Hecatomb

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Hecatomb_2:

Francisco Couto - ei12189@fe.up.pt
Joel Dinís - ei12064@fe.up.pt

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

8 de Novembro de 2015

Resumo

O primeiro trabalho da unidade curricular de Programação em Lógica consistiu no desenvolvimento de um jogo de tabuleiro em *prolog*, neste caso, o *Hecatomb*.

A implementação teria de contemplar três modos de jogo: jogador contra jogador, jogador contra computador e computador contra computador.

Todas as jogadas baseiam-se num *game engine* que permite decidir quais das jogadas são válidas e quais não são. No caso de uma jogada de um humano o *input* dado é verificado, mas na do computador foi necessário, para cada jogada dada como válida, guarda-la para posteriormente selecionar apenas uma (*random* no caso do modo fácil e por *value* da jogada no caso do modo difícil).

Numa análise superficial do trabalho o seu objectivo foi concluído com sucesso. Todos os modos de jogos estão operacionais e permitem que se usufrua sem limitações do jogo.

Em relação à implementação é notório que o *prolog* é uma linguagem ideal para implementar um jogo deste género pela simplicidade de implementação de um motor de jogo bem como de um processo automato de seleção e aplicação de jogadas.

Índice

1. Introdução

No âmbito da unidade curricular de *PLOG* foi-nos pedido para implementarmos um jogo de tabuleiro de forma a nos familiarizarmos com os princípios básicos da linguagem de *prolog,* nomeadamente o *backtracking*, manipulação de listas e o uso do operador *cut*.

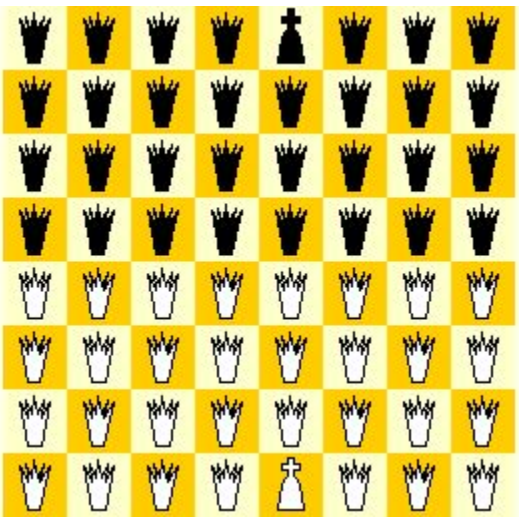
O relatório estará estruturado de forma a fazer uma breve introdução ao jogo, às suas principais regras e a forma como estas foram traduzidas em código. Será, no fim, feita uma conclusão sobre os resultados obtidos bem como sobre o que o grupo retirou da realização do trabalho.

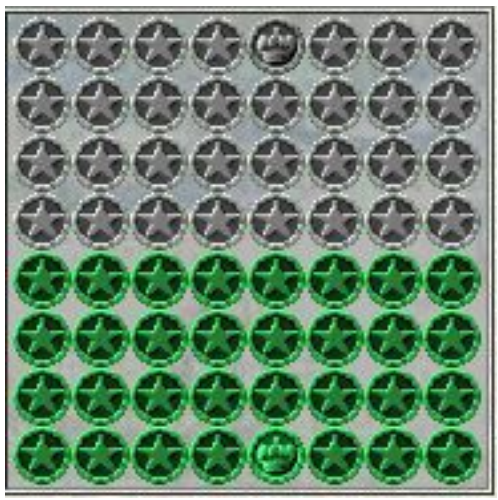
2. O *Hecatomb*

O *hecatomb* é um jogo de tabuleiro que deriva de uma adaptação do xadrez. Foi uma adaptação de *Kevin Maroney* datada de 1999 e que consiste em 64 casas, distribuídas por 31 rainhas e 1 rei para cada jogador.

O objectivo do jogo é igual ao do xadrez, ou seja, capturar o rei inimigo. Para isso os jogadores dispoêm de 5 turnos cada um. A estratégia a adaptar por cada jogador terá de ter em conta a proteção do seu próprio rei de forma a que não sejam surpreendidos por um movimento do adversário.

Caso no fim dos 10 turnos não haja um vencedor não existe qualquer sistema de pontuação e o jogo termina empatado.





Figuras 1 e 2: Representação hipotética de um tabuleiro de jogo de *Hecatomb*

3. Lógica do Jogo

a. Representação do Estado do Jogo

O tabuleiro é representado por uma lista de listas com dimensão de 8 casas por 8. A rainhas do jogador 1 são representadas por um **$** e o seu rei por um **+**, as do jogador 2 por **&** e um ***** respectivamente.

Existe ainda um estado extra que representa uma casa vazia: ‘ ‘.

```
board(B):-
    B=[
        [' $ ',' $ ',' $ ',' $ ',' + ',' $ ',' $ ',' $ '],
        [' $ ',' $ ',' $ ',' $ ',' $ ',' $ ',' $ ',' $ '],
        [' $ ',' $ ',' $ ',' $ ',' $ ',' $ ',' $ ',' $ '],
        [' $ ',' $ ',' $ ',' $ ',' $ ',' $ ',' $ ',' $ '],
        [' & ',' & ',' & ',' & ',' & ',' & ',' & ',' & '],
        [' & ',' & ',' & ',' & ',' & ',' & ',' & ',' & '],
        [' & ',' & ',' & ',' & ',' & ',' & ',' & ',' & '],
        [' & ',' & ',' & ',' & ',' * ',' & ',' & ',' & ']
        ].
```

Figura 3: código prolog para a definição do Board

b. Visualização do Tabuleiro

```
| ?- heca.

   0   1   2   3   4   5   6   7
0  $ | $ | $ | $ | + | $ | $ | $ |
1  $ | $ | $ | $ | $ | $ | $ | $ |
2  $ | $ | $ | $ | $ | $ | $ | $ |
3  $ | $ | $ | $ | $ | $ | $ | $ |
4  & | & | & | & | & | & | & | & |
5  & | & | & | & | & | & | & | & |
6  & | & | & | & | & | & | & | & |
7  & | & | & | & | * | & | & | & |
```

Figura 4: predicado heca que permite ao utilizador verificar o tabuleiro inicial de jogo

```
   0   1   2   3   4   5   6   7
0  $ | & | $ | $ | + | $ | $ | $ |
1  $ | $ | $ | $ | $ | $ | $ | $ |
2  $ | $ | $ | $ | $ | $ | $ | $ |
3  $ | $ | $ |   | $ | $ | $ | $ |
4  & | & | & | & | & | & | & | & |
5  & | & | & | & | & | & | & | & |
6  & | & | & | & |   | & | & | & |
7  $ | & | & | & | * | & | & | & |
```

Figura 5:representação intermédia do tabuleiro após duas jogadas

c. Lista de Jogadas Válidas

```
|: pvp.
Prepare to face another player!

Player number 2 will start the game!
   0   1   2   3   4   5   6   7
0  $ | $ | $ | $ | + | $ | $ | $ |
1  $ | $ | $ | $ | $ | $ | $ | $ |
2  $ | $ | $ | $ | $ | $ | $ | $ |
3  $ | $ | $ | $ | $ | $ | $ | $ |
4  & | & | & | & | & | & | & | & |
5  & | & | & | & | & | & | & | & |
6  & | & | & | & | & | & | & | & |
7  & | & | & | & | * | & | & | & |
Player 2 turn:
Coordenada X da peÃ§a a mover : |: 7.
Coordenada Y da peÃ§a a mover : |: 4.
Coordenada X da casa destino : |: 7.
Coordenada Y da casa destino : |: 3.
```

```
   0   1   2   3   4   5   6   7
0  $ | $ | $ | $ | + | $ | $ | $ |
1  $ | $ | $ | $ | $ | $ | $ | $ |
2  $ | $ | $ | $ | $ | $ | $ | $ |
3  $ | $ | $ | $ | $ | $ | $ | & |
4  & | & | & | & | & | & | & |   |
5  & | & | & | & | & | & | & | & |
6  & | & | & | & | & | & | & | & |
7  & | & | & | & | * | & | & | & |
```

Figura 4 e 5: Representação inicial do tabuleiro e posterior representação com uma jogada feita pelo jogador 2

Todas as peças podem mover-se ortogonal ou diagonalmente, porém os reis apenas o poderão fazer caso a posição final seja adjacente à sua posição inicial.

Se porventura movêssemos uma peça e entre a posição inicial e a posição final existisse uma peça essa jogada também não seria válida e seria pedido que repetíssemos a jogada. Exemplificado na figura seguinte.

```
   0   1   2   3   4   5   6   7
0  $ | $ | $ | $ | + | $ | $ | $ |
1  $ | $ | $ | $ | $ | $ | $ | $ |
2  $ | $ | $ | $ | $ | $ | $ | $ |
3  $ | $ | $ | $ | $ | $ | $ | & |
4  & | & | & | & | & | & | & |   |
5  & | & | & | & | & | & | & | & |
6  & | & | & | & | & | & | & | & |
7  & | & | & | & | * | & | & | & |
Player 1 turn:
Coordenada X da peÃ§a a mover : |: 7.
Coordenada Y da peÃ§a a mover : |: 2.
Coordenada X da casa destino : |: 7.
Coordenada Y da casa destino : |: 5.
```

Figura 6: *Input* do jogador 1 que, com a representação demonstrada levaria a que a jogada não fosse aceite

Para que a execução das jogadas seja possível existe um predicado extremamente importante que recebe como parâmetros o *board* atual, duas coordenadas (X,Y) e uma peça e que retorna um **novo *board*** com a posição especificada modificada com a nova peça.

```
replace( L , Y , X , Z , R ) :-
  append(RowPfx,[Row|RowSfx],L),
  length(RowPfx,X) ,
  append(ColPfx,[_|ColSfx],Row) ,
  length(ColPfx,Y) ,
  append(ColPfx,[Z|ColSfx],RowNew) ,
  append(RowPfx,[RowNew|RowSfx],R).
```

d. Execução de Jogadas

```
movePiece(Xi,Yi,Xf,Yf,Player,Board):-
        repeat,
        getPlayerMovement(Xi,Yi,Xf,Yf),
        positionsDifferent(Xi,Yi,Xf,Yf),
        checkOutOfBounds(Xi,Yi,Xf,Yf),
        checkCorrectPiece(Player,Board,Xi,Yi,Xf,Yf),
        checkMovement(Xi,Yi,Xf,Yf,Board).

movement(Xi,Yi,Xf,Yf,Player,Board):-movePiece(Xi,Yi,Xf,Yf,Player,Board),!.
```

O predicado **movePiece** consiste numa junção de condições (verificadas por vários predicados) que permitem validar uma jogada e atribuir a Xi,Yi,Xf e Yf coordenadas válidas para serem usadas no motor de jogo (predicados getPiece e replace referidos anteriormente).

O predicado **getPlayerMovement** obtém o input do utilizador para a jogada desejada. O **positionsDifferent** permite verificar se não estamos a jogar a peça para a mesma posição, complementado pelo **CheckOutOfBounds** que não permite que estas coordenadas se encontrem fora do tabuleiro.

De seguida os predicados **CheckCorrectPiece** e **checkMovement** verificam, respectivamente, se estou a comer uma peça do adversário (ou se me movimento para uma casa vazia) e se o meu movimento é valido (condições de movimento para cada peça já descritas e caso encontre peças a meio do percurso).

e. Final do Jogo

```
% Function to stop the game once we reach the 10th turn
gameDraw(Board):-printB(Board), write('The Game ended in a draw (reached 10 turns), this is the final board!').

%Checks if player 1 has won
game_over(Board):-
        \+ (getPiece(_,_,Board,' + ')),
        write('Game has ended! Player 2 wins!'), nl.

%Checks if player 2 has won
game_over(Board):-
        \+ (getPiece(_,_,Board,' * ')),
        write('Game has ended! Player 1 wins!'), nl.
```

É verificado se o jogo já terminou através de dois predicados:

-***gameDraw***: permite verificar se o jogo terminou empatado, ou seja, se chegamos ao fim dos 10 turnos e não houve um vencedor;

-***game_over***: permite verificar se houve um vencedor,ou seja, se um dos reis já não se encontra no tabuleiro;

f. Jogada do Computador

O trabalho contempla dois modos de jogo para o *bot*, um fácil e um difícil. Desse modo, foram então implementados dois predicados distintos que permitem **gerar diferentes jogadas** consoante o modo escolhido.

```
getRandomMove(Board,Player,Value,X,Y,XF,YF):-
    listAllPossibleMoves(Board,Player,Value,List),
    random_member(X-Y-XF-YF-Value,List).

getBestMove(Board,Player,Value,X,Y,XF,YF):-
    listAllPossibleMoves(Board,Player,Value,List),
    chooseBestMove(List,X-Y-XF-YF-Value).
```

Para ambos os modos são **geradas todas as jogadas possíveis (listAllPossibleMoves)** a realizar. Para o modo fácil uma dessas jogadas é selecionada **aleatoriamente (getRandomMove)** enquanto que para o difícil, existindo um **valor associado** a cada jogada, é selecionada a melhor possível (**getBestMove**).

```
%retrieve all the possible movements to list
listAllPossibleMoves(Board,Player,Value,List):-
    findall(X-Y-XF-YF-Value,
    allPossibleMoves(Board,Player,Value,X,Y,XF,YF),List).
```

O valor associado a cada jogada é atribuído de acordo com a **fórmula da distância**, ou seja, quanto maior for a distância que uma peça pode percorrer no tabuleiro, maior sera o valor da jogada.

```
%recursivly checks for the best value play
maxList([], Max, Max).
maxList([Xi-Yi-Xf-Yf-V|T], CXi-CYi-CXf-CYf-Cv, Max) :-
        ( V > Cv -> maxList(T, Xi-Yi-Xf-Yf-V, Max)
        ; maxList(T, CXi-CYi-CXf-CYf-Cv, Max) ).
```

NOTA: formula da distância = $\sqrt{(yf-yi)^2+(xf-xi)^2}$

4. Interface com o Utilizador

Menu para escolher o tipo de jogo

```
%Initial Menu Predicate, Lets the player choose the game mode
menu:-
write('*------------------------*'),nl,
write('*  WELCOME TO HECATOMB  !*'),nl,
write('*------------------------*'),nl,
write('Escolha um modo de jogo: pvp/pvb/bvb'),nl,nl,
read(Choice),gameMode(Choice),nl.
```

Menu para escolher a dificuldade (player vs bot e bot vs bot)

```
% Auxiliar menu to allow human to choose bot difficulty
menuDificuldade(Dificuldade):-
        print('Dificuldade do Jogo'), nl,
        print('Fácil (f)'), nl,
        print('Dificil (d)'), nl,
        read(Dificuldade).
```

Menu para requisitar *input* das coordenadas da jogada (pvp e pvb)

```
%Retrieves input from the player about his desired move
getPlayerMovement(Xi,Yi,Xf,Yf) :-
        print('Coordenada X da peça a mover : '), read(Xi),
        print('Coordenada Y da peça a mover : '), read(Yi),
        print('Coordenada X da casa destino : '), read(Xf),
        print('Coordenada Y da casa destino : '), read(Yf).
```

Acima encontram-se os principais menus de interface com o utilizador. O primeiro é o menu principal do jogo, permite que naveguemos até ao modo de jogo pretendido e que acedamos ao segundo menu se for necessário escolher a dificuldade do *bot*.

A cada jogada do humano é então utilizado o terceiro menu que solicita, coordenada a coordenada, o *input* do utilizador para a posição inicial e final.

Sem eles seria impossível desenvolver o jogo, tornando-se cruciais para a realização de jogadas e escolha do motor de jogo a utilizar.

5. Conclusões

Em suma, o trabalho elaborado foi um bom desafio e um bom ponto de partida para ficarmos familiarizados com a linguagem do *prolog*. A maior dificuldade foi encontrada no desenvolvimento do *bot*, nomeadamente de forma a impedir que se guiasse por uma lógica básica de geração aleatória de coordenadas.

No final, no nosso entender, foram implementados todos os requisitos à partida estipulados nomeadamente no que diz respeito aos vários modos de jogos e de dificuldade que teriam de existir.

Como melhorias a implementar talvez fosse possível experimentar **diferentes tipos de funções de avaliação** para o *bot*, ou a criação de uma **nova heurística** que permitisse extrair o máximo deste modo de jogo e que proporcionasse ao user um desafio mais difícil.

Bibliografia

http://homepages.di.fc.ul.pt/~jpn/gv/hecatomb.htm
http://www.thegamesjournal.com/articles/VaryingChess.shtml
http://www.pathguy.com/chess/Hecatomb.htm
http://stackoverflow.com/
http://www.zillions-of-games.com/cgi-bin/zilligames/submissions.cgi?do=show;id=487

Código Prolog

```
/* -*- Mode:Prolog; coding:iso-8859-1; -*- */
:-use_module(library(random)). /*used to randomize which player starts*/
:-use_module(library(lists)).
:-use_module(library(system)).

%Initializes the Board (8x8)
board(B):-
   B=[
      [' $ ',' $ ',' $ ',' $ ',' + ',' $ ',' $ ',' $ '],
      [' $ ',' $ ',' $ ',' $ ',' $ ',' $ ',' $ ',' $ '],
      [' $ ',' $ ',' $ ',' $ ',' $ ',' $ ',' $ ',' $ '],
      [' $ ',' $ ',' $ ',' $ ',' $ ',' $ ',' $ ',' $ '],
      [' & ',' & ',' & ',' & ',' & ',' & ',' & ',' & '],
      [' & ',' & ',' & ',' & ',' & ',' & ',' & ',' & '],
      [' & ',' & ',' & ',' & ',' & ',' & ',' & ',' & '],
      [' & ',' & ',' & ',' & ',' * ',' & ',' & ',' & ']
      ].

%Allow the players to visualize the initial board and plan their moves
heca:-board(B),nl, printB(B).

%Auxiliar function to print the board in a user friendly way
printB(H):-
      write('  0  1  2  3  4  5  6  7'), nl,
      printB(H, 0).

printB([], _).

printB([H|T], Line):-
      write(Line), write(' '), printL(H), nl,
      NLine is Line+1,
      printB(T, NLine).

printL([]).

printL([H|T]):-
      write(H),write('|'), printL(T).
```

```
%Initial Menu Predicate, Lets the player choose the game mode
menu:-
write('*-----------------------*'),nl,
write('*  WELCOME TO HECATOMB  !*'),nl,
write('*-----------------------*'),nl,
write('Escolha um modo de jogo: pvp/pvb/bvb'),nl,nl,
read(Choice),gameMode(Choice),nl.

%Game choice menu settings preparation
gameMode(pvp):-write('Prepare to face another player!'),nl,nl, play(pvp).

gameMode(pvb):-write('Prepare to face the computer!'),nl,nl,
      menuDificuldade(D),
      (D == 'f' ->play('hbf');
      play('hbd')).

gameMode(bvb):-write('Computer against itself!'),nl,nl,
      menuDificuldade(D),
      (D == 'f' ->play(bvb,'bbf');
      play(bvb,'bbd')).

gameMode(_):-write('Wrong option, choose a new one!'), nl, read(Choice),
gameMode(Choice),nl.


%Launch the game in the desired move and difficulty
play(pvp):-
   random(1,3, Player), write('Player number '),
   write(Player), write(' will start the game!'),nl,
   board(B),printB(B),pvpgame(Player, B, 10).

play(hbf):-random(1,3, Player),
   ((Player == 1)->write('Human starts playing!');
   write('Bot starts playing!')),
   nl,board(B),printB(B),
   pvbgame(Player, B, 10, hbf).

play(hbd):-random(1,3, Player),
   ((Player == 1)->write('Human starts playing!');
   write('Bot starts playing!')),
   nl,board(B),printB(B),
   pvbgame(Player, B, 10, hbd).
```

```
play(bvb,Dif):-random(1,3, Player),write('Player number '),
   write(Player), write(' will start the game!'),nl,
   board(B),printB(B),
   bvbgame(Player, B, 10,Dif).

%Checks if the game is over
pvpgame(_,Board,_):-
      game_over(Board).

%Checks if the game is a draw
pvpgame(_,Board,0):-gameDraw(Board).

%Game Engine for a player versus player game
pvpgame(P,Board,Turns):-
   Turns > 0,
   Turns2 is Turns-1,
   printPlayer(P,h),
   movement(Xi,Yi,Xf,Yf,P,Board),
   getPiece(Xi,Yi,Board, Element),
   replace(Board, Xf, Yf, Element, B2),
   replace(B2, Xi, Yi, '   ', B3),
printB(B3),changePlayer(P,Novo),pvpgame(Novo,B3,Turns2).

%check if we shall stop the game now and not continue to the next play
pvbgame(_,Board,_,_):-
      game_over(Board).

%the game reached 10 turns
pvbgame(_,Board,0,_):-gameDraw(Board).

%Game Engine for a player versus bot game - bot is always the second player (2)
pvbgame(P,Board,Turns,Dif):-
   Turns > 0,
   Turns2 is Turns-1,
   printPlayer(P,h),
   ((P == 1)->movement(Xi,Yi,Xf,Yf,P,Board);
   ((Dif == hbf)->getRandomMove(Board,2,_,Xi,Yi,Xf,Yf);
   getBestMove(Board,2,_,Xi,Yi,Xf,Yf))),
   printPlay(P,Xf,Yf),
   getPiece(Xi,Yi,Board,Element),
   replace(Board, Xf, Yf, Element, B2),
```

```
   replace(B2, Xi, Yi, '   ', B3), printB(B3),sleep(1),changePlayer(P,Novo),
pvbgame(Novo,B3,Turns2,Dif).

bvbgame(_,Board,_,_):-
      game_over(Board).

bvbgame(_,Board,0,_):-gameDraw(Board).

%Game Engine for a bot versus bot game
bvbgame(P,Board,Turns,Dif):-
   Turns > 0,
   Turns2 is Turns-1,
   printPlayer(P,h),
   ((Dif == bbf)->getRandomMove(Board,P,_,Xi,Yi,Xf,Yf);
   getBestMove(Board,P,_,Xi,Yi,Xf,Yf)),
   getPiece(Xi,Yi,Board,Element),
   replace(Board, Xf, Yf, Element, B2),
   replace(B2, Xi, Yi, '   ', B3), printB(B3),
   printPlay(P,Xf,Yf),
   sleep(2),
   changePlayer(P,Novo),
   bvbgame(Novo,B3,Turns2,Dif).


%Predicate that allows the replacement of a piece in the desired position of the
matrix
replace( L , Y , X , Z , R ) :-
  append(RowPfx,[Row|RowSfx],L),
  length(RowPfx,X) ,
  append(ColPfx,[_|ColSfx],Row) ,
  length(ColPfx,Y) ,
  append(ColPfx,[Z|ColSfx],RowNew) ,
  append(RowPfx,[RowNew|RowSfx],R).

% Get Piece From [X,Y] of matrix
getPiece(X,Y,Board,Element):-
     nth0(Y,Board,Elem),nth0(X,Elem,Element).
```

```
%Retrieves input from the player about his desired move
getPlayerMovement(Xi,Yi,Xf,Yf) :-
      print('Coordenada X da peça a mover : '), read(Xi),
      print('Coordenada Y da peça a mover : '), read(Yi),
      print('Coordenada X da casa destino : '), read(Xf),
      print('Coordenada Y da casa destino : '), read(Yf).

%Checks if the input coordinates are inside the board
checkOutOfBounds(Xi,Yi,Xf,Yf) :-
   (Xf > -1, Xi > -1,
   Xf < 8, Xi < 8,
   Yf > -1, Yi > -1,
   Yf < 8, Yi < 8).

% Checks if the input coordinates are different
positionsDifferent(Xi,Yi,Xf,Yf) :-
      (Xi \= Xf; Yi \= Yf).

%Check if for each player the pieces involved in the play are correct
checkCorrectPiece(1,Board,Xi,Yi,Xf,Yf):-getPiece(Xi,Yi,Board,InP),
getPiece(Xf,Yf,Board,FnP),
   ((InP == ' $ '; InP == ' + '),(FnP == ' & '; FnP == ' * '; FnP == '   '));
   (write('You can only play YOUR pieces and eat your oponent´s!'),nl,fail).

checkCorrectPiece(2,Board,Xi,Yi,Xf,Yf):-getPiece(Xi,Yi,Board,InP),
getPiece(Xf,Yf,Board,FnP),
   ((InP == ' & '; InP == ' * '),(FnP == ' $ '; FnP == ' + '; FnP == '   '));
   (write('You can only play YOUR pieces and eat your oponent´s!'),nl,fail).

checkMovement(Xi,Yi,Xf,Yf,Board):-
   getPiece(Xi,Yi,Board,P), checkPlay(Xi,Yi,Xf,Yf,P,D),
checkPath(Xi,Yi,Xf,Yf,P,D,Board,0).

%Checks the if a piece movement is valid either diagonaly, horizontaly or verticaly
checkPlay(Xi,Yi,Xf,Yf,' $ ',D):-
   (Xi == Xf, Yi \= Yf)->(D = 'V');
   (Xi \= Xf, Yi == Yf)->(D = 'H');
   (abs(Xi - Xf) =:= abs(Yi - Yf))->(D = 'D');
   (write('Soldiers can only move orthogonaly or diagonaly!'),nl,fail).
```

```
checkPlay(Xi,Yi,Xf,Yf,' & ',D):-
   (Xi == Xf, Yi \= Yf)->(D = 'V');
   (Xi \= Xf, Yi == Yf)->(D = 'H');
   (abs(Xi - Xf) =:= abs(Yi - Yf))->(D = 'D');
   (write('Soldiers can only move orthogonaly or diagonaly!'),nl,fail).

checkPlay(Xi,Yi,Xf,Yf,' + ',_):-
   ((Xi == Xf, (Yi == (Yf-1); Yi == (Yf+1)));
   (((Xi == (Xf-1); Xi == Xf+1)), Yi == Yf);
   ((abs(Xi - Xf) =:= abs(Yi - Yf)),abs(Xi - Xf) =:= 1));
   (write('Kings can only move orthogonaly or diagonaly to an adjacent cell!'),nl,fail).

checkPlay(Xi,Yi,Xf,Yf,' * ',_):-
   ((Xi == Xf, (Yi == (Yf-1); Yi == (Yf+1)));
   (((Xi == (Xf-1); Xi == Xf+1)), Yi == Yf);
   ((abs(Xi - Xf) =:= abs(Yi - Yf)),abs(Xi - Xf) =:= 1));
   (write('Kings can only move orthogonaly or diagonaly to an adjacent cell!'),nl,fail).

%Checks if there are no obstacles between the initial and final positions of the
play-vertically,horizontaly and diagonaly
checkPath(Xi,Yi,Xf,Yf,P,'V',Board,N):-
   (N > 1)->fail;
   Yi \= Yf,
   (Yi < Yf)->(
   Y2 is Yi + 1,
   getPiece(Xi,Y2,Board,Piece),
   ((Piece == '   ')->(N2 is N, checkPath(Xi,Y2,Xf,Yf,P,'V',Board,N));
   (Piece \= P)->(N2 is N + 1, checkPath(Xi,Y2,Xf,Yf,P,'V',Board,N2));
   fail)
   );
   (Yi > Yf)->(
   Y2 is Yi - 1,
   getPiece(Xi,Y2,Board,Piece),
   ((Piece == '   ')->(N2 is N, checkPath(Xi,Y2,Xf,Yf,P,'V',Board,N));
   (Piece \= P)->(N2 is N + 1, checkPath(Xi,Y2,Xf,Yf,P,'V',Board,N2));
   fail)
   );
   (write('Clear Path'),nl).

checkPath(Xi,Yi,Xf,Yf,P,'H',Board,N):-
```

```
    (N > 1)->fail;
    Xi \= Xf,
    (Xi < Xf)->(
    X2 is Xi + 1,
    getPiece(X2,Yi,Board,Piece),
    ((Piece == '   ')->(N2 is N, checkPath(X2,Yi,Xf,Yf,P,'H',Board,N));
    (Piece \= P)->(N2 is N + 1, checkPath(X2,Yi,Xf,Yf,P,'H',Board,N2));
    fail)
    );
    (Xi > Xf)->(
    X2 is Xi - 1,
    getPiece(X2,Yi,Board,Piece),
    ((Piece == '   ')->(N2 is N, checkPath(X2,Yi,Xf,Yf,P,'H',Board,N));
    (Piece \= P)->(N2 is N + 1, checkPath(X2,Yi,Xf,Yf,P,'H',Board,N2));
    fail)
    );
    (write('Clear Path'),nl).

checkPath(Xi,Yi,Xf,Yf,P,'D',Board,N):-
    (N > 1)->fail;
    Yi \= Yf,
    Xi \= Xf,
    (Yi < Yf,Xi < Xf)->(
    Y2 is Yi + 1,
    X2 is Xi + 1,
    getPiece(X2,Y2,Board,Piece),
    ((Piece == '   ')->(N2 is N, checkPath(X2,Y2,Xf,Yf,P,'D',Board,N));
    (Piece \= P)->(N2 is N + 1, checkPath(X2,Y2,Xf,Yf,P,'D',Board,N2));
    fail)
    );
    (Yi > Yf, Xi > Xf)->(
    Y2 is Yi - 1,
    X2 is Xi - 1,
    getPiece(X2,Y2,Board,Piece),
    ((Piece == '   ')->(N2 is N, checkPath(X2,Y2,Xf,Yf,P,'D',Board,N));
    (Piece \= P)->(N2 is N + 1, checkPath(X2,Y2,Xf,Yf,P,'D',Board,N2));
    fail)
    );
    (Yi > Yf, Xi < Xf)->(
    Y2 is Yi - 1,
    X2 is Xi + 1,
    getPiece(X2,Y2,Board,Piece),
```

```
   ((Piece == '   ')->(N2 is N, checkPath(X2,Y2,Xf,Yf,P,'D',Board,N));
   (Piece \= P)->(N2 is N + 1, checkPath(X2,Y2,Xf,Yf,P,'D',Board,N2));
   fail)
   );
   (Yi < Yf, Xi > Xf)->(
   Y2 is Yi + 1,
   X2 is Xi - 1,
   getPiece(X2,Y2,Board,Piece),
   ((Piece == '   ')->(N2 is N, checkPath(X2,Y2,Xf,Yf,P,'D',Board,N));
   (Piece \= P)->(N2 is N + 1, checkPath(X2,Y2,Xf,Yf,P,'D',Board,N2));
   fail)
   );
   (write('Clear Path'),nl).

%Loop that must check each and every step of a play before it proceeds
movePiece(Xi,Yi,Xf,Yf,Player,Board):-
        repeat,
        getPlayerMovement(Xi,Yi,Xf,Yf),
        positionsDifferent(Xi,Yi,Xf,Yf),
        checkOutOfBounds(Xi,Yi,Xf,Yf),
        checkCorrectPiece(Player,Board,Xi,Yi,Xf,Yf),
        checkMovement(Xi,Yi,Xf,Yf,Board).

movement(Xi,Yi,Xf,Yf,Player,Board):-movePiece(Xi,Yi,Xf,Yf,Player,Board),!.

%retrieve all the possible movements to list
listAllPossibleMoves(Board,Player,Value,List):-
   findall(X-Y-XF-YF-Value,
   allPossibleMoves(Board,Player,Value,X,Y,XF,YF),List).


%Tests if the Move can be done
testMoves(Board,Value,XIndex,YIndex,XFIndex,YFIndex):-
   write('\33\[2J'),
   checkMovement(XIndex,YIndex,XFIndex,YFIndex,Board),
   checkValue(XIndex,YIndex,XFIndex,YFIndex,Value).

%Generate and Test all movements
allPossibleMoves(Board,2,Value,X,Y,XF,YF):-
   (getPiece(XIndex,YIndex,Board,' & '); getPiece(XIndex,YIndex,Board,' * ')),
   (getPiece(XFIndex,YFIndex,Board,' $ ');getPiece(XFIndex,YFIndex,Board,' + ')),
   testMoves(Board,Value,XIndex,YIndex,XFIndex,YFIndex),
```

```
    X is XIndex,Y is YIndex, XF is XFIndex, YF is YFIndex.


allPossibleMoves(Board,1,Value,X,Y,XF,YF):-
    (getPiece(XIndex,YIndex,Board,' $ '); getPiece(XIndex,YIndex,Board,' + ')),
    (getPiece(XFIndex,YFIndex,Board,' & ');getPiece(XFIndex,YFIndex,Board,' * ')),
    testMoves(Board,Value,XIndex,YIndex,XFIndex,YFIndex),
    X is XIndex,Y is YIndex, XF is XFIndex, YF is YFIndex.

%Auxiliar function to get a play for the easy bot move (random)
getRandomMove(Board,Player,Value,X,Y,XF,YF):-
    listAllPossibleMoves(Board,Player,Value,List),
    random_member(X-Y-XF-YF-Value,List).

%Auxiliar function to get a play for the hard bot move (per value)
getBestMove(Board,Player,Value,X,Y,XF,YF):-
    listAllPossibleMoves(Board,Player,Value,List),
    chooseBestMove(List,X-Y-XF-YF-Value).

%retrieves the best play
chooseBestMove([PH|PT], BestMove):-
        maxList(PT, PH, BestMove).

%recursivly checks for the best value play
maxList([], Max, Max).
maxList([Xi-Yi-Xf-Yf-V|T], CXi-CYi-CXf-CYf-Cv, Max) :-
        ( V > Cv -> maxList(T, Xi-Yi-Xf-Yf-V, Max)
        ; maxList(T, CXi-CYi-CXf-CYf-Cv, Max) ).


%calculates the value of a certain play from (XY,YI) to (XF,YF)
checkValue(XIndex, YIndex, XFIndex,YFIndex,Value):-
    Xv is XFIndex - XIndex,
    Yv is YFIndex - YIndex,
    Value is round(sqrt(exp(Xv,2) + exp(Yv,2))).

% Changes the player each turn
changePlayer(1,Novo):-
        Novo is 2.
changePlayer(2,Novo):-
        Novo is 1.
```

```
% Writes whose turn it is to play
printPlayer(1,h):- write('Player 1 turn: '), nl.
printPlayer(2,h):- write('Player 2 turn: '), nl.

%Auxiliar predicate to print a play to make it easier for the user to understand
printPlay(P,Xf,Yf):-write('Player number '), write(P), write(' played to '),
write(Xf-Yf),nl.

% Auxiliar menu to allow human to choose bot difficulty
menuDificuldade(Dificuldade):-
        print('Dificuldade do Jogo'), nl,
        print('Fácil (f)'), nl,
        print('Dificil (d)'), nl,
        read(Dificuldade).

% Function to stop the game once we reach the 10th turn
gameDraw(Board):-printB(Board), write('The Game ended in a draw (reached 10
turns), this is the final board!').

%Checks if player 1 has won
game_over(Board):-
      \+ (getPiece(_,_,Board,' + ')),
      write('Game has ended! Player 2 wins!'), nl.

%Checks if player 2 has won
game_over(Board):-
      \+ (getPiece(_,_,Board,' * ')),
      write('Game has ended! Player 1 wins!'), nl.
```