Blockade

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo 2:
Maria João dos Santos Aguiar e Mira Paulo - up201403820
Nuno Miguel Mendes Ramos - up201405498

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

13 de Novembro de 2016

Resumo

Este trabalho consiste na construção de um Jogo chamado Blockade baseado em regras e problemas de decisão, utilizando uma linguagem de programação em lógica, denominada Prolog.

O jogo Blockade consiste num jogo de tabuleiro para dois jogadores, sendo que o objetivo principal baseia-se em conseguir mover o peão para a casa de partida do oponente, tentando fugir às paredes que este pode colocar para proteger o caminho ate à sua posição de partida.

Através da manipulação de predicados disponibilizados pelo SICStus Prolog, demonstramos neste relatório que além de ter sido possível a resolução do problema em específico, ainda foi possível a realização deste de uma forma eficiente e prática.

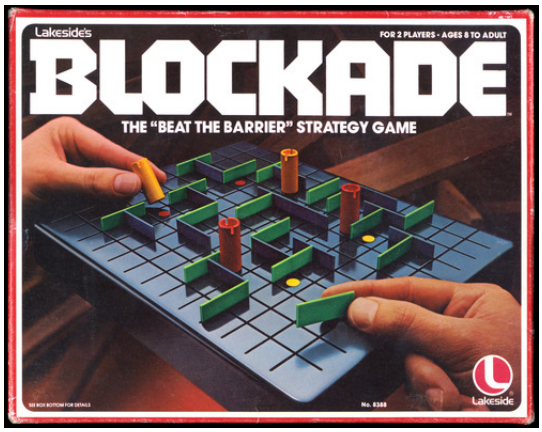
No inicio, a realização deste trabalho numa linguagem nova e diferente ao habitual pareceu algo bastante complicado mas com o passar do tempo conseguimos atingir os objetivos propostos. Podemos concluir que foi, através deste projeto, que conseguimos consolidar todos os conhecimentos aprendidos nas aulas práticas.

Conteúdo

1 Introdução

Este trabalho teve como objetivo uma melhor compreensão da linguagem Prolog e da utilização da recursividade de forma a melhorar a otimização do jogo. Neste relatório iremos apresentar a estrutura do jogo Blockade e os principais predicados usados para a sua criação, isto é tanto Lógica de jogo como Visualização e Manipulação do Tabuleiro.

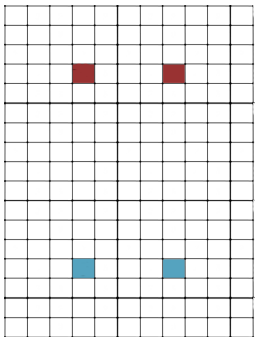
2 O Jogo Blockade

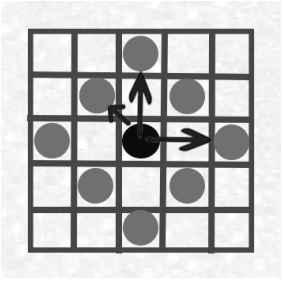


Blockade é um jogo de estratégia para dois jogadores inventado por Mirko Marchesi em 1975. O tabuleiro usado, com dimensões 11 x 14 contém 4 pontos, um em cada canto 4 x 4, onde se encontra cada peão no inicio do jogo.

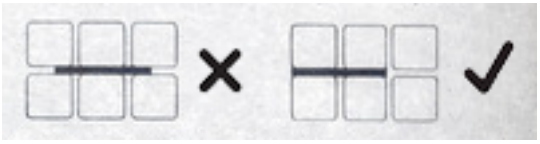
Cada jogador contém dois peões, nove paredes verdes, podendo estas ser colocadas no tabuleiro apenas verticalmente e nove paredes azuis, podendo estas ser colocadas unicamente na horizontal.

Assim, os quatro peões devem ser colocados nos pontos do tabuleiro de acordo com a sua cor. A imagem seguinte representa o estado inicial do tabuleiro.





A cada jogada, o jogador pode alterar a posição do seu peão uma ou duas casas para cima, uma ou duas casas para baixo, uma ou duas casas para a direita ou esquerda ou uma casa na diagonal.



Além disso pode ainda colocar uma parede no sentido de dificultar o jogo do adversário. No caso do jogador ficar sem paredes para colocar, só poderá mover o seu peão assim que for a sua vez de jogar.

O primeiro jogador a conseguir chegar com o seu peão ao ponto inicial do seu adversário ganha.

3 Lógica do Jogo

3.1 Representação do Estado do Jogo

O tabuleiro de jogo tem dimensões 11 x 14. Para melhor visualização decidimos trocar o nome dos elementos da lista para diferentes nomes, com menor tamanho. De seguida encontram-se explicadas as relações entre esses elementos.

- **e** - Empty
- **nVw** - noVerticalWall
- **vW** - verticalWall
- **nW** - wall
- **w** - wall
- **n** - null
- **p11** - Player 1, Pawn 1
- **p12** - Player 1, Pawn 2
- **p21** - Player 2, Pawn 1
- **p22** - Player 2, Pawn 2
- **wP11** - Winner Player 1, Pawn 1
- **wP12** - Winner Player 1, Pawn 2
- **wP21** - Winner Player 2, Pawn 1
- **wP22** - Winner Player 2, Pawn 2
- **sP1** - Start Player 1
- **sP2** - Start Player 2

3.2 Representação do estado inicial do tabuleiro

```
emptyBoard(Board):-
   Board =
   [[e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
   [nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
   [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
   [nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
   [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e]
   [nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
   [e,nVw,e,nVw,e,nVw,p11,nVw,e,nVw,e,nVw,e,nVw,p12,nVw,e,nVw,e,nVw,e],
   [nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
   [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
   [nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
   [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
   [nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
   [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
   [nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
   [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
   [nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
   [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
   [nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
   [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
   [nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
   [e,nVw,e,nVw,e,nVw,p21,nVw,e,nVw,e,nVw,e,nVw,p22,nVw,e,nVw,e,nVw,e],
   [nW,n,nW,n,nW,n,nW,n,nW,n,nW,e,nW,n,nW,n,nW,n,nW,n,nW],
   [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
   [nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
   [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
   [nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
   [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e]].
```

Listing 1: Code example

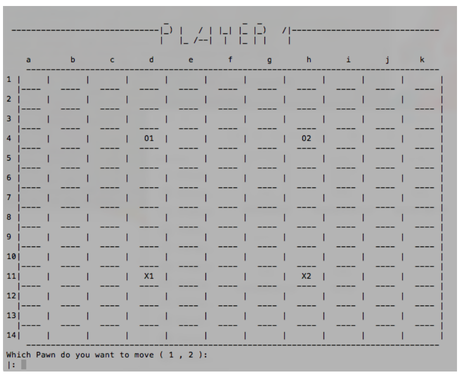
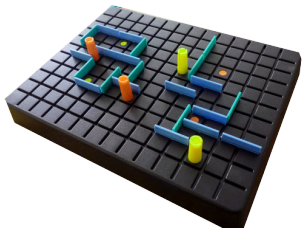


Figura 1: Representação do estado inicial do tabuleiro

3.3 Representação de um possível estado intermédio do tabuleiro



```
intermediateBoard(Board):-
Board=[[e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw
    ,e],
[nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
[e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
[nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
[e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
[nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
[e,nVw,e,nVw,e,nVw,sP1,nVw,e,nVw,e,nVw,e,nVw,sP1,nVw,e,nVw,e,nVw,e
    ],
[nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,w,n,w,n,nW,n,nW],
[e,nVw,e,nVw,e,vW,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
[nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,p12,nW,n,nW],
[e,nVw,e,nVw,e,vW,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
[nW,n,nW,n,nW,n,nW,n,w,n,w,n,nW,n,nW,n,nW,n,nW],
[e,nVw,p21,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,vW,e,nVw,e,nVw,e],
[nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
[e,nVw,e,nVw,e,nVw,p11,nVw,e,nVw,e,nVw,e,nVw,e,vW,e,nVw,e,nVw,e],
[nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
[e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,p22,nVw,e],
[nW,n,nW,n,nW,n,w,n,w,n,nW,n,nW,n,w,n,nW,n,w,n,nW],
[e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,vW,e,nVw,e,nVw,e,nVw,e,nVw,e],
[nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
[e,nVw,e,nVw,e,nVw,sP2,nVw,e,nVw,e,vW,e,nVw,sP2,nVw,e,nVw,e,nVw,e],
[nW,n,nW,n,nW,n,nW,n,nW,n,nW,e,nW,n,nW,n,nW,n,nW,n,nW],
[e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
[nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
[e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
[nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW,n,nW],
[e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e]].
```

Listing 2: Code example

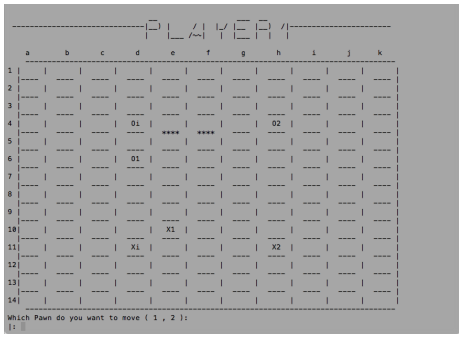


Figura 2: Representação de um possível estado intermédio do tabuleiro

3.4 Representação de um possível estado Final do tabuleiro

```
finalBoard(Board):-
    Board =[[e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,
        e,nVw,e],
     [nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW],
  [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,p11,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e
      ], [nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW],
  [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
 [nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW],
  [e,nVw,e,nVw,e,nVw,p21,vW,e,nVw,e,nVw,e,nVw,wp21,nVw,e,nVw,p22,
      nVw,e],[nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW
      ],
  [e,nVw,e,nVw,e,nVw,e,vW,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
    [nW,e,nW,e,w,e,w,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW],
  [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
 [nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW],
  [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
 [nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW],
  [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
[nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW],
  [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
[nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,w,e,w,e,nW],
 [e,nVw,e,nVw,e,nVw,sP2,nVw,e,nVw,e,nVw,e,nVw,sP2,nVw,e,nVw,e,nVw,e
     ], [nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW],
  [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],
[nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW],
  [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e],[
      nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW,e,nW],
 [e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e,nVw,e]].
```

Listing 3: Code example

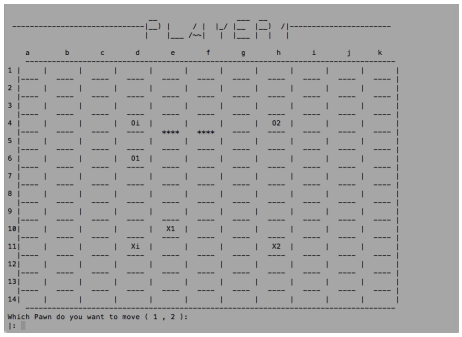


Figura 3: Representação de um possível estado final do tabuleiro

3.5 Visualização do Tabuleiro

De seguida, apresenta-se o código responsável por mostrar o tabuleiro na consola:

```
board_display([L1|LS]):-
 display_x_coord,
 nl, write('    ------------------------------------
 ------------------------------------------------  '),nl,
 display_board([L1|LS],1,1).

display_board([L1|LS],Y, X1):-
 display_y_coord(Y, X1, X2),
 write('|'),
 display_line(L1), nl,
 Y1 is Y+1,
 display_board(LS,Y1, X2).

display_board([], Y, X):-
   Y>=X,
   write('    ----------------------------------------
   ------------------------------------------- '),nl.

display_line([E1|ES]):-traduz(E1,V),write(V),
 display_line(ES).

display_line([]):-
 write(' |').

display_x_coord:-
 write('    a         b        c        d       e        f
 g        h        i        j       k ').

display_y_coord(Y, X1, X2):-X1<10,1 is Y mod 2,write(X1),
 write(' '),
 X2 is X1+1.

display_y_coord(Y, X1, X2):-1 is Y mod 2,write(X1),
 X2 is X1+1.

display_y_coord(Y, X1, X2):-
 0 is Y mod 2,
 X2 is X1,
 write('  ').
 traduz(empty, '    ').
 traduz(null, '    ').
 traduz(startPlayer1, ' Oi ').
 traduz(startPlayer2, ' Xi ').
 traduz(verticalwall, ' *  ').
 traduz(wall, '****').
 traduz(noWall, '----').
 traduz(noVerticalWall, ' |  ').
 traduz(player21,' X1 ').
 traduz(player22,' X2 ').
 traduz(player12,' O2 ').
 traduz(player11,' O1 ').
 traduz(winnerplayer2,' Xw ').
 traduz(winnerplayer1,' Ow ').
```

Listing 4: Code example

3.6 Lista de Jogadas Válidas

Em cada jogada, o utilizador deve escolher qual o peão com que quer jogar (1 ou 2) e a direção em que quer mover esse peão: direita, esquerda, cima, baixo, diagonal cima direita, diagonal cima esquerda, diagonal baixo direita ou diagonal baixo esquerda. De seguida essa posição é testada no sentido de garantir que se trata de um movimento válido.

```
readingInput(Pawn, Direction,NewPawn,NewDirection ):-
  readPawn(Pawn),
  validateInputPawn(Pawn, NewPawn),
  readDirection(Direction),
  validateInputDirection(Direction, NewDirection).

readPawn(Pawn):-
  write('Which Pawn do you want to move ( 1 , 2 ):'),nl,
  read(Pawn).

readDirection(Direction):-
  write('Which direction do you want to take ( left->(l1 or l2),
      right->(r1 or r2), top->(t1 or t2), bottom->(b1 or b2),
      diagonalTopRight->dtr , diagonalTopLeft->dtl ,
      diagonalBottomLeft->dbl ,   diagonalBottomRight->dbr)'),
  read(Direction).
```

Listing 5: Code example

Além disso pode ainda escolher se quer ou não colocar uma parede, as coordenadas onde a quer colocar e a direção (vertical ou horizontal). A direção e coordenadas são testadas para garantir que se trata de uma parede válida.

```
askingPosition(WallX,WallY):-
  write('Position x (a....k) :'),nl,  read(Answer1),
  validateInputPosition(Answer1,NewAnswer),
  letterCoordinateToNumber(NewAnswer,X),
  transformToRealCoordinates(X,WallX),
  write('Postion y (1....14) :'),nl,
  read(Answer2),nl,
  validateInputPosition2(Answer2),
  transformToRealCoordinates(Answer2,WallY).

wallOrientation(Orientation,NewOrientation):-
  write('Wall orientation (v or h) :'),nl,
  read(Orientation),
  validOrientation(Orientation,NewOrientation).

wallPositionInside(v,WallPositionInside):-
  write('You want your wall on left top, right top, left bottom ,
      right bottom (lt , rt, lb , rb): '),nl,
  read(Answer),
  validPositionInside(v,Answer,WallPositionInside).

wallPositionInside(h,WallPositionInside):-
  write('You want your wall on top to the left, top to the right,
      bottom to the left or bottom to the right (tl, tr, bl, br): '
      ),nl,
  read(Answer),
  validPositionInside(h,Answer,WallPositionInside).
```

Listing 6: Code example

3.7 Execução de Jogadas

A partir do momento em que o utilizador decide a sua jogada, segue-se a sua validação. A função **isAvalidMove** é responsável por retornar a validade da jogada. Para uma jogada ser válida é necessário que não haja nenhuma parede no caminho ate à posição escolhida, que as posições finais não ultrapassem as margens do tabuleiro e que não haja nenhum peão nessa mesma posição. Para isso foram criadas os predicados **hasNoWall**, **checkBorders** e **isApawnPosition**.

```
1 isAvalidMove([L1|LS],Xi,Yi,Xf,Yf,Direction, Xlimit, Ylimit):-
2   hasNoWall([L1|LS],Direction,Xi,Yi),
3   \+checkBorders(Xf,Yf, Xlimit, Ylimit),
4   \+isApawnPosition([L1|LS],Xf,Yf).
```

Listing 7: Code example

3.8 Avaliação do Tabuleiro

A avaliação e manipulação do tabuleiro é conseguida através dos Predicados:

- **getListElement** que recebe o Tabuleiro, uma posição (coordenada X e coordenada Y) e retorna o nome do elemento nela contido.

```
1  getListElement([L1|LS],Xelement,Yelement,X,Y,Element):-
2    checkTabLine(L1,Xelement,Yelement,X,Y,Element),!;
3    (Y1 is Y+1,
4    getListElement(LS,Xelement,Yelement,X,Y1,Element)).
5
6  getListElement([],_Xelement,_Yelement,_X,_Y,_Element).
7
8  checkTabLine([L1|LS],Xelement,Yelement,X,Y,Element):-
9    compareCoordinates(L1,Xelement,Yelement,X,Y,Element),!;
10   ( X1 is X+1,
11   checkTabLine(LS,Xelement,Yelement,X1,Y,Element)).
12
13 compareCoordinates(L1,Xelement,Yelement,X,Y,Element):-
14   Xelement=X,
15   Yelement=Y,
16   Element=L1.
```

Listing 8: Code example

- **returnPosition** que recebe o Tabuleiro e o nome do Elemento/Peão e retorna a posição onde este se encontra (coordenada X e coordenada Y).

```
1  returnPosition(_Name,[],_X, _Y, _Xf, _Yf):-
2    write('Dont found the element'),fail.
3
4  returnPosition(Name,[L1|LS],X, Y, Xf, Yf):-
5    checkLine(Name,L1,X,Y,Xf,Yf),
6    Yf == Y,!;
7    (Y1 is Y+1,
8    returnPosition(Name,LS,X,Y1,Xf,Yf)).
9
10 checkLine(Name,[L1|LS],X,Y,Xf,Yf):-
```

```
11    compareName(Name,L1,X,Y,Xf,Yf),!;
12    (X1 is X+1,
13    checkLine(Name,LS,X1,Y,Xf,Yf)).
14
15  checkLine(_Name,[],_X,_Y,_Xf,_Yf).
16
17  compareName(Name,T1,X,Y,Xf,Yf):-
18    Name=T1,
19    Xf is X,
20    Yf is Y.
```

Listing 9: Code example

- **setListElement** que atualiza o Tabuleiro com um novo elemento. Recebe o Tabuleiro, uma posição (coordenada X e coordenada Y) e o nome do Elemento/Peão e retorna um novo tabuleiro onde a posição (X,Y) contém esse mesmo elemento.

```
1   setListElement([],_Xelement,_Yelement,_X,_Y,_Element,[]).
2
3   setListElement([L1|LS],Xelement,Yelement,X,Y,Element,[N1|NS])
        :-
4     setTabLine(L1,Xelement,Yelement,X,Y,Element,N1),
5     Y1 is Y+1,
6     setListElement(LS,Xelement,Yelement,X,Y1,Element,NS).
7
8   setTabLine([],_Xelement,_Yelement,_X,_Y,_Element,[]).
9
10  setTabLine([L1|LS],Xelement,Yelement,X,Y,Element,[N1|NS]):-
11    (setElementValue(Xelement,Yelement,X,Y),
12    N1=Element;
13    N1=L1),
14    X1 is X+1,
15    setTabLine(LS,Xelement,Yelement,X1,Y,Element,NS).
16
17  setElementValue(Xelement,Yelement,X,Y):-
18    Xelement=X,
19    Yelement=Y.
```

Listing 10: Code example

3.9 Final do Jogo

A cada jogada realizada é necessário testar o estado do jogo, ou seja verificar se o jogo deve continuar ou se algum dos jogadores ganhou a partida e por esse motivo, o jogo deve terminar. Para isso existe o predicado denominado **isAwinner** que recebe o tabuleiro e a posição final escolhida pelo jogador e verifica se se trata de uma posição vencedora.

```
1  isAwinner([L1|LS],PlayerNumber,Xf,Yf):-
2    nl,nl,
3    getWinnerPosition(PlayerNumber,FinalPosition),
4    getListElement([L1|LS],Xf,Yf,1,1,NewElement),
5    NewElement==FinalPosition,nl.
6
7  getWinnerPosition(1,FinalPosition):-
8    FinalPosition=startPlayer2.
9
10 getWinnerPosition(2,FinalPosition):-
11   FinalPosition=startPlayer1.
```

Listing 11: Code example

3.10 Jogada do Computador

Caso o utilizador opte por escolher o Modo Computador-Jogador ou o modo Computador-Computador, no inicio do jogo pode decidir se quer ou não aumentar a dificuldade.

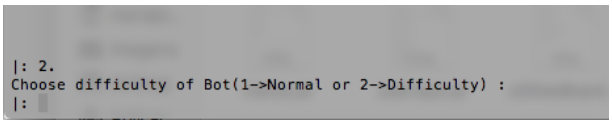


Figura 4: Escolha da dificuldade do *Bot*

No jogo com **dificuldade normal**, foram elaborados predicados que tratassem de retornar posições e movimentos dos peões de forma aleatória. Para isso criamos o predicado **randomPawn** que retorna um peão aleatório (1 ou 2) e o predicado **randomDirection** que retorna uma direção aleatória (l1,l2,r1,r2,b1,b2,t1,t2,dtr,dtl,dbr,dbl).

```
1  randomPawn(Pawn):-
2    random(0,2,GeneratePawn),
3    generatePawn(GeneratePawn,Pawn).
4
5  randomDirection(Direction):-
6    random(0,12,GenerateDirection),
7    generateDirection(GenerateDirection,Direction).
```

Listing 12: Code example

A colocação de paredes também é feita de forma aleatória. Para isso existe o predicado **randomWallAnswer** que decide se deve ou não existir uma nova parede, retornando sim ("y") ou não ("n"). Caso a resposta seja sim, ou seja, de colocação de uma parede é chamado o predicado **randomOrientation** que

gera uma orientação aleatória (vertical ou horizontal). Por fim, é chamado o predicado **randomWallPosition** que retorna uma posição aleatória (valor X e Y de 1 a 14) e o predicado **randomPositionInside** que gera uma posição aleatória da parede em relação à posição escolhida.

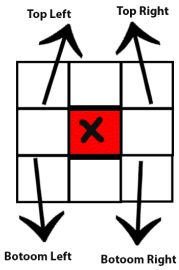


Figura 5: Possíveis posições de parede horizontal *Bot*

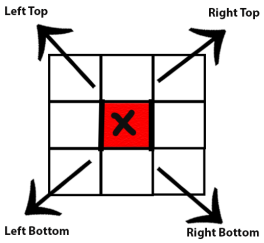


Figura 6: Possíveis posições de parede vertical *Bot*

```
randomWallAnswer(Answer):-
  random(0,2,NumberGenerated),
  generateWallAnswer(NumberGenerated,Answer).

randomOrientation(Orientation):-
  random(0,2,NumberGenerated),
  generateOrientation(NumberGenerated,Orientation).

randomPositionInside(Orientation,WallPositionInside):
  random(0,4,NumberGenerated),generatePositionInside(Orientation,
      NumberGenerated,WallPositionInside).

randomWallPosition(Xlimit,Ylimit,WallX,WallY):-
  MaxX is (Xlimit+1)/2+1,
  random(1,MaxX,X),
  X2 is round(X),
  transformToRealCoordinates(X2,WallX),
  MaxY is (Ylimit+1)/2+1,
  random(1,MaxY,Y),
  Y2 is round(Y),
  transformToRealCoordinates(Y2,WallY).
```

Listing 13: Code example

Relativamente ao modo de jogo com **dificuldade elevada**, foram sentidas algumas dificuldades na execução deste ponto, ainda assim foi conseguido melhorar o modo de jogo com sucesso. As alterações relativamente ao modo normal, é a escolha do Computador na direção que vai tomar. Esta escolha é mais inteligente. Depois de ter sido gerado um peão aleatório, é chamado o predicado **preparingForRandomDirection** onde se calcula as distâncias à casa de partida do outro jogador, usando o predicado **distanceTo**. É escolhida a casa de partida que estiver mais perto do peão. Depois de feita a escolha da casa de partida passa-se para decidir a direção. É nesta fase que há uma grande diferença. O programa compara as coordenadas do peão e da casa de partida do outro jogador. Desta comparação pode tirar 4 conclusões: a casa de partida esta em cima do lado esquerdo, em cima do lado direito, em baixo do lado esquerdo ou em baixo do lado direto, relativamente ao peão. Nós chamamos a estas posições quadrantes e escolhemos o quadrante a ser usado com o predicado **choosingQuadrant**.Dentro de cada quadrante existe 5 direções possíveis. Por exemplo, se a casa de partida do oponente estiver em relação ao peão em cima do lado direito, o peão pode ir para a direita, 1 ou 2 casas, ir para a cima, 1 ou 2 casas ou ir na diagonal para cima para a direita.(Quadrante 1). Depois de escolhido o quadrante utiliza-se o predicado **randomDirectionNewDifficulty** para gerar uma das 5 possíveis direções a ser tomada pelo peão.

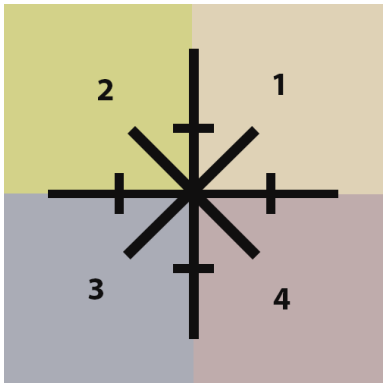


Figura 7: Possíveis posições de parede vertical *Bot*

```
1  preparingForRandomDirection([L1|LS],Player,Pawn,Direction):-
2  %(player,pawn)
3    input(Player,Pawn,PawnName),
4    returnPosition(PawnName,[L1|LS],1,1,X,Y),
5    distanceTo(X,Y,7,21,Distance1),
6    distanceTo(X,Y,15,21,Distance2),
7    Distance1>=Distance2->
8      (
9        input(Player,Pawn,PawnName),
10       returnPosition(PawnName,[L1|LS],1,1,X,Y),
11       choosingQuadrant(X,Y,15,21,Quadrant)
12     );
13     (
14       input(Player,Pawn,PawnName),
15       returnPosition(PawnName,[L1|LS],1,1,X,Y),
16       choosingQuadrant(X,Y,7,21,Quadrant)
17     ),
18     randomDirectionNewDifficulty(Quadrant,Direction).
```

```
19
20 choosingQuadrant(Xp,Yp,Xf,Yf,Quadrant):-
21   Xp > Xf->
22     (
23       Yp >= Yf->
24         Quadrant=2;
25         Quadrant=3
26     );
27     (
28       Yp >= Yf->
29         Quadrant=1;
30         Quadrant=4
31     ).
32
33 distanceTo(X1,Y1,X2,Y2,Distance):-
34   Xdist is X1-X2,
35   Ydist is Y1-Y2,
36   Distance is sqrt(Xdist**2+Ydist**2).
```

Listing 14: Code example

4 Interface com o Utilizador

A interface da linha de comandos foi feita de forma a proporcionar uma experiência agradável e simples ao utilizador.

Figura 8: Menu Inicial do Jogo

O menu principal do jogo permite ao utilizador Jogar (1. 2. 3. seguido de Enter), aceder às instruções do Jogo (4. seguido de Enter) ou sair (5. seguido de Enter).

O jogador pode escolher um dos três modos de jogo : Modo Jogador Jogador, Modo Jogador Computador ou Modo Computador Computador.

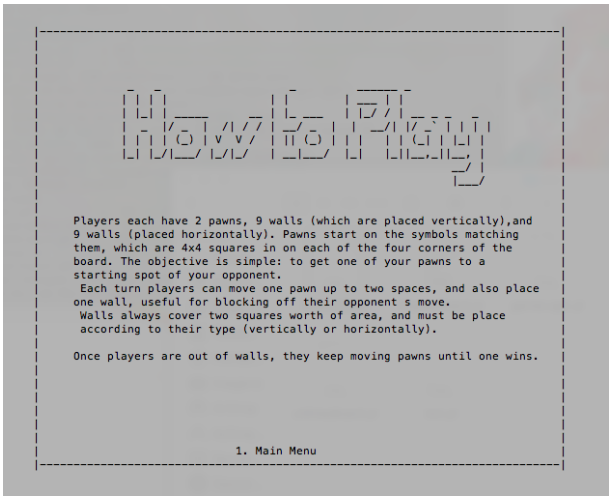


Figura 9: Instruções de Jogo

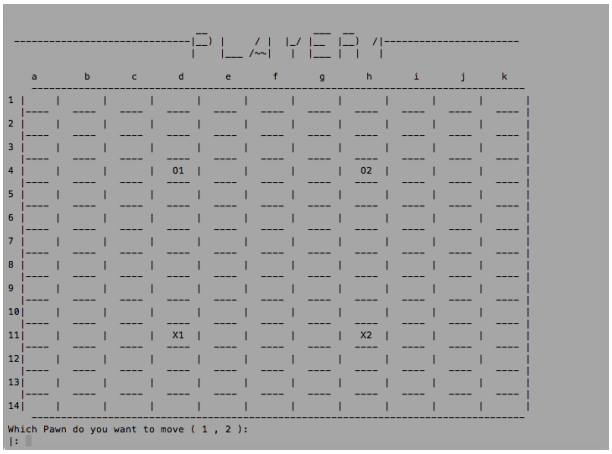


Figura 10: Estado Inicial Tabuleiro

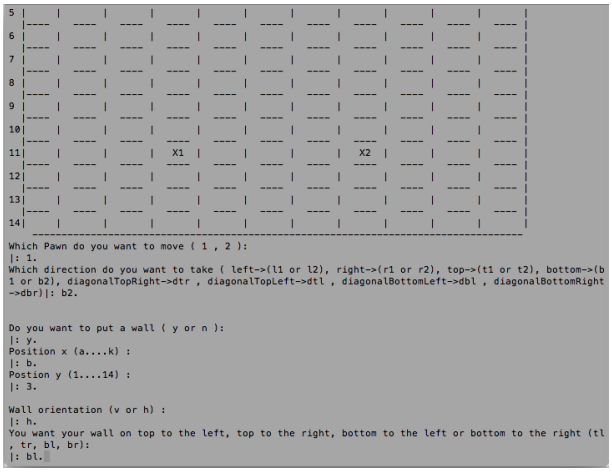


Figura 11: Exemplo de Jogada

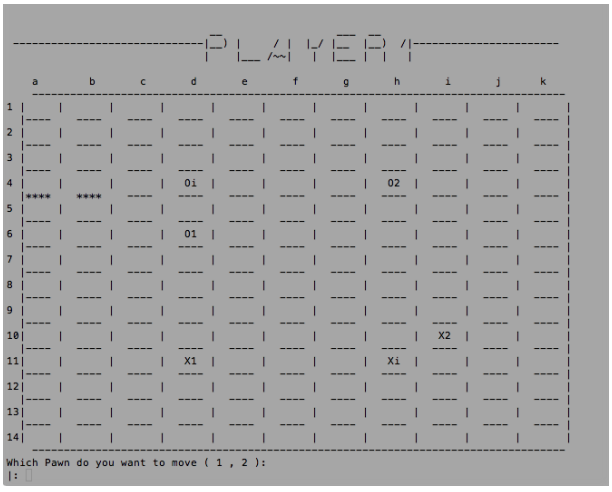


Figura 12: Jogada Player 1 (Modo Jogador - Jogador)

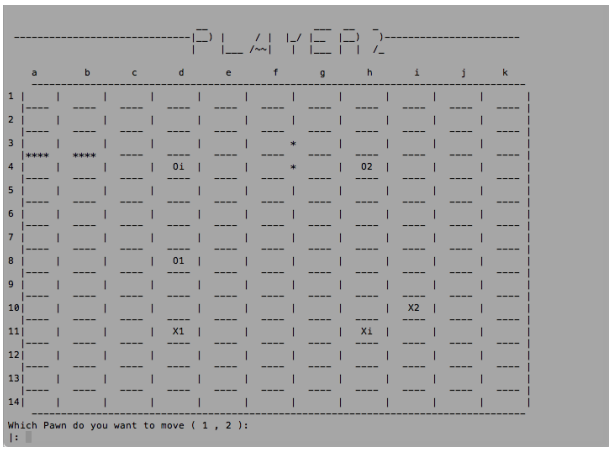


Figura 13: Jogada Player 2 (Modo Jogador - Jogador)

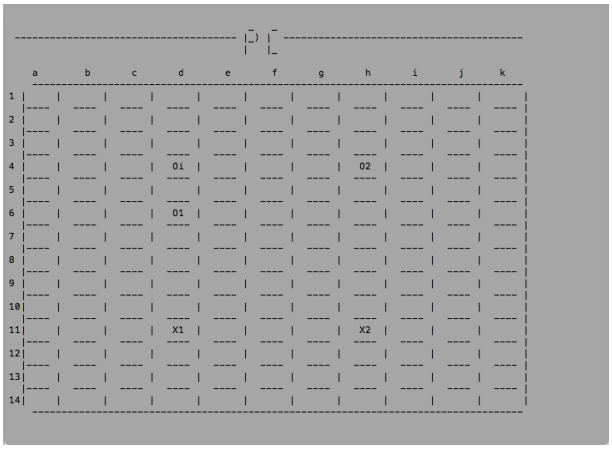


Figura 14: Jogada PC (Modo Jogador - Computador)

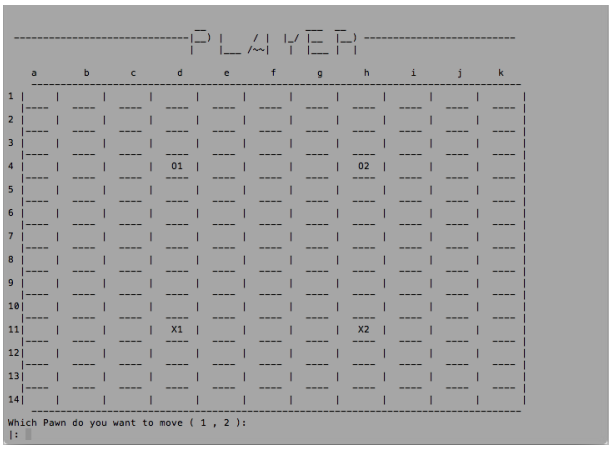


Figura 15: Jogada Player (Modo Jogador - Computador)

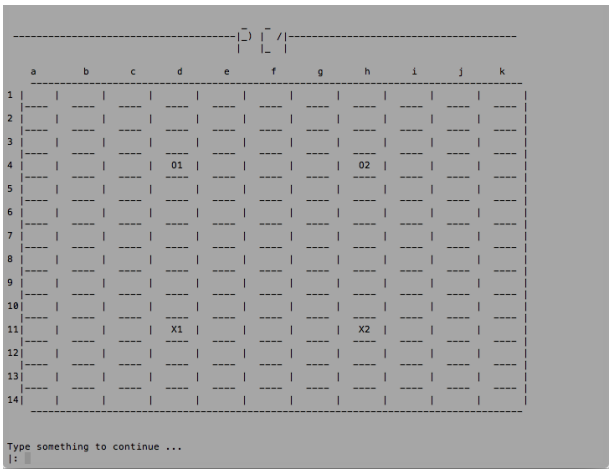


Figura 16: Jogada PC1 (Modo Computador - Computador)

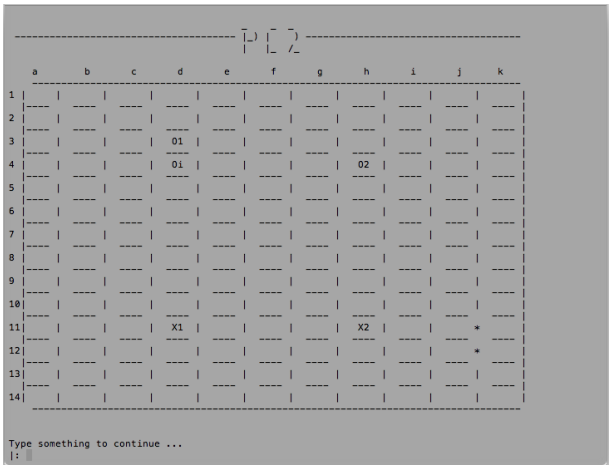


Figura 17: Jogada PC2 (Modo Computador - Computador)

5 Conclusões

A principal conclusão a retirar deste segundo projeto de Programação em Lógica é que, a linguagem Prolog é bastante poderosa para a resolução de uma enorme variedade de questões de otimização.

Apesar das dificuldades, a solução implementada pelo grupo correspondeu às expectativas e foi de encontro ao que foi pedido.

No futuro, seria possível melhorar a forma como implementamos o modo de jogo Computador-Computador e Computador-Jogador, com a implementação do tabuleiro através de um grafo, tornando assim o este modo mais inteligente.

Em suma, o desenvolvimento do projeto com o apoio da linguagem Prolog foi uma forma de perceber, na prática, o pensamento lógico e recursivo que esta linguagem requer na construção de cada predicado.

6 Bibliografia

- **Documentação** - Documentos fornecidos na página do Moodle da Unidade Curricular.
- **Páginas Web** - http://www.swi-prolog.org/

A Anexos

O código fonte do projeto encontra-se na pasta *src* anexada junto com este relatório.