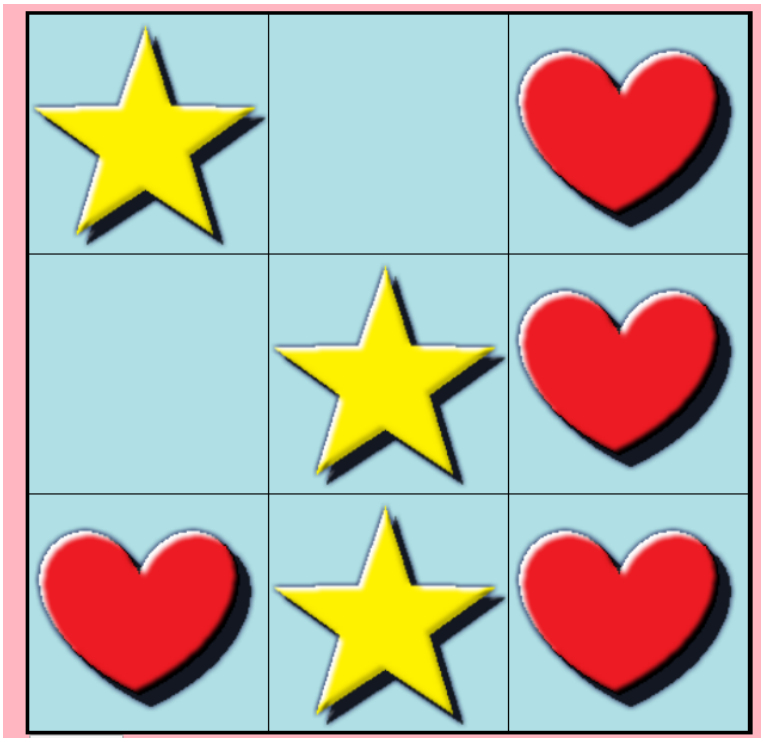
```
var valor=(computador==1? 1:-1);
//verificar se a IA tem possibilidade de vitoria
pos=hardRules(tab.join(','),valor);

//verificar se o advers rio tem possibilidade de vitoria
if (pos==-1){
pos=hardRules(tab.join(','), (-1)*valor);
}
```

Veja para o mesmo caso mostrado no nível fácil:

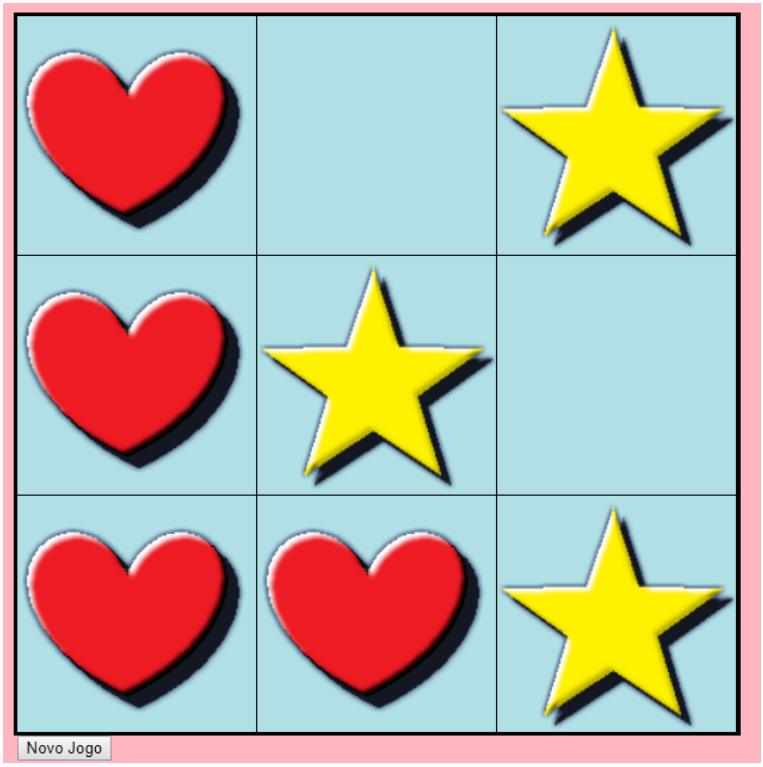


computador-estrela, jogador-coração

No entanto, ainda há 2 situações que o computador perde. São essas:



computador-estrela, jogador-coração



computador-estrela, jogador-coração

3.1.3 Nível Difícil

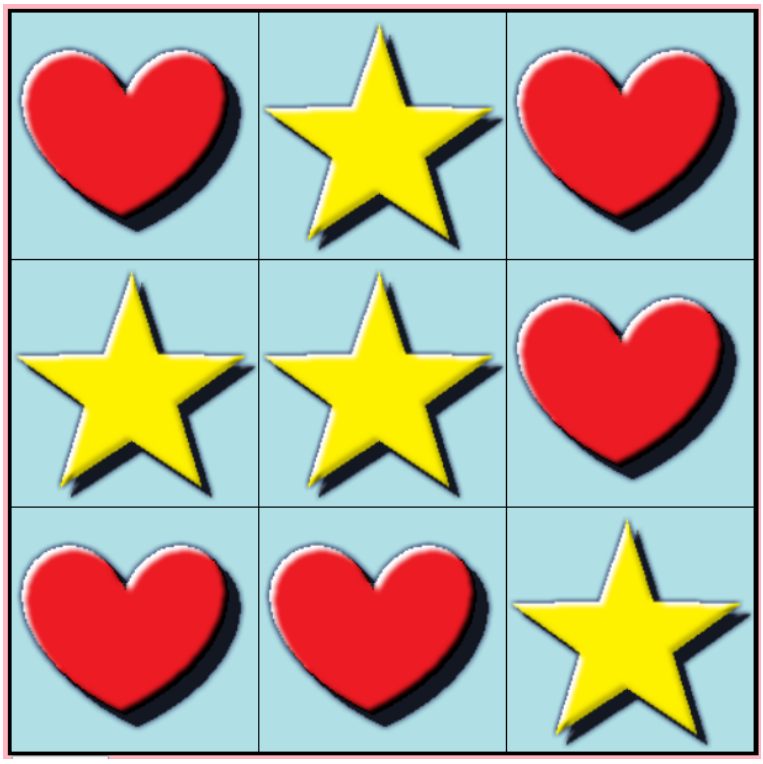
Levando em consideração os defeitos dos níveis anteriores. Foi adicionada mais 1 hard rule para evitar "ciladas". As situação identificadas no nível médio foram tradas pelas condições:

```
    if(pos==-1 && jogadas==3 && diff==3){
   //verificar caso especial --> cilada

                    /*
                       |     |  X
                    ___|____|____
                       |  O  |
                    ___|____|____
                     X |     |
                       |     |


                    */

    if(tab[4]==valor){
        if((tab[0]==(-1)*valor && tab[8]==(-1)*valor) || (tab[2]==(-1)
           *valor && tab[6]==(-1)*valor) ){
            var aleatoria= Math.floor(Math.random() *4);
            pos= 2*aleatoria+1

            }

        }
    }

    if(pos==-1 && jogadas==3 && diff==3){
    //verificar caso especial --> cilada 2

                    /*
                     X |     |
                    ___|____|____
                       |  O  |
                    ___|____|____
                       |  X  |
                       |     |
```

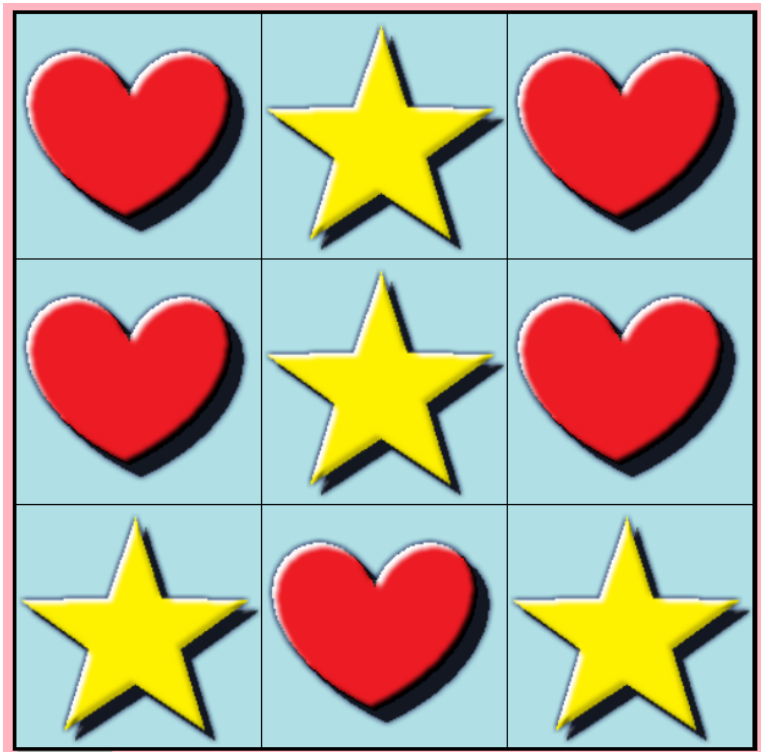
```
39
40
41                  O |     | X
42                 ___|_____|____
43                    |  O  |
44                 ___|_____|____
45                    |  X  |
46                    |     |
47
48
49                    |  X  |
50                 ___|_____|____
51                    |  O  |
52                 ___|_____|____
53                  X |     |  O
54                    |     |
55
56
57                 */
58
59          if(tab[4]==valor){
60              if(tab[1]==(-1)*valor && ( tab[6]==(-1)*valor|| tab
                    [8]==(-1)*valor)){
61
62                  var aleatoria= Math.floor(Math.random() *2);
63                  pos= 2*aleatoria; //jogar em 0 ou 2
64
65              }
66
67              if(tab[3]==(-1)*valor && ( tab[2]==(-1)*valor|| tab
                    [8]==(-1)*valor)){
68
69                  var aleatoria= Math.floor(Math.random() *2);
70                  pos= 6*aleatoria; //jogar em 0 ou 6
71
72              }
73
74              if(tab[5]==(-1)*valor && ( tab[0]==(-1)*valor|| tab
                    [6]==(-1)*valor)){
75
76                  var aleatoria= Math.floor(Math.random() *2);
77                  pos= 6*aleatoria +2; //jogar em 2 ou 8
78
```

```
79              }
80
81              if(tab[7]==(-1)*valor && ( tab[0]==(-1)*valor|| tab
                    [2]==(-1)*valor)){
82
83                  var aleatoria= Math.floor(Math.random() *2);
84                  pos= 2*aleatoria + 6; //jogar em 6 ou 8
85
86              }
87
88          }
```

Veja para o mesmo caso mostrado no nível médio:



computador-estrela, jogador-coração



computador-estrela, jogador-coração

4 Conclusão

Minimax é um algoritmo considerado como conhecimento básico dentro das competências de um Engenheiro de Inteligência Artificial e pode ser usado para o desenvolvimento de complexas aplicações, principalmente no âmbito de jogos, valendo-se do seu poderoso potencial de decisão.

Em virtude disso, esperamos que a apresentação do jogo da velha baseado no MINMAX com a utilização de hard rules tenha estimulado o aprendizado das técnicas de busca em árvores.

5 Referências

https://www.organicadigital.com/seeds/algoritmo-minimax-introducao-a-inteligencia-artificial/
Apostila desenvolvida por Fábio Meneghetti Ugulino de Araújo : Controle_Inteligente_-_Apostila_-_2015.pdf