



UNIVERSIDADE DE ÉVORA

Relatório Inteligência Artificial

Rúben Peixoto e Vanessa Santos  
37514 e 34191

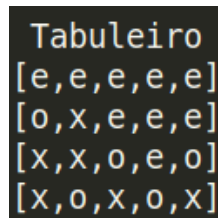
26 de Abril 2020

# 1 Terceiro Trabalho

Este projecto usa predicados que só funcionam no SWI Prolog.

## 1.1 Estrutura de Dados

Uma estrutura de dados que o grupo usou para representar os estados do jogo foi uma lista de listas, tal como se pode ver na figura a baixo.

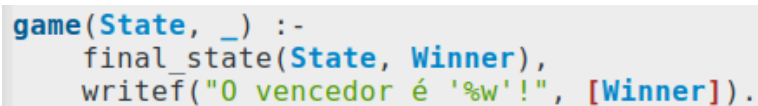


```
Tabuleiro
[e,e,e,e,e]
[o,x,e,e,e]
[x,x,o,e,o]
[x,o,x,o,x]
```

Figure 1: Pessoa = o Computador = x Vazio = e

## 1.2 Predicado Terminal

O grupo em vez de definir o predicado "terminal(Estado)", definimos um predicado parecido a "terminal(Estado, Jogador)". Escolhemos esta opção porque há situações no nosso código onde o uso deste ultimo predicado é mais vantajoso. Por exemplo, na figura a baixo, é mostrado o predicado "game", mais especificamente, o caso base.



```
game(State, _) :-
    final_state(State, Winner),
    writef("O vencedor é '%w'!", [Winner]).
```

Figure 2: Caso base do predicado game.

O predicado "game" é responsável por executar o jogo entre Pessoa VS Computador. No caso base, o grupo pretende que o predicado calcule se já se chegou a um estado final. Se sim, é retornado o jogador que venceu ("x" ou "o") ou se foi empate ("x-o") e esse resultado é imprimido no terminal através de uma mensagem a informar quem foi o vencedor.

## 1.3 Valor de Utilidade

Como o grupo decidiu dedicou a peça "o" para o computador, o valor da utilidade será 1 quando a peça for "o", será 0 quando for empate "x-o" e será -1 quando for a peça "x".

## 1.4 Minimax

Para executar o minimax, é necessário usar os ficheiros `minimax.pl` e `problema.pl`. Quando aceder ao top level do SWI Prolog selecciona-se o predicado `"play."` para começar.

```
?- play.  
A ação que irá ser tomada é 'insert_col3'.  
true.  
  
?- halt.  
  
real    2m30,587s  
user    2m24,750s  
sys     0m0,004s
```

Figure 3: Tempo que o algoritmo minimax leva para encontrar a próxima jogada

Quando executado o algoritmo minimax para o estado que está representado na Figura 1, o programa demorou aproximadamente dois minuto e meio.

## 1.5 Alpha Beta Search

Para executar o alpha beta, é necessário usar os ficheiros `alpha_beta_search.pl` e `problema.pl`. Quando aceder ao top level do SWI Prolog selecciona-se o predicado `"test_time."` para começar.

```
A ação que irá ser tomada é 'insert_col3'.  
  
real    0m0,493s  
user    0m0,197s  
sys     0m0,009s
```

Figure 4: Tempo que o algoritmo alpha beta leva para encontrar a próxima jogada

Quando executado o algoritmo alpha beta para o estado que está representado na Figura 1, o programa demorou aproximadamente 5 micro-segundos, o que é uma enorme diferença.

## 1.6 Três em Linha

Nesta fase o grupo preferiu utilizar o algoritmo alpha beta search devido à sua performance face ao algoritmo minimax. Mesmo assim, o algoritmo escolhido é muito lento quando tem que realizar as primeiras jogadas, por isso,

testamos o jogo com algumas peças no tabuleiro. De todos os testes que fizemos vamos destacar dois. Destacamos que a posição das peças e a quantidade foi igual nos dois teste.

No primeiro teste colocamos a peça "x" no lugar menos eficiente. Com isto o computador aproveitou-se na nossa falha e ganha o jogo, vide Figure 5.

No segundo teste, Figure 6, colocamos a peça "x" na coluna 1 não dando qualquer vantagem ao computador, por isso, ganhamos o jogo.

```

Tabuleiro
[e,e,e,e,e]
[e,e,e,e,e]
[e,e,e,e,e]
[o,x,o,x,e]

Jogadas:
  0 -> Inserir 'x' na coluna 0 (Mais à esquerda)
  1 -> Inserir 'x' na coluna 1
  2 -> Inserir 'x' na coluna 2
  3 -> Inserir 'x' na coluna 3
  4 -> Inserir 'x' na coluna 4 (Mais à direita)

|: 4.

Tabuleiro
[e,e,e,e,e]
[e,e,e,e,e]
[e,e,o,e,e]
[o,x,o,x,x]

Jogadas:
  0 -> Inserir 'x' na coluna 0 (Mais à esquerda)
  1 -> Inserir 'x' na coluna 1
  2 -> Inserir 'x' na coluna 2
  3 -> Inserir 'x' na coluna 3
  4 -> Inserir 'x' na coluna 4 (Mais à direita)

|: 1.
0 vencedor é 'o'!
Tabuleiro
[e,e,e,e,e]
[e,e,o,e,e]
[e,x,o,e,e]
[o,x,o,x,x]

```

Figure 5: Jogamos na coluna 4, e demos vantagem ao Computador

```

Tabuleiro
[e,e,e,e,e]
[e,e,e,e,e]
[e,e,e,e,e]
[e,e,e,e,e]
[o,x,o,x,e]

Jogadas:
  0 -> Inserir 'x' na coluna 0 (Mais à esquerda)
  1 -> Inserir 'x' na coluna 1
  2 -> Inserir 'x' na coluna 2
  3 -> Inserir 'x' na coluna 3
  4 -> Inserir 'x' na coluna 4 (Mais à direita)

|: 1.

Tabuleiro
[e,e,e,e,e]
[e,e,e,e,e]
[e,x,e,e,e]
[e,x,e,e,e]
[o,x,o,x,o]

Jogadas:
  0 -> Inserir 'x' na coluna 0 (Mais à esquerda)
  1 -> Inserir 'x' na coluna 1
  2 -> Inserir 'x' na coluna 2
  3 -> Inserir 'x' na coluna 3
  4 -> Inserir 'x' na coluna 4 (Mais à direita)

|: 1.
0 vencedor é 'x'!
Tabuleiro
[e,e,e,e,e]
[e,x,e,e,e]
[e,x,e,e,e]
[e,x,e,e,e]
[o,x,o,x,o]

```

Figure 6: Jogamos na coluna 1 e não demos qualquer vantagem ao Computador