Campo Bello

Relatório Intercalar

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Campo Bello 1:
Francisca Cerquinho - up201505791
Mariana Silva - up201506197

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

13 de Novembro de 2017

Resumo

Este projeto consiste na implementação de um jogo de tabuleiro, o *Campo Bello*, através da linguagem de programação em lógica, Prolog, que é baseada num paradigma declarativo assente em três conceitos fundamentais: a unificação, a recursividade e o *backtracking*.

O principal objetivo é que cada jogador remova o maior número possível das suas próprias peças, até que não possua mais movimentos válidos ou mais peças.

O problema proposto foi na sua totalidade implementado, através da utilização dos predicados disponibilizados pelo *SICStus Prolog*, onde foi tido em conta não só a funcionalidade do próprio jogo, como também a eficiência do próprio código. Assim, foi possível a consolidação dos conceitos lecionados ao longo da unidade curricular.

Conteúdo

1 Introdução

No âmbito da unidade curricular Programação em Lógica foi-nos proposto a realização de um jogo de tabuleiro com base na linguagem de programação *Prolog*, pondo à prova os nossos conhecimentos relativamente a regras e problemas intrínsecos à linguagem.

Efetivamente, ao longo deste relatório irão ser abordados três grandes tópicos, nomeadamente, a história e regras do próprio jogo, a descrição dos principais predicados utilizados para a sua lógica, manipulação e visualização do tabuleiro e, por fim, a interface com o utilizador. Teremos em conta, também, no final, as principais conclusões deste projeto e possíveis melhorias.

Desta forma, procuramos responder ao que nos é exigido, de forma sucinta e explícita, sendo nossa intenção fazer com que o presente relatório sirva de guia e suporte para os interessados no jogo.

2 O Jogo Campo Bello

2.1 História

O *Campo Bello* foi pensado no início de 2014 por John Caddell e baseado no jogo *Cracker Barrel Peg*, adaptando-o para múltiplos jogadores. Depois de vários protótipos e dezenas de horas de teste, este jogo demorou mais de dois anos a ser implementado.

2.2 Objetivo e regras do jogo

Neste jogo, cada jogador tenta remover o maior número possível das suas peças do tabuleiro. Na sua vez, o jogador deve saltar com a sua peça para outra e se a peça que saltou for uma das suas, deve retirá-la do jogo. Caso contrário, se for uma das adversárias, então pode remover qualquer uma das suas peças do tabuleiro (incluindo a usada no salto).

Cada jogador pode "encadear"até 3 saltos com a mesma peça durante a sua vez, mas não pode saltar sobre a mesma duas vezes. A peça com que o jogador salta não pode ocupar o mesmo espaço durante a mesma jogada. Caso não consiga dar um salto, o jogador pode ignorar a sua vez. O jogo continua até que um jogador não tenha mais peças no tabuleiro ou nenhum jogador consiga fazer um salto válido.

No final do jogo, cada jogador obtém 1 ponto para cada uma das suas peças fora da sua área de partida e 3 pontos para cada peça que está na sua área de partida.

O jogador com o menor número de pontos ganha.

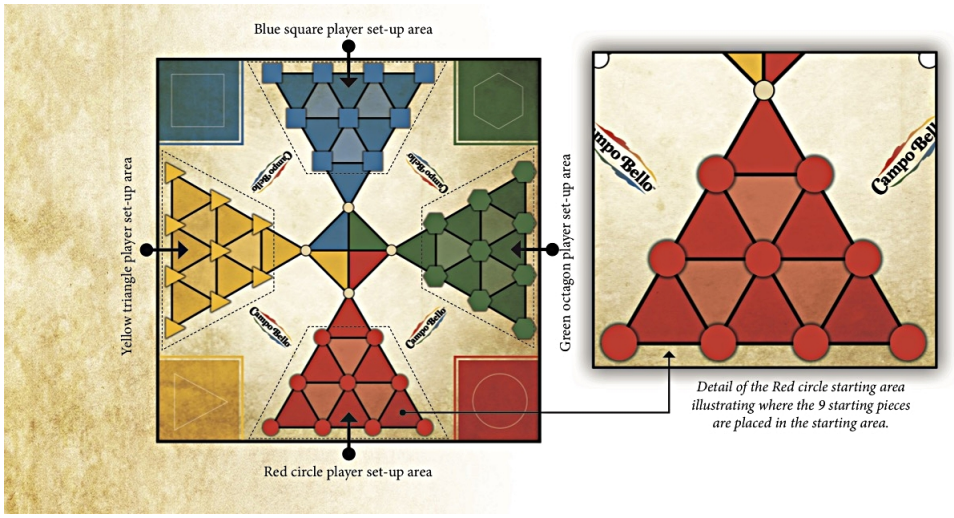

Figura 1: Aspeto do tabuleiro

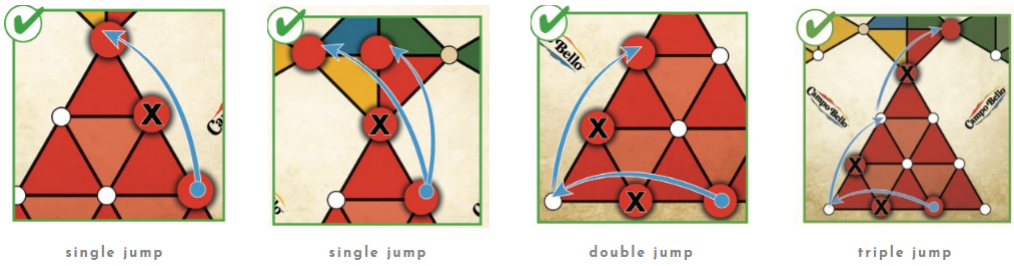

Figura 2: Movimentos permitidos

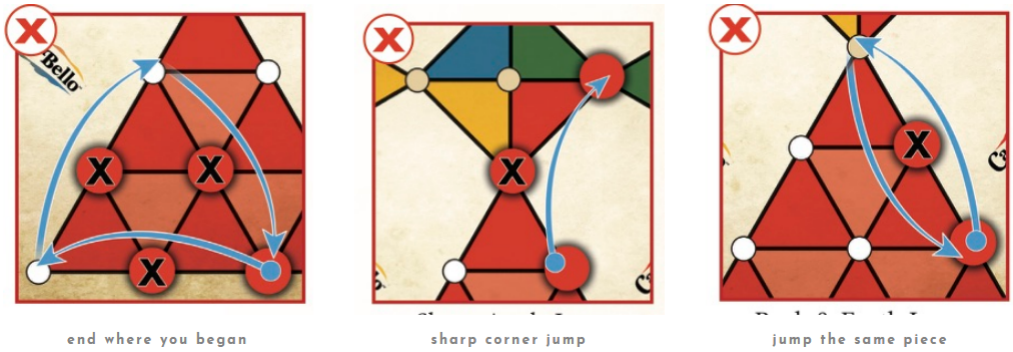

Figura 3: Movimentos não permitidos

3 Lógica do jogo

3.1 Representação do Estado do Jogo

A forma de representação do estado do tabuleiro usada foi uma lista com 9 listas. Nela estão presentes quatro tipos de peças diferentes, *pieceX1*, *pieceX2*, *pieceY1*, *pieceY2*, em que o X e Y distingue as peças de cada jogador e *noPiece* é um lugar do tabuleiro onde é possível mover as peças, mas onde naquele momento não está nenhuma.

3.1.1 Representação do estado inicial do tabuleiro

```
initialBoard([[empty,pieceX1,pieceX1,pieceX1,pieceX1,empty,
    empty,empty,empty],
[empty,empty,pieceX1,pieceX1,pieceX1,empty,empty,empty,
    pieceY1],
[empty,empty,empty,pieceX1,pieceX1,empty,empty,pieceY1,
    pieceY1],
[empty,empty,empty,empty,noPiece,empty,pieceY1,pieceY1,
    pieceY1],
[pieceX2,pieceX2,pieceX2,noPiece,noPiece,noPiece,pieceY1,
    pieceY1,pieceY1],
[pieceX2,pieceX2,pieceX2,empty,noPiece,empty,empty,empty,
    empty],
[pieceX2,pieceX2,empty,empty,pieceY2,pieceY2,empty,empty,
    empty],
[pieceX2,empty,empty,empty,pieceY2,pieceY2,pieceY2,empty,
    empty],
[empty,empty,empty,empty,pieceY2,pieceY2,pieceY2,pieceY2,
    empty]]).
```

3.2 Visualização do Tabuleiro

De forma a visualizarmos o tabuleiro, representado por uma matriz de nove linhas por nove colunas, foram construídos os seguintes predicados:

```
printFinalBoard([L|Ls]):-
    nl,
    printLetters,nl,
    printBoard([L|Ls],0),
    printLine.

printLetters:-write('     A     B     C     D     E     F
       G     H     I').

printSpaces:-write('   |     |     |     |     |     |     |
       |     |     |').

printBoard([],_).
printBoard([L|Ls],Y) :-
          printLine,nl,
          printSpaces,nl,
          Y1 is Y+1,
```

```
16          printFinalRow(L,Y1),nl,
17          printSpaces,nl,
18          printBoard(Ls,Y1).
19
20  printFinalRow([X|Xs],Y):-
21          write(Y),
22          write('  |'), printRow([X|Xs]).
23  printRow([X|Xs]):-
24          getSymbol(X,Piece),
25          write(' '), write(Piece),write('  | '),
26          printRow(Xs).
27  printRow([]).
28  printLine:-write('
    ---------------------------------------------------------').
```

A notar que cada tipo de peça é visualizada com um símbolo específico:

- ***empty*** representado com o símbolo **' '**;
- ***pieceX1*** representado com o símbolo **'X'**;
- ***pieceX2*** representado com o símbolo **'X'**;
- ***pieceY1*** representado com o símbolo **'Y'**;
- ***pieceY2*** representado com o símbolo **'Y'**;
- ***noPiece*** representado com o símbolo **'N'**.

```
| ?- initialBoard(B),printFinalBoard(B).
    A     B     C     D     E     F     G     H     I
  ---------------------------------------------------------
1 |     | X   | X   | X   | X   |     |     |     |     |
  ---------------------------------------------------------
2 |     |     | X   | X   | X   |     |     |     | Y   |
  ---------------------------------------------------------
3 |     |     |     | X   | X   |     |     | Y   | Y   |
  ---------------------------------------------------------
4 |     |     |     |     | N   |     | Y   | Y   | Y   |
  ---------------------------------------------------------
5 | X   | X   | X   | N   | N   | N   | Y   | Y   | Y   |
  ---------------------------------------------------------
6 | X   | X   | X   |     | N   |     |     |     |     |
  ---------------------------------------------------------
7 | X   | X   |     |     | Y   | Y   |     |     |     |
  ---------------------------------------------------------
8 | X   |     |     |     | Y   | Y   | Y   |     |     |
  ---------------------------------------------------------
9 |     |     |     |     | Y   | Y   | Y   | Y   |     |
  ---------------------------------------------------------
```

Figura 4: Imagem do tabuleiro correspondente ao output produzido pelo predicado de visualização

3.3 Lista de Jogadas Válidas

Em cada jogada, o utilizador deve inserir qual a peça que quer mover e quais as coordenadas de destino. De forma a validar a posição de origem, foi criado um predicado *validateSourcePiece(-Ncol, -Nrow, -Board, -Piece)* que verifica se o jogador escolheu uma das suas peças. A posição de destino da peça é testada no sentido de garantir que se trata de um movimento válido, através do predicado *validateDestinyPiece(-LastCol, -LastRow, -Ncol, -Nrow, -Board, -Piece, -Area, -BoardOut)* que invoca os predicados *checkIfCanMoveX(-Ncol, -Nrow, -LastCol, -LastRow, -Board, -Piece, -BoardOut, -Area)* e *checkIfCanMoveY(-Ncol, -Nrow, -LastCol, -LastRow, -Board, -Piece, -BoardOut, -Area))* para validar os movimentos do jogador X e Y, respetivamente.

3.4 Execução de Jogadas

A partir do momento em que o jogador decide a sua jogada, através dos predicados *chooseSourceCoords(+RowSource, +ColSource, -Board, -Piece)* e

chooseDestinyCoords(-RowSource, -ColSource, -Board, -Piece, +Area, +BoardOut) é verificado se o movimento é válido. Caso não seja validado, é pedida a inserção de novas posições, caso contrário é verificado se o jogador foi para uma posição onde não existiam peças. Se isso acontecer, o jogador pode saltar para uma nova posição e se esta estiver novamente vazia, o utilizador pode, pela última vez, saltar novamente, isto é, pode fazer saltos duplos ou triplos, pelo predicado *chooseNewJump(-Board, +BoardOut, -LastColPiece, -LastRowPiece, -LastRow,-LastCol, +Row, +Col, -Piece, -Area, +Continue)*. Caso o jogador salte para uma posição onde exista uma peça sua, esta é removida do jogo se for uma peça do adversário, é perguntado ao utilizador qual a peça do tabuleiro que quer remover, através do predicado choosePieceToRemove(-Board, +BoardOut).

3.5 Avaliação do Tabuleiro

A avaliação e manipulação do tabuleiro foi conseguida através dos predicados *setPiece(-BoardIn, -Nrow, -Ncol, -Piece, +BoardOut)*, *setOnRow(Pos, [Row—Remainder], Ncol, Piece, [Row—Newremainder])*, *setOnCol(1, [—Remainder], Piece, [Piece—Remainder])* e *getElement(-Board, -Nrow, -Ncol, +Element)*.

Para o nível de dificuldade elevada foi peremptória a avaliação do tabuleiro para que o computador comparasse todos os seus movimentos possíveis e escolhesse o melhor, isto é, aquele onde obtém o menor número de pontos. Esta avaliação é feita através dos predicados *evaluateBoards(-Board, +Points)* e *listOfBestMovements(+FinalList, -Board)* que avaliam todos os tabuleiros com todas as jogadas válidas e retornam uma lista ordenada, de forma crescente, pelo número de pontos seguida da posição. Assim, o primeiro elemento dessa lista é a melhor jogada que o computador pode efetuar. No entanto, não foi possível a sua completa implementação.

3.6 Final do Jogo

O jogo termina em duas situações diferentes, nomeadamente quando o jogador não possui mais movimentos válidos, pelo predicado *checkMoves(-Piece,*

-Board) ou quando não possui mais peças, através do predicado *checkPieces(-Piece,-Board)*. O predicado responsável por essa verificação dessas duas condições é o endGame(-Board).

Posteriormente e de modo a identificar o vencedor foram criados os predicados *calculatePoints(-Board, +PointsX, +PointsY)* e *checkWinner(-Board, -PointsX, -PointsY)* que calculam os pontos de cada jogador e verificam qual o vencedor, respetivamente. O cálculo da pontuação é baseado no número de peças que cada jogador tem dentro e fora da sua área de partida, em que é obtido 1 ponto para cada peça fora e 3 pontos para cada peça dentro da sua área de partida. O vencedor é o jogador com o menor número de pontos.

3.7 Jogada do Computador

Se o utilizador escolher os modos Computador contra Jogador ou Computador contra Computador, ele poderá escolher qual o nível de dificuldade do jogo.

No nível de dificuldade normal, foram elaborados predicados responsáveis por retornar posições e movimentos válidos de forma aleatória. Para a escolha da peça que quer mover foi criado o predicado *listOfValidSourceMoveX(+Board, -FinalListX)* que cria uma lista com as peças do computador e que é percorrida no predicado *listOfPiecesThatHasPossibleMoveX(-FinalList, +Board)* que cria uma outra com apenas peças que têm movimentos possíveis. Posteriormente uma das peças é escolhida aleatoriamente através do predicado *random(+L, +U, -R)*. Relativamente à escolha do movimento, é criado o predicado *listOfValidDestinyMove(-List, +LastRow, +LastCol, +Area, +Board)* que cria uma lista com todos os destinos válidos da peça que o computador escolheu anteriormente e retorna, aleatoriamente, um desses destinos.

Pelo contrário no nível de dificuldade elevada é feita uma avaliação do tabuleiro por parte do computador para escolher qual a sua melhor jogada. Assim, o computador em vez de selecionar aleatoriamente uma peça válida, escolhe qual a que lhe permite obter um menor número de pontos. Os predicados utilizados foram os enunciados no tópico **Avaliação do Tabuleiro**. A notar que esta funcionalidade não foi implementada na totalidade com sucesso, pela tentativa infrutuita de percorrer uma lista acedendo a elementos de outra.

4 Interface com o Utilizador

O módulo de interface com o utilizador é iniciado com um menu principal que permite ao utilizador nas opções 1., 2. e 3. jogar jogador contra jogador, *pc* contra jogador e *pc* contra *pc*, respetivamente. Na opção 4. aceder a um outro menu com as instruções de jogo e finalmente na opção 5., sair do jogo.

Figura 5: Menu principal

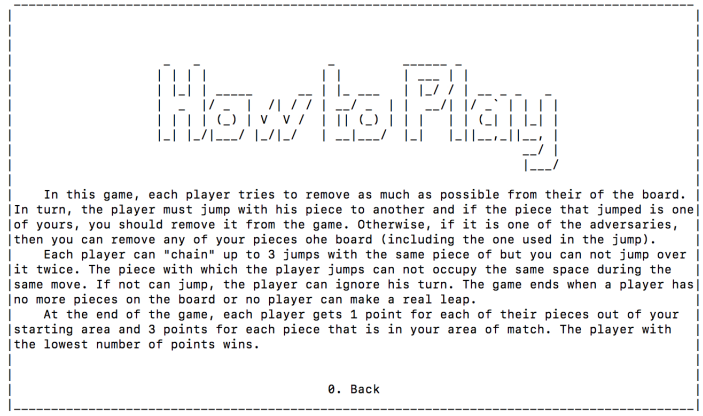

Figura 6: Menu "How To Play"

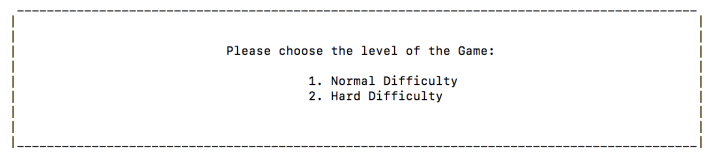

Figura 7: Menu "Set Level"

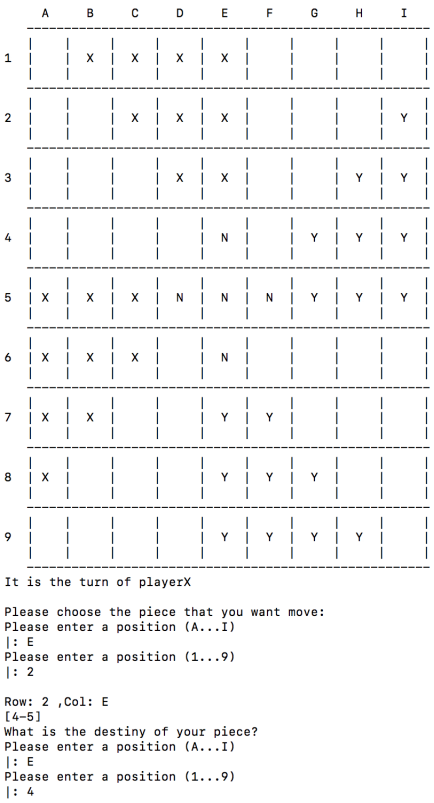

Figura 8: During the game

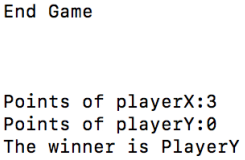

Figura 8: End Game

5 Conclusões

A realização deste projeto, para além de nos ser muito útil do ponto de vista algorítmico e estrutural, permitiu-nos perceber as notórias e múltiplas vantagens da linguagem *Prolog*.

Efetivamente, a solução implementada correspondeu ao que era exigido relativamente ao nível de dificuldade normal. No entanto, para o nível de dificuldade elevada a avaliação do tabuleiro poderia ter sido na totalidade implementada e com possíveis melhorias, centrando-se não só no cálculo das jogadas com o menor número de pontos, como também com as jogadas que permitissem remover o maior número de peças possível.

Em suma, este trabalho teve uma grande influência no nosso percurso como alunas de Engenharia Informática e Computação, pois permitiu-nos não só solidificar os conhecimentos lecionados, como também pôr em prática a construção dos predicados assentes nos principais pilares de *Prolog*, a unificação, a recursividade e o *backtraking*.

6 Bibliografia

[**1**]Clocksin, W. F.; Programming in prolog. ISBN: 0-387-58350-5
[**2**]*Campo Bello Game*. Acedido em Outubro e Novembro de 2017. Disponível em http://www.campobellogame.com/blog/2016/9/27/the-campo-bello-story-tuesday-august-2-2016
[**3**]*SWI-Prolog*. Acedido em Outubro e Novembro de 2017. Disponível em http://www.swi-prolog.org/

7 Anexos

Anexo I

Ficheiro "campoBello.pl"

```
:- include('gameLogic.pl').
:- include('menus.pl').
:- include('displayBoard.pl').
:- include('utilities.pl').
:- include('validateMoves.pl').
:- include('levelDifficulty.pl').

campoBello :- mainMenu.
```

Anexo II

Ficheiro "displayBoard.pl"

```
getSymbol(empty, ' ').
getSymbol(pieceX1, 'X').
getSymbol(pieceX2, 'X').
getSymbol(pieceY1, 'Y').
getSymbol(pieceY2, 'Y').
getSymbol(noPiece, 'N').

initialBoard([[empty,pieceX1,pieceX1,pieceX1,pieceX1,empty,
    empty,empty,empty],
[empty,empty,pieceX1,pieceX1,pieceX1,empty,empty,empty,
    pieceY1],
[empty,empty,empty,pieceX1,pieceX1,empty,empty,pieceY1,
    pieceY1],
[empty,empty,empty,empty,noPiece,empty,pieceY1,pieceY1,
    pieceY1],
[pieceX2,pieceX2,pieceX2,noPiece,noPiece,noPiece,pieceY1,
    pieceY1,pieceY1],
[pieceX2,pieceX2,pieceX2,empty,noPiece,empty,empty,empty,
    empty],
[pieceX2,pieceX2,empty,empty,pieceY2,pieceY2,empty,empty,
    empty],
[pieceX2,empty,empty,empty,pieceY2,pieceY2,pieceY2,empty,
    empty],
[empty,empty,empty,empty,pieceY2,pieceY2,pieceY2,pieceY2,
    empty]]).

finalBoard([[empty,noPiece,noPiece,noPiece,noPiece,empty,
    empty,empty,empty],
[empty,empty,noPiece,noPiece,noPiece,empty,empty,empty,
    noPiece],
[empty,empty,empty,pieceX1,noPiece,empty,empty,noPiece,
    noPiece],
[empty,empty,empty,empty,pieceY1,empty,noPiece,noPiece,
    noPiece],
[noPiece,noPiece,noPiece,noPiece,noPiece,noPiece,noPiece,
    noPiece,noPiece],
[noPiece,noPiece,pieceX2,empty,noPiece,empty,empty,empty,
    empty],
```

```
24 [noPiece,noPiece,empty,empty,noPiece,noPiece,empty,empty,
       empty],
25 [noPiece,empty,empty,empty,noPiece,noPiece,noPiece,empty,
       empty],
26 [empty,empty,empty,empty,noPiece,noPiece,noPiece,noPiece,
       empty]]).
27
28 middleBoard([[empty,pieceX1,noPiece,noPiece,noPiece,empty,
       empty,empty,empty],
29 [empty,empty,noPiece,noPiece,pieceY2,empty,empty,empty,
       noPiece],
30 [empty,empty,empty,pieceX1,noPiece,empty,empty,noPiece,
       noPiece],
31 [empty,empty,empty,empty,pieceY1,empty,noPiece,noPiece,
       noPiece],
32 [noPiece,pieceX1,noPiece,noPiece,noPiece,noPiece,noPiece,
       noPiece,noPiece],
33 [noPiece,noPiece,pieceX2,empty,noPiece,empty,empty,empty,
       empty],
34 [noPiece,noPiece,empty,empty,pieceY2,noPiece,empty,empty,
       empty],
35 [noPiece,empty,empty,empty,noPiece,noPiece,noPiece,empty,
       empty],
36 [empty,empty,empty,empty,noPiece,noPiece,noPiece,noPiece,
       empty]]).
37
38 printFinalBoard([L|Ls]):-
39     nl,
40     printLetters,nl,
41     printBoard([L|Ls],0),
42     printLine.
43
44 printLetters:-write('     A     B     C     D     E     F
       G     H     I').
45
46 printSpaces:-write('   |     |     |     |     |     |     |
       |     |     |').
47
48 printBoard([],_).
49 printBoard([L|Ls],Y) :-
50           printLine,nl,
51           printSpaces,nl,
52           Y1 is Y+1,
53           printFinalRow(L,Y1),nl,
54           printSpaces,nl,
55           printBoard(Ls,Y1).
56
57 printFinalRow([X|Xs],Y):-
58         write(Y),
59         write('  |'), printRow([X|Xs]).
60 printRow([X|Xs]):-
61         getSymbol(X,Piece),
62         write(' '), write(Piece),write('  | '),
63         printRow(Xs).
64 printRow([]).
65 printLine:-write('
```

```
    ----------------------------------------------------').
```

Anexo III

Ficheiro "gameLogic.pl"

```
:-use_module(library(lists)).
:-use_module(library(random)).
:-use_module(library(system)).

%Predicate responsible for the main game cycle
play(Board) :- mode_game(Curr_mode),
user_is(Curr_user),
chooseSourceCoords(RowSource, ColSource, Board, Piece,
    AskForDestinyPiece),
if_then_else(AskForDestinyPiece==0,
chooseDestinyCoords(RowSource, ColSource, Board, Piece,
    BoardOut),duplicate(Board,BoardOut)),nl,nl,
if_then_else(Curr_mode==2,
if_then_else(Curr_user=='pcX',set_user_is('player'),
    set_user_is('pcX')),
if_then_else(Curr_mode==3,
if_then_else(Curr_user=='pcX',set_user_is('pcY'),set_user_is
    ('pcX')),true)),
level(Curr_level),
write(Curr_level),
if_then_else(endGame(BoardOut),(nl,write('End Game'),
    checkWinner(BoardOut)),play(BoardOut)),
sleep(1).

%Predicate responsible for choosing the origin coordinates
chooseSourceCoords(RowSource, ColSource,Board,Piece,
    AskForDestinyPiece) :-   mode_game(Curr_mode),
user_is(Curr_user),
level(Curr_level),
if_then_else((Curr_mode == 1; Curr_user=='player'),
(AskForDestinyPiece is 0,
repeat,
player(Curr_player),nl,
write('It is the turn of '),
if_then_else(Curr_mode==1,write(Curr_player),write(Curr_user
    )),
nl,
write('Please choose the piece that you want move:'), nl,
write('Please enter a position (A...I)'),nl,
getChar(ColLetter),
once(letterToNumber(ColLetter, ColSource)),
write('Please enter a position (1...9)'),
nl,
getCode(RowSource),
validateSourcePiece(ColSource, RowSource,Board,Piece),
getPiece(Board,RowSource,ColSource,Piece)),
(if_then_else(Curr_level==1,
(if_then_else(Curr_user=='pcX',
listOfPiecesThatHasPossibleMoveX(FinalList,Board),
listOfPiecesThatHasPossibleMoveY(FinalList,Board)),
length(FinalList,LengthOfList),
```

```
45 if_then_else(LengthOfList==0,AskForDestinyPiece is 1,
       AskForDestinyPiece is 0),
46 random(0,LengthOfList,Index),
47 nth0(Index,FinalList,RowSource-ColSource),
48 getPiece(Board,RowSource,ColSource,Piece)),
49 (if_then_else(Curr_user=='pcX',
50 listOfPiecesThatHasPossibleMoveX(FinalList,Board),
51 listOfPiecesThatHasPossibleMoveY(FinalList,Board)),
52 listOfBestMovements(ListOfBestMoves,Board),
53 nth0(0,ListOfBestMoves,Points-RowSource-ColSource))))),
54
55 nl,write('Row: '),write(RowSource),write(' ,Col: '),
56 numberToLetter(ColSource,Letter),write(Letter),nl.
57
58 %Predicate responsible for choosing the destiny coordinates
59 chooseDestinyCoords(RowSource, ColSource, Board,Piece,
       BoardOut) :- mode_game(Curr_mode),
60 user_is(Curr_user),
61 level(Curr_level),
62 if_then_else(areaX1(RowSource,ColSource),Area='areaX1',
63 (if_then_else(areaX2(RowSource,ColSource),Area='areaX2',
64 (if_then_else(areaY1(RowSource,ColSource),Area='areaY1',
65 (if_then_else(areaY2(RowSource,ColSource),Area='areaY2',true
       ))))))),
66 listOfValidDestinyMove(List,RowSource,ColSource,Area,Board),
       length(List,LengthOfList),
67 write(List),
68 if_then_else(LengthOfList\=0,
69 (if_then_else((Curr_mode == 1; Curr_user=='player'),
70 (repeat,nl,
71 write('What is the destiny of your piece?'),
72 nl,
73 write('Please enter a position (A...I)'),
74 nl,
75 getChar(ColLetter),
76 once(letterToNumber(ColLetter, ColDestiny)),
77 write('Please enter a position (1...9)'),
78 nl,
79 getCode(RowDestiny),
80 validateDestinyPiece(ColSource,RowSource,ColDestiny,
       RowDestiny,Board,Piece,Area, BoardOut),
81 player(Curr_player),
82 if_then_else(Curr_player == 'playerX', set_player('playerY')
       ,set_player('playerX'))),
83 (if_then_else(Curr_level==1,
84 (random(0,LengthOfList,Index),
85 nth0(Index,List,RowDestiny-ColDestiny),
86 validateDestinyPiece(ColSource,RowSource,ColDestiny,
       RowDestiny,Board,Piece, Area,BoardOut)),
87 (listOfBestMovements(ListOfBestMoves,Board),
88 nth0(0,ListOfBestMoves,Points-RowDestiny-ColDestiny))))),
89 if_then_else(Curr_player == 'playerX', set_player('playerY')
       ,set_player('playerX'))),nl,
90 write('List Of Possible Moves: '),
91 write(List), write(' Row: '),write(RowDestiny), write(' Col:
       '),
```

```
92  numberToLetter(ColDestiny,Letter),write(Letter),nl.
93
94  %Predicate that returns a list with parts that have possible
        moves
95  listOfPiecesThatHasPossibleMoveX(FinalList,Board):-
96  saveElements(Board,'pieceX1',List1),
97  saveElements(Board,'pieceX2',List2),
98  append(List1,List2,ListOfDestiny),
99  scrollList(ListOfDestiny,FinalList,Board).
100
101 %Predicate that returns a list with parts that have possible
        moves
102 listOfPiecesThatHasPossibleMoveY(FinalList,BosaveElements(
        Board,'pieceY1',LsaveElements(Board,'pieceY2',Lappend(
        List1,List2,ListOfDesscrollList(ListOfDestiny,FinalList,
        Bo%Predicate that walks through a list filling them with
        positions that have possible scrollList([],scrollList([
        Nrow-Ncol|Rest], FinalList,Board):-
103 if_then_else(areaX1(Nrow,Ncol),Area='areaX1',
104 (if_then_else(areaX2(Nrow,Ncol),Area='areaX2',
105 (if_then_else(areaY1(Nrow,Ncol),Area='areaY1',
106 (if_then_else(areaY2(Nrow,Ncol),Area='areaY2',Area='areaX1')
        ))))))),
107 if_then_else(
108 % IF
109 (validateMovePC(Area,Ncol,Nrow,Col,Row,Board)),
110 % THEN
111 (scrollList(Rest, List_Temp,Board), append(List_Temp, [Nrow-
        Ncol], FinalList)),
112 % ELSE
113 scrollList(Rest, FinalList,Board)%Predicate that returns a
        list with valid target molistOfValidDestinyMove(List,
        LastRow,LastCol,Area,Board) :-
114 if_then_else(setof(Nrow-Ncol,validateMovePC(Area,LastCol,
        LastRow,Ncol,Nrow,Board),List),true,
115 findall(Nrow-Ncol,validateMovePC(Area,LastCol,LastRow,Ncol,
        Nrow,Board),List)).
116
117 %Predicate that checks which pieces the player can choose to
        move
118 validateSourcePiece(Ncol, Nrow,Board,Piece) :- getPiece(
        Board, Nrow, Ncol, Piece),
119 user_is(Curr_user),
120 player(Curr_player),
121 mode_game(Curr_mode),
122 if_then_else(Curr_mode==1,if_then_else(Curr_player=='playerX
        ',
123 (Piece \= 'pieceY1',
124 Piece \= 'pieceY2'),
125 (Piece \= 'pieceX1',
126 Piece \= 'pieceX2')),
127 if_then_else(Curr_user='pcX',
128 (Piece \= 'pieceY1',
129 Piece \= 'pieceY2'),
130 (Piece \= 'pieceX1',
131 Piece \= 'pieceX2'))),
```

```
132 Piece \= 'empty',
133 Piece \= 'noPiece'.
134
135 %Predicate that verifies the valid movements
136 validateDestinyPiece(LastCol,LastRow,Ncol,Nrow,Board, Piece,
    Area,BoardOut) :- if_then_else((Piece=='pieceX1';Piece=='
    pieceX2'),
137 checkIfCanMoveX(Ncol, Nrow, LastCol,LastRow,Board,Piece,
    BoardOut,Area),
138 checkIfCanMoveY(Ncol, Nrow, LastCol,LastRow,Board,Piece,
    BoardOut,Area)).
139
140 %Predicate that the piece to which it jumped is a noPiece
    and in case it is asked to jump again.
141 checkIfIsNotNoPiece(Board,BoardOut,LastColPiece,LastRowPiece
    ,Row,Col,FinalRow,FinalCol,Piece,Area,NewContinue):-
    repeat,
142 chooseNewJump(Board,BoardOut,LastColPiece,LastRowPiece,Row,
    Col,FinalRow,FinalCol,Piece,Area,NewContinue),
143 if_then_else(NewContinue\=1,
144 (getPiece(BoardOut, FinalRow, FinalCol, SecondPiece),
145 SecondPiece \= 'noPiece'),true).
146
147 %Predicate that calls the function responsible for printing
    the board after a new jump.
148 printBoardAfterJump(Row,Col,LastRow,LastCol,Board,BoardOut,
    Piece) :- setPiece(Board,LastRow,LastCol,'noPiece',
    BoardOut2),
149 setPiece(BoardOut2,Row,Col,Piece,BoardOut),
150 printFinalBoard(BoardOut),nl.
151
152 %Predicate that checks if it did not jump to the same piece
    in the same movement.
153 checkIfIsNotRedo(LastColPiece,LastRowPiece,ColPiece,RowPiece
    ):-LastColPiece==ColPiece,
154 LastRowPiece==RowPiece.
155
156 %Predicate responsible for executing all rules of the game
    for player X, namely when the user can make single,
    double or triple jumps, validating them.
157 checkIfCanMoveX(Ncol,Nrow,LastCol,LastRow,Board,Piece,
    BoardOut,Area) :-
158 validateMove(Area, LastCol, LastRow, Ncol, Nrow,Board),
159 getPiece(Board, Nrow, Ncol, NewPiece),
160 if_then_else((NewPiece=='noPiece'),
161 (printBoardAfterJump(Nrow,Ncol,LastRow,LastCol,Board,
    BoardOut2,Piece),(chooseNewJump(BoardOut2,BoardOut3,
    LastCol,LastRow,Nrow, Ncol,Row,Col,Piece,Area,Continue),
162 if_then_else(Continue\=1,(getPiece(Board,Row,Col,Piece2),
163 if_then_else(areaX1(Row,Col),Area2='areaX1',
164 (if_then_else(areaX2(Row,Col),Area2='areaX2',
165 (if_then_else(areaY1(Row,Col),Area2='areaY1',
166 (if_then_else(areaY2(Row,Col),Area2='areaY2',Area2=Area)))))
    )),
167 if_then_else(Piece2=='noPiece',(checkIfIsNotNoPiece(
    BoardOut3,BoardOut4,Ncol,Nrow,Row,Col,FinalRow,FinalCol,
```

```
    SecondPiece,Area2,NewContinue),
168 if_then_else(NewContinue\=1,
169 (if_then_else(areaX1(FinalRow,FinalCol),Area3='areaX1',
170 (if_then_else(areaX2(FinalRow,FinalCol),Area3='areaX2',
171 (if_then_else(areaY1(FinalRow,FinalCol),Area3='areaY1',
172 (if_then_else(areaY2(FinalRow,FinalCol),Area3='areaY2',Area3
    =Area2)))))))),
173 (if_then_else((SecondPiece=='pieceY1';SecondPiece=='pieceY2'
    ),
174 (choosePieceToRemove(BoardOut4, BoardOut5),
175 setPiece(BoardOut5,FinalRow,FinalCol,Piece,BoardOut6),
    setPiece(BoardOut6,FinalRow,FinalCol,'noPiece',BoardOut))
    ,(validateMove(Area3, Col, Row, FinalCol, FinalRow,
    BoardOut4),
176 setPiece(BoardOut4,FinalRow,FinalCol,Piece,BoardOut))))),
    duplicate(BoardOut4,BoardOut))),
177 (if_then_else((Piece2=='pieceY1';Piece2=='pieceY2'),
178 (choosePieceToRemove(BoardOut3, BoardOut4),
179 setPiece(BoardOut4,Row,Col,Piece,BoardOut)),(setPiece(
    BoardOut3,Row,Col,Piece,BoardOut))))))),duplicate(
    BoardOut3,BoardOut)))),
180 (if_then_else((NewPiece=='pieceY1';NewPiece=='pieceY2'), (
    validateMove(Area, LastCol, LastRow, Ncol, Nrow,Board),
181 choosePieceToRemove(Board, BoardOut2),setPiece(BoardOut2,
    LastRow,LastCol,'noPiece',BoardOut3),setPiece(BoardOut3,
    Nrow,Ncol,Piece,BoardOut)),
182 (validateMove(Area, LastCol, LastRow, Ncol, Nrow,Board),
    setPiece(Board,LastRow,LastCol,'noPiece',BoardOut2),
183 setPiece(BoardOut2,Nrow,Ncol,Piece,BoardOut))))),
184 printFinalBoard(BoardOut).
185
186 %Predicate responsible for executing all rules of the game
    for player Y, namely when the user can make single,
    double or triple jumps, validating them.
187 checkIfCanMoveY(Ncol,Nrow,LastCol,LastRow,Board,Piece,
    BoardOut,Area) :-
188 validateMove(Area, LastCol, LastRow, Ncol, Nrow,Board),
189 getPiece(Board, Nrow, Ncol, NewPiece),
190 if_then_else((NewPiece=='noPiece'),
191 (printBoardAfterJump(Nrow,Ncol,LastRow,LastCol,Board,
    BoardOut2,Piece),(chooseNewJump(BoardOut2,BoardOut3,
    LastCol,LastRow,Nrow, Ncol,Row,Col,Piece,Area,Continue),
192 if_then_else(Continue\=1,(getPiece(Board,Row,Col,Piece2),
193 if_then_else(areaX1(Row,Col),Area2='areaX1',
194 (if_then_else(areaX2(Row,Col),Area2='areaX2',
195 (if_then_else(areaY1(Row,Col),Area2='areaY1',
196 (if_then_else(areaY2(Row,Col),Area2='areaY2',Area2=Area)))))
    )),
197 if_then_else(Piece2=='noPiece',(checkIfIsNotNoPiece(
    BoardOut3,BoardOut4,Ncol,Nrow,Row,Col,FinalRow,FinalCol,
    SecondPiece,Area2,NewContinue),
198 if_then_else(NewContinue\=1,
199 (if_then_else(areaX1(FinalRow,FinalCol),Area3='areaX1',
200 (if_then_else(areaX2(FinalRow,FinalCol),Area3='areaX2',
201 (if_then_else(areaY1(FinalRow,FinalCol),Area3='areaY1',
202 (if_then_else(areaY2(FinalRow,FinalCol),Area3='areaY2',Area3
```

```
    =Area2))))))),
203 (if_then_else((SecondPiece=='pieceX1';SecondPiece=='pieceX2'
    ),
204 (choosePieceToRemove(BoardOut4, BoardOut5),
205 setPiece(BoardOut5,FinalRow,FinalCol,Piece,BoardOut6),
    setPiece(BoardOut6,FinalRow,FinalCol,'noPiece',BoardOut))
    ,(validateMove(Area3, Col, Row, FinalCol, FinalRow,
    BoardOut4),
206 setPiece(BoardOut4,FinalRow,FinalCol,Piece,BoardOut))))),
    duplicate(BoardOut3,BoardOut))),
207 (if_then_else((Piece2=='pieceX1';Piece2=='pieceX2'),
208 (choosePieceToRemove(BoardOut3, BoardOut4),
209 setPiece(BoardOut4,Row,Col,Piece,BoardOut)),(setPiece(
    BoardOut3,Row,Col,Piece,BoardOut))))),duplicate(
    BoardOut2,BoardOut)))),
210 (if_then_else((NewPiece=='pieceX1';NewPiece=='pieceX2'), (
    validateMove(Area, LastCol, LastRow, Ncol, Nrow,Board),
211 choosePieceToRemove(Board, BoardOut2),setPiece(BoardOut2,
    LastRow,LastCol,'noPiece',BoardOut3),setPiece(BoardOut3,
    Nrow,Ncol,Piece,BoardOut)),
212 (validateMove(Area, LastCol, LastRow, Ncol, Nrow,Board),
    setPiece(Board,LastRow,LastCol,'noPiece',BoardOut2),
213 setPiece(BoardOut2,Nrow,Ncol,Piece,BoardOut))))),
214 printFinalBoard(BoardOut).
215
216 %Predicate responsible for requesting a new leap to the
    player when a "noPiece" part is found
217 chooseNewJump(Board,BoardOut,LastColPiece,LastRowPiece,
    LastRow,LastCol,Row,Col,Piece,Area,Continue) :-
218 if_then_else(areaX1(LastRow,LastCol),AreaDestiny='areaX1',
219 (if_then_else(areaX2(LastRow,LastCol),AreaDestiny='areaX2',
220 (if_then_else(areaY1(LastRow,LastCol),AreaDestiny='areaY1',
221 (if_then_else(areaY2(LastRow,LastCol),AreaDestiny='areaY2',
    AreaDestiny=Area))))))),
222 mode_game(Curr_mode),
223 user_is(Curr_user),
224 listOfValidDestinyMove(List,LastRow,LastCol,AreaDestiny,
    Board),length(List,LengthOfList),
225 if_then_else(LengthOfList\=0,
226 if_then_else((Curr_mode == 1; Curr_user=='player'),
227 (repeat,nl,write('You need jump one more time!'),
228 nl,
229 write('Please enter a position (A...I)'),
230 nl,
231 getChar(ColLetter),
232 once(letterToNumber(ColLetter, Col)),
233 write('Please enter a position (1...9)'),
234 nl,
235 getCode(Row),
236 if_then_else(checkIfIsNotRedo(LastColPiece,LastRowPiece,Col,
    Row),
237 if_then_else(LengthOfList==1,(write('Cant move to this Piece
    '),
238 Continue is 1,setPiece(Board,LastRowPiece,LastColPiece,Piece
    ,BoardOut2),
239 getPiece(Board,Row,Col,PieceChoosen),setPiece(BoardOut2,Row,
```

```
    Col,PieceChoosen,BoardOut)),
(Continue is 0,
chooseNewJump(Board,BoardOut,LastColPiece,LastRowPiece,
    LastRow,LastCol,Row,Col,Piece,Area,Continue))),
(Continue is 0,
validateMove(AreaDestiny, LastCol, LastRow, Col, Row,Board),
setPiece(Board,Row,Col,Piece,BoardOut2),
setPiece(BoardOut2,LastRow,LastCol,'noPiece',BoardOut),
(nl,write('List Of Possible Moves: '),write(List),
write(' Row: '),write(Row),write('  Col: '),write(Col),
printFinalBoard(BoardOut))))),
(nl,write('Row: '),write(LastRow),write('  Col: '),write(
    LastCol),nl,
nl,write('Lista: '),write(List),
if_then_else(checkIfIsNotRedo(LastColPiece,LastRowPiece,Col,
    Row),
if_then_else(LengthOfList==1,(write('Cant move to this Piece
    '),
Continue is 1,duplicate(Board,BoardOut),
printFinalBoard(BoardOut)),
(Continue is 0,   random(0,LengthOfList,Index),
nth0(Index,List,Row-Col),
setPiece(Board,Row,Col,Piece,BoardOut2),
setPiece(BoardOut2,LastRow,LastCol,'noPiece',BoardOut),
nl,write('List Of Possible Moves: '),write(List),
write(' Row: '),write(Row),write('  Col: '),write(Col),
printFinalBoard(BoardOut))),
(Continue is 0,
random(0,LengthOfList,Index),
nth0(Index,List,Row-Col),
setPiece(Board,Row,Col,Piece,BoardOut2),
setPiece(BoardOut2,LastRow,LastCol,'noPiece',BoardOut),
nl,write('List Of Possible Moves: '),write(List),
write(' Row: '),write(Row),write('  Col: '),write(Col),
printFinalBoard(BoardOut))))),
(write('Without Possible Moves!'),
duplicate(Board,BoardOut),
printFinalBoard(BoardOut))).


%Predicate responsible for asking the user which part of the
    board he wants to remove
choosePieceToRemove(Board, BoardOut) :-mode_game(Curr_mode),
user_is(Curr_user),
player(Curr_player),
if_then_else((Curr_mode == 1; Curr_user == 'player'),
(repeat,nl, write('What is the piece that you want remove?')
    ,
nl,
write('Please enter a position (A...I)'),
nl,
getChar(ColLetter),
once(letterToNumber(ColLetter, Col)),
write('Please enter a position (1...9)'),
nl,
```

```
289 getCode(Row),
290 if_then_else(Curr_mode==1,
291 if_then_else(Curr_player=='playerX',
292 checkIfCanRemoveX(Board, Col, Row),
293 checkIfCanRemoveY(Board, Col, Row)),
294 if_then_else(Curr_user=='pcX',
295 checkIfCanRemoveX(Board, Col, Row),
296 checkIfCanRemoveY(Board, Col, Row)))),
297 (if_then_else(Curr_user=='pcX',
298 listOfPiecesThatCanRemoveX(Board,List),
299 listOfPiecesThatCanRemoveY(Board,List)),
300 length(List,LengthOfList),
301 random(0,LengthOfList,Index),
302 nth0(Index,List,Row-Col))),
303 setPiece(Board,Row,Col,'noPiece',BoardOut).
304
305 %Predicate that returns a list of the parts that the X
      computer can remove
306 listOfPiecesThatCanRemoveX(Board,List):-if_then_else(setof(
      Nrow-Ncol,checkIfCanRemoveX(Board,Ncol,Nrow),List),true,
307 findall(Nrow-Ncol,checkIfCanRemoveX(Board,Ncol,Nrow),List)).
308
309 %Predicate that returns a list of the parts that the X
      computer can remove
310 listOfPiecesThatCanRemoveY(Board,List):-if_then_else(setof(
      Nrow-Ncol,checkIfCanRemoveY(Board,Ncol,Nrow),List),true,
311 findall(Nrow-Ncol,checkIfCanRemoveY(Board,Ncol,Nrow),List)).
312
313 %Predicate that tests whether the piece chosen by player X
      to remove is one of his own pieces.
314 checkIfCanRemoveX(Board, Col, Row) :- getPiece(Board, Row,
      Col, NewPiece),
315 NewPiece \= 'empty',
316 NewPiece \= 'pieceY1',
317 NewPiece \= 'pieceY2',
318 NewPiece \= 'noPiece'.
319
320 %Predicate that tests whether the piece chosen by player X
      to remove is one of his own pieces.
321 checkIfCanRemoveY(Board, Col, Row) :- getPiece(Board, Row,
      Col, NewPiece),
322 NewPiece \= 'empty',
323 NewPiece \= 'pieceX1',
324 NewPiece \= 'pieceX2',
325 NewPiece \= 'noPiece'.
326
327 %Predicate that returns the part in a given row and column.
328 getPiece(Board, Nrow, Ncol, Piece) :-  getElement(Board,Nrow
      ,Ncol,Piece).
329
330 %Predicate that changes a certain piece in the board and
      updates the new board.
331 setPiece(BoardIn, Nrow, Ncol, Piece, BoardOut) :- setOnRow(
      Nrow, BoardIn, Ncol, Piece, BoardOut).
332
333 %Predicate that changes the part on the board in a certain
```

```
    line
setOnRow(1, [Row|Remainder], Ncol, Piece, [Newrow|Remainder
    ]):- setOnCol(Ncol, Row, Piece, Newrow).
setOnRow(Pos, [Row|Remainder], Ncol, Piece, [Row|
    Newremainder]):- Pos @> 1,
Next is Pos-1,
setOnRow(Next, Remainder, Ncol, Piece, Newremainder).
%Predicate that changes the part on the board in a given
    column
setOnCol(1, [_|Remainder], Piece, [Piece|Remainder]).
setOnCol(Pos, [X|Remainder], Piece, [X|Newremainder]):- Pos
    @> 1,
Next is Pos-1,
setOnCol(Next, Remainder, Piece, Newremainder).
%Predicate that does an if than else
if_then_else(If, Then,_):- If,!, Then.
if_then_else(_, _, Else):- Else.

%Predicate that returns the element that is contained in the
     tray in a given column and row.
getElement(Board,Nrow,Ncol,Element) :- nth1(Nrow, Board, Row
    ),
nth1(Ncol,Row,Element).

%Predicate that checks if is in area X1
areaX1(Nrow,Ncol):- (Ncol@>1,
Ncol@<6,
Nrow@>0,
Nrow@<5).
%Predicate that checks if is in area X2
areaX2(Nrow,Ncol):- (Ncol@>0,
Ncol@<5,
Nrow@>4,
Nrow@<9).

%Predicate that checks if is in area Y1
areaY1(Nrow,Ncol):- (Ncol@>5,
Ncol@<10,
Nrow@>1,
Nrow@<6).

%Predicate that checks if is in area Y2
areaY2(Nrow,Ncol):-(Ncol@>4,
Ncol@<9,
Nrow@>5,
Nrow@<10).

%Predicate that checks if is the area of playerX
areaX(Nrow,Ncol):- (Ncol@>1,
Ncol@<6,
Nrow@>0,
Nrow@<5);
(Ncol@>0,
Ncol@<5,
Nrow@>4,
Nrow@<9).
```

```
383
384 %Predicate that checks if is the area of playerY
385 areaY(Nrow,Ncol):- (Ncol@>5,
386 Ncol@<10,
387 Nrow@>1,
388 Nrow@<6);
389 (Ncol@>4,
390 Ncol@<9,
391 Nrow@>5,
392 Nrow@<10).
393
394 %Predicate thata does a list with the pieces of the player
395 saveElements(Board,'pieceX1',List):- if_then_else(setof(Nrow
    -Ncol,getElement(Board,Nrow,Ncol,'pieceX1'),List),
396 true,findall(Nrow-Ncol,getElement(Board,Nrow,Ncol,'pieceX1')
    ,List)).
397 saveElements(Board,'pieceX2',List):- if_then_else(setof(Nrow
    -Ncol,getElement(Board,Nrow,Ncol,'pieceX2'),List),
398 true,findall(Nrow-Ncol,getElement(Board,Nrow,Ncol,'pieceX2')
    ,List)).
399 saveElements(Board,'pieceY1',List):- if_then_else(setof(Nrow
    -Ncol,getElement(Board,Nrow,Ncol,'pieceY1'),List),
400 true,findall(Nrow-Ncol,getElement(Board,Nrow,Ncol,'pieceY1')
    ,List)).
401 saveElements(Board,'pieceY2',List):- if_then_else(setof(Nrow
    -Ncol,getElement(Board,Nrow,Ncol,'pieceY2'),List),
402 true,findall(Nrow-Ncol,getElement(Board,Nrow,Ncol,'pieceY2')
    ,List)).
403
404 %Predicatethat checks which area the piece is in and
    calculate the points
405 getNrowNcol([],PointsXIn,PointsXOut,'playerX').
406 getNrowNcol([],PointsYIn,PointsYOut,'playerY').
407 getNrowNcol([Nrow-Ncol|Rest],PointsXIn,PointsXOut,'playerX')
    :-
408 if_then_else(areaX(Nrow,Ncol),PointsXOut is PointsXIn+3,
409 PointsXOut is PointsXIn+1),nl,
410 getNrowNcol(Rest,PointsXOut,PointsXOutNew,'playerX').
411 getNrowNcol([Nrow-Ncol|Rest],PointsYIn,PointsYOut,'playerY')
    :-
412 if_then_else(areaY(Nrow,Ncol),PointsYOut is PointsYIn+3,
413 PointsYOut is PointsYIn+1),
414 getNrowNcol(Rest,PointsYOut,PointsYOutNew,'playerY').
415
416 %Predicate that checks if the playerX has pieces on the
    board
417 checkIfExistsPiecesX(Board) :- saveElements(Board,'pieceX1',
    List),
418 saveElements(Board,'pieceX2',List2),
419 append(List,List2,FinalList),
420 length(FinalList,LengthOfFinalList),
421 if_then_else(LengthOfFinalList==0,fail,true).
422
423 %Predicate that checks if the playerY has pieces on the
    board
424 checkIfExistsPiecesY(Board) :-  saveElements(Board,'pieceY1'
```

```
    ,List),
saveElements(Board,'pieceY2',List2),
append(List,List2,FinalList),
length(FinalList,LengthOfFinalList),
if_then_else(LengthOfFinalList==0,fail,true).

%Predicate that checks if is the end of the game
endGame(Board) :- listOfPiecesThatHasPossibleMoveX(FinalList
    ,Board),
length(FinalList,LengthOfFinalList),
listOfPiecesThatHasPossibleMoveY(FinalList2,Board),
length(FinalList2,LengthOfFinalList2),
(if_then_else(checkIfExistsPiecesY(Board),fail,true);
if_then_else(checkIfExistsPiecesX(Board),fail,true);
if_then_else(LengthOfFinalList==0,true,fail);
if_then_else(LengthOfFinalList2==0,true,fail)).

%Predicate that calculates the points of each player
calculatePoints(Board,PointsX,PointsY):- saveElements(Board,
    'pieceX1',List),
saveElements(Board,'pieceX2',List2),
append(List,List2,FinalListX),
getNrowNcol(FinalListX,0,PointsX,'playerX'),
length(FinalListX,LengthOfFinalListX),
if_then_else(LengthOfFinalListX==0,PointsX is 0,true),
saveElements(Board,'pieceY1',List3),
saveElements(Board,'pieceY2',List4),
append(List3,List4,FinalListY),
getNrowNcol(FinalListY,0,PointsY,'playerY'),
length(FinalListY,LengthOfFinalListY),
if_then_else(LengthOfFinalListY==0,PointsY is 0,true),
nl,
write('Points of playerX:'), write(PointsX),nl,
write('Points of playerY:'), write(PointsY),nl.

%Predicate that checks the winner of the game
checkWinner(Board) :- calculatePoints(Board,PointsX,PointsY)
    ,
if_then_else(PointsX@>PointsY,write('The winner is PlayerY')
    ,write('The winner is PlayerX')).
```

Anexo IV

Ficheiro "levelDisfficulty.pl"

```
%Predicate that calculates the player's points through the
    received board
evaluateBoards(Board,Points):-user_is(Curr_user),
if_then_else(Curr_user=='pcX',
(saveElements(Board,'pieceX1',List),
saveElements(Board,'pieceX2',List2),
append(List,List2,FinalList),
getNrowNcol(FinalList,0,Points,'playerX')),

(saveElements(Board,'pieceY1',List),
saveElements(Board,'pieceY2',List2),
append(List,List2,FinalList),
```

```
12  getNrowNcol(FinalList,0,Points,'playerY'))).
13
14  %Instance Nrow e Ncol
15  is_member([],_,_).
16  is_member([Nrow-Ncol|Rest],Row,Col):-if_then_else((Row ==
        Nrow,Col == Ncol),true,is_member(Rest,Row,Col)).
17
18  %Predicate that makes a list with the best plays,
        increasingly ordered by the number of points of each
        player
19  listOfBestMovements(FinalList,Board):-
20    user_is(Curr_user),
21    setof(Points-Nrow-Ncol,(
22    if_then_else(Curr_user=='pcX',
23    listOfPiecesThatHasPossibleMoveX(List,Board),
24    listOfPiecesThatHasPossibleMoveY(List,Board)),
25    is_member(List,Nrow,Ncol),
26    if_then_else(areaX1(Nrow,Ncol),Area='areaX1',
27    (if_then_else(areaX2(Nrow,Ncol),Area='areaX2',
28    (if_then_else(areaY1(Nrow,Ncol),Area='areaY1',
29    (if_then_else(areaY2(Nrow,Ncol),Area='areaY2',Area='areaX1
          ')))))))),write(Nrow),write(Ncol),
30    validateMovePC(Area,Ncol,Nrow,Col,Row,Board),getPiece(
          Board,Nrow,Ncol,Piece),
31    setPiece(Board,Col,Row,Piece,BoardOut2),setPiece(BoardOut2
          ,Nrow,Ncol,'noPiece',BoardOut),
32    evaluateBoards(BoardOut,Points)),FinalList).
```

Anexo V

Ficheiro "menus.pl"

```
1   printMainMenu:-
2     nl,nl,nl,
3     write('
          ------------------------------------------------------------------------
          '),nl,
4     write('         |

          |'),nl,
5     write('         |

          |'),nl,
6     write('         |

          |'),nl,
7     write('         |

          |'),nl,
8     write('         |  _____    ______    _     _    ______
          ______      ______    ______    __        __       ______  |
          '),nl,
9     write('         | |  ___| |  __  | | |   | | |   __   | |  __
           |   |  __  | |  ____| |  |      |  |     |  __  | |'),nl
          ,
10    write('         | | |     | |__| | | /__/ | | |__| | | |  |
           |   | |__| | | |____  |  |      |  |     | |  | | |'),nl
```

```
    ,
11  write('         | | |     | |__| | | |  | | |  ____| | |  |
        |   |  __  | |  ____| |  |     |  |    | |  | | |'),nl
    ,
12  write('         | | |___  | |  | | | |  | | | |      | |__|
        |   | |__| | | |____  |  |___  |  |___ | |__| | |'),nl
    ,
13  write('         | |_____| |_|  |_| |_|  |_| |_|      |
    ______|   |______| |______| |______| |______||______| |
    '),nl,
14  write('         |

    |'),nl,
15  write('         |

    |'),nl,
16  write('         |                          Francisca Leao
    Cerquinho Ribeiro da Fonseca                          |')
    ,nl,
17  write('         |
    Mariana Lopes Silva
    |'),nl,
18  write('         |

    |'),nl,
19  write('         |

    |'),nl,
20  write('         |                                1.Start
    Game Player vs Player                                 |')
    ,nl,
21  write('         |

    |'),nl,
22  write('         |                                  2.Start
    Game PC vs Player                                    |'),
    nl,
23  write('         |

    |'),nl,
24  write('         |                                   3.Start
        Game PC vs PC                                     |'),nl
    ,
25  write('         |

    |'),nl,
26  write('         |                                     4.
    Set Difficulty                                      |'),
    nl,
27  write('         |

    |'),nl,
28  write('         |                                     5.
    How to Play                                        |'),
    nl,
29  write('         |
```

```
    |'),nl,
30  write('         |
                                                6.Exit
                                               |'),nl,
31  write('         |

    |'),nl,
32  write('         |

    |'),nl,
33  write('         |
    ------------------------------------------------------------------------------
    | '),nl,nl,nl,nl,
34  write('                                          Choose an
     option, please!                                    '),
    nl, nl, nl, nl.
35
36
37  printHowToPlayMenu:-
38    nl,nl,nl,
39    write('
        ------------------------------------------------------------------------------
        '),nl,
40    write('         |

        |'),nl,
41    write('         |

        |'),nl,
42    write('         |

        |'),nl,
43    write('         |                      _   _
                          _          ______ _
                                          |'),nl,
44    write('         |                       | | | |
                     | |         | ___ | |
                                        |'),nl,
45    write('         |                       | |_| | _____       __
       | |_ ___    | |_/ / | __ _ _   _                      |'
      ),nl,
46    write('         |                       |  _  |/ _  |  /| / /
       | __/ _  | |  __/| |/ _` | | | |                     |'
      ),nl,
47    write('         |                       | | | | (_) | V  V /
       | || (_) | | |   | | (_| | |_| |                      |'
      ),nl,
48    write('         |                       |_| |_/|___/ |_/|_/
        | __|___/  |_|   |_||__,_||__, |                     |
      '),nl,
49    write('         |

        __/ |                    |'),nl,
50    write('         |
```

```
        |___/                   |'),nl,
51      write('         |
                                |'),nl,
52      write('         |    In this game, each player tries to
            remove as much as possible from their of the board. |
            '),nl,
53      write('         |In turn, the player must jump with his
            piece to another and if the piece that jumped is one|
            '),nl,
54      write('         |of yours, you should remove it from the
            game. Otherwise, if it is one of the adversaries,  |'
            ),nl,
55      write('         |then you can remove any of your pieces
            ohe board (including the one used in the jump).     |
            '),nl,
56      write('         |    Each player can "chain" up to 3
            jumps with the same piece of but you can not jump
            over |'),nl,
57      write('         |it twice. The piece with which the
            player jumps can not occupy the same space during the
               |'),nl,
58      write('         |same move. If not can jump, the player
            can ignore his turn. The game ends when a player has|
            '),nl,
59      write('         |no more pieces on the board or no player
             can make a real leap.                              |'
            ),nl,
60      write('         |    At the end of the game, each player
            gets 1 point for each of their pieces out of your  |'
            ),nl,
61      write('         |starting area and 3 points for each
            piece that is in your area of match. The player with
              |'),nl,
62      write('         |the lowest number of points wins.
                                |'),nl,
63      write('         |
                                |'),nl,
64      write('         |
                                |'),nl,
65      write('         |
                                                    0. Back
                                                    |'),nl,
66      write('         |
            ------------------------------------------------------------------------
            | '),nl,nl,nl,nl.
67
68      printSetLevelMenu:-
69        nl,nl,nl,
70        write('
              ------------------------------------------------------------------------
            '),nl,
71        write('         |
```

```
        |'),nl,
72      write('         |

        |'),nl,
73      write('         |                                Please
        choose the level of the Game:
                                      |'),nl,
74      write('         |

        |'),nl,
75      write('         |
                                                  1. Normal
        Difficulty                                 |'),nl,
76      write('         |
                                                  2. Hard
        Difficulty                                    |'),nl,
77      write('         |

        |'),nl,
78      write('         |

        |'),nl,
79      write('         |
        ------------------------------------------------------------------------
        | '),nl,nl,nl,nl.
80
81      mainMenu :- printMainMenu,
82      now(X), setrand(X),
83      read(Input),
84      set_mode_game(Input),
85      readInput(Input).
86
87      readInput(0) :- mainMenu.
88
89      readInput(1) :- initialBoard(Board),printFinalBoard(
        Board),
90      play(Board),
91      mainMenu.
92
93      readInput(2) :- initialBoard(Board), printFinalBoard(
        Board),
94      set_user_is('pcX'),
95      play(Board),
96      mainMenu.
97
98      readInput(3) :- initialBoard(Board), printFinalBoard(
        Board),
99      set_user_is('pcX'),
100     play(Board),
101     mainMenu.
102
103     readInput(4) :- printSetLevelMenu,
104     read(Input),
105     readInput2(Input).
106
```

```
107       readInput2(1) :- set_level(1),
108       mainMenu.
109
110       readInput2(2) :- set_level(2),
111       mainMenu.
112
113       readInput(5) :- printHowToPlayMenu,
114       read(Input),
115       readInput(Input).
116
117       readInput(6) :- write('Exiting...').
```

Anexo VI

Ficheiro "utilities.pl"

```
1  %Predicate that read a char
2  getChar(Input) :- get_char(_Input),
3                    get_char(Input).
4
5  %Predicate that read a number
6  getCode(Input) :- get_code(_TempInput),
7                    get_code(TempInput),
8                    Input is TempInput - 48.
9
10 :-dynamic player/1.
11 :-dynamic mode_game/1.
12 :-dynamic user_is/1.
13 :-dynamic level/1.
14
15 mode_game(1).
16 player(playerX).
17 user_is(player).
18 level(1).
19
20 %Predicate that sets the player
21 set_player(Player):-
22   nonvar(Player),
23   retract(player(_)),
24   asserta(player(Player)).
25
26 %Predicate that sets the mode game
27 set_mode_game(Newmode):-
28   nonvar(Newmode),
29   integer(Newmode),
30   retract(mode_game(_)),
31   asserta(mode_game(Newmode)).
32
33 %Predicate that sets the user
34 set_user_is(NewPlayer):-
35   nonvar(NewPlayer),
36   retract(user_is(_)),
37   asserta(user_is(NewPlayer)).
38
39 %Predicate that sets the level
40 set_level(Level):-
41     nonvar(Level),
```

```
    integer(Level),
    retract(level(_)),
    asserta(level(Level)).

%Predicate that converts each letter into its respective number
    letterToNumber('A',1).
    letterToNumber('B',2).
    letterToNumber('C',3).
    letterToNumber('D',4).
    letterToNumber('E',5).
    letterToNumber('F',6).
    letterToNumber('G',7).
    letterToNumber('H',8).
    letterToNumber('I',9).

%Predicate that converts each number into its respective letter
    numberToLetter(1,'A').
    numberToLetter(2,'B').
    numberToLetter(3,'C').
    numberToLetter(4,'D').
    numberToLetter(5,'E').
    numberToLetter(6,'F').
    numberToLetter(7,'G').
    numberToLetter(8,'H').
    numberToLetter(9,'I').

%Predicate copying one board to another
duplicate(_Old,_New):-fail.
duplicate(_Old,_Old).
```

Anexo VII

Ficheiro "validateMoves.pl"

```
%Predicate that validates the plays of the pc in the area X1
validateMovePC('areaX1',LastCol,LastRow,Ncol,Nrow,Board) :-
    if_then_else(((LastCol==5,LastRow==3);(LastCol==5,LastRow==4);(LastCol==5,LastRow==2)),
(Nrow is LastRow+2,
Ncol is LastCol),
((Ncol is LastCol+2,
Nrow is LastRow+2,
getPiece(Board,Nrow,Ncol,Piece),
Piece\='empty');
(Nrow is LastRow,
Ncol is LastCol+2,
getPiece(Board,Nrow,Ncol,Piece),
Piece\='empty');
(Nrow is LastRow+2,
Ncol is LastCol,
getPiece(Board,Nrow,Ncol,Piece),
Piece\='empty'))),
if_then_else((LastCol==4,LastRow==3),
(Nrow is LastRow+2,
Ncol is LastCol+2),true).
```

```
20 %Predicate that validates the plays of the pc in the area X2
21 validateMovePC('areaX2',LastCol,LastRow,Ncol,Nrow,Board) :-
       if_then_else((LastCol==2,LastRow==5),
22 (Nrow is LastRow+2,
23 Ncol is LastCol;
24 (Nrow is LastRow,
25 Ncol is LastCol+2)),
26 ((Nrow is LastRow-2,
27 Ncol is LastCol+2,
28 getPiece(Board,Nrow,Ncol,Piece),
29 Piece\='empty');
30 (Nrow is LastRow,
31 Ncol is LastCol+2,
32 getPiece(Board,Nrow,Ncol,Piece),
33 Piece\='empty');
34 (Nrow is LastRow+2,
35 Ncol is LastCol,
36 getPiece(Board,Nrow,Ncol,Piece),
37 Piece\='empty'))),
38
39 if_then_else((LastCol==3,LastRow==5),
40 (Nrow is LastRow,
41 Ncol is LastCol+2),true),
42
43 if_then_else((LastCol==3,LastRow==6),
44 (Nrow is LastRow-2,
45 Ncol is LastCol+2),true).
46
47 %Predicate that validates the plays of the pc in the area Y1
48 validateMovePC('areaY1',LastCol,LastRow,Ncol,Nrow,Board) :-
       if_then_else(((LastCol==7,LastRow==5);(LastCol==8,LastRow
       ==5);(LastCol==6,LastRow==5)),
49 (Ncol is LastCol-2,
50 Nrow is LastRow),
51 ((Ncol is LastCol-2,
52 Nrow is LastRow+2,
53 getPiece(Board,Nrow,Ncol,Piece),
54 Piece\='empty');
55 (Ncol is LastCol,
56 Nrow is LastRow+2,
57 getPiece(Board,Nrow,Ncol,Piece),
58 Piece\='empty');
59 (Ncol is LastCol-2,
60 Nrow is LastRow,
61 getPiece(Board,Nrow,Ncol,Piece),
62 Piece\='empty'))),
63
64 if_then_else((LastCol==7,LastRow==4),
65 (Nrow is LastRow+2,
66 Ncol is LastCol-2),true).
67
68 %Predicate that validates the plays of the pc in the area Y2
69 validateMovePC('areaY2',LastCol,LastRow,Ncol,Nrow,Board) :-
70   if_then_else(((LastCol==5,LastRow==8);(LastCol==5,LastRow
         ==7);(LastCol==5,LastRow==6)),
71   (Ncol is LastCol,
```

```
72    Nrow is LastRow-2),
73    ((Ncol is LastCol-2,
74    Nrow is LastRow-2,
75    getPiece(Board,Nrow,Ncol,Piece),
76    Piece\='empty');
77    (Ncol is LastCol-2,
78    Nrow is LastRow,
79    getPiece(Board,Nrow,Ncol,Piece),
80    Piece\='empty');
81    (Ncol is LastCol,
82    Nrow is LastRow-2,
83    getPiece(Board,Nrow,Ncol,Piece),
84    Piece\='empty'))),
85
86    if_then_else((LastCol==6,LastRow==5),
87    (Ncol is LastCol-2,
88    Nrow is LastRow),true).
89    %Predicate that validates the plays of the player in the
          area X1
90    validateMove('areaX1',LastCol,LastRow,Ncol,Nrow,Board) :-
          if_then_else(((LastCol==5,LastRow==3);(LastCol==5,
          LastRow==4);(LastCol==5,LastRow==2)),
91    (RowTemp is LastRow+2,
92    Nrow == RowTemp,
93    Ncol == LastCol),
94    ((ColTemp is LastCol+2,
95    RowTemp is LastRow+2,
96    Ncol == ColTemp,
97    Nrow == RowTemp,
98    getPiece(Board,Nrow,Ncol,Piece),
99    Piece\='empty');
100   (ColTemp is LastCol+2,
101   Nrow == LastRow,
102   Ncol == ColTemp,
103   getPiece(Board,Nrow,Ncol,Piece),
104   Piece\='empty');
105   (RowTemp is LastRow+2,
106   Nrow == RowTemp,
107   Ncol == LastCol,
108   getPiece(Board,Nrow,Ncol,Piece),
109   Piece\='empty'))),
110   if_then_else((LastCol==4,LastRow==3),
111   (RowTemp is LastRow+2,
112   ColTemp is LastCol+2,
113   Nrow == RowTemp,
114   Ncol == ColTemp),true).
115
116   %Predicate that validates the plays of the player in the
          area X2
117   validateMove('areaX2',LastCol,LastRow,Ncol,Nrow,Board) :-
          if_then_else((LastCol==2,LastRow==5),
118   (RowTemp is LastRow+2,
119   Nrow == RowTemp,
120   Ncol == LastCol;
121   (ColTemp is LastCol+2,
122   Nrow == LastRow,
```

```
123     Ncol == ColTemp)),
124     ((ColTemp is LastCol+2,
125     RowTemp is LastRow-2,
126     Nrow == RowTemp,
127     Ncol == ColTemp,
128     getPiece(Board,Nrow,Ncol,Piece),
129     Piece\='empty');
130     (ColTemp is LastCol+2,
131     Nrow == LastRow,
132     Ncol == ColTemp,
133     getPiece(Board,Nrow,Ncol,Piece),
134     Piece\='empty');
135     (RowTemp is LastRow+2,
136     Nrow == RowTemp,
137     Ncol == LastCol,
138     getPiece(Board,Nrow,Ncol,Piece),
139     Piece\='empty'))),
140
141     if_then_else((LastCol==3,LastRow==5),
142     (ColTemp is LastCol+2,
143     Nrow == LastRow,
144     Ncol == ColTemp),true),
145
146     if_then_else((LastCol==3,LastRow==6),
147     (ColTemp is LastCol+2,
148     RowTemp is LastRow-2,
149     Nrow == RowTemp,
150     Ncol == ColTemp),true).
151
152     %Predicate that validates the plays of the player in the
        area Y1
153     validateMove('areaY1',LastCol,LastRow,Ncol,Nrow,Board) :-
        if_then_else(((LastCol==7,LastRow==5);(LastCol==8,
        LastRow==5);(LastCol==6,LastRow==5)),
154     (ColTemp is LastCol-2,
155     Ncol == ColTemp,
156     Nrow == LastRow),
157     ((ColTemp is LastCol-2,
158     Ncol == ColTemp,
159     RowTemp is LastRow+2,
160     Nrow == RowTemp,
161     getPiece(Board,Nrow,Ncol,Piece),
162     Piece\='empty');
163     (Ncol == LastCol,
164     RowTemp is LastRow+2,
165     Nrow == RowTemp,
166     getPiece(Board,Nrow,Ncol,Piece),
167     Piece\='empty');
168     (ColTemp is LastCol-2,
169     Ncol == ColTemp,
170     Nrow == LastRow,
171     getPiece(Board,Nrow,Ncol,Piece),
172     Piece\='empty'))),
173
174     if_then_else((LastCol==7,LastRow==4),
175     (RowTemp is LastRow+2,
```

```
176   Nrow == RowTemp,
177   ColTemp is LastCol-2,
178   Ncol == ColTemp),true).
179
180   %Predicate that validates the plays of the player in the
          area Y2
181   validateMove('areaY2',LastCol,LastRow,Ncol,Nrow,Board) :-
           if_then_else(((LastCol==5,LastRow==8);(LastCol==5,
          LastRow==7);(LastCol==5,LastRow==6)),
182   (RowTemp is LastRow-2,
183   Ncol == LastCol,
184   Nrow == RowTemp),
185   ((ColTemp is LastCol-2,
186   RowTemp is LastRow-2,
187   Ncol == ColTemp,
188   Nrow == RowTemp,
189   getPiece(Board,Nrow,Ncol,Piece),
190   Piece\='empty');
191   (ColTemp is LastCol-2,
192   Ncol == ColTemp,
193   Nrow == LastRow,
194   getPiece(Board,Nrow,Ncol,Piece),
195   Piece\='empty');
196   (RowTemp is LastRow-2,
197   Ncol == LastCol,
198   Nrow == RowTemp,
199   getPiece(Board,Nrow,Ncol,Piece),
200   Piece\='empty'))),
201
202   if_then_else((LastCol==6,LastRow==5),
203   (ColTemp is LastCol-2,
204   Ncol == ColTemp,
205   Nrow == LastRow),true).
```