Martian Chess

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo 4:
João Chaves-up201406225
Rui Araújo-up201403263

Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

12 de Novembro de 2016

Índice

Resumo

O principal objetivo do projeto é a implementação das regras do jogo Martian Chess utilizando a linguagem PROLOG. Na especificação do projeto foi também pedido que se desenvolvesse uma interface em linha de texto para o mesmo jogo, por forma a ser jogado entre dois jogadores, entre um jogador e o computador em dois níveis, fácil e difícil, e por fim computador contra computador.

A principal dificuldade do trabalho foi a implementação de uma inteligência que conseguisse calcular a melhor jogada possível num turno tendo em conta que seria possível fazer diferentes movimentos com 3 peças diferentes, mas conseguimos resolver tudo a que nos propusemos.

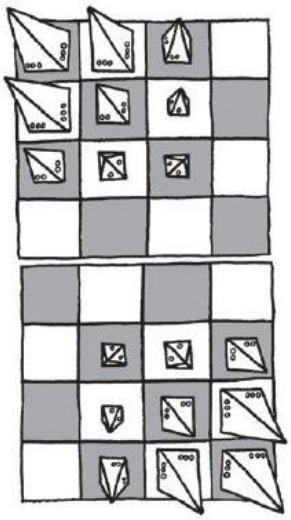
Introdução

Escolhemos este jogo porque é parecido com o Xadrez que ambos conhecemos bem e gostamos e com o propósito de aprender uma nova linguagem de programação.

O relatório está dividido em quatro partes, a explicação do jogo e das suas regras seguido da lógica do jogo, que por sua vez apresenta as representações de estado do jogo, o modo de visualização do tabuleiro em modo texto, lista de jogadas válidas, o modo de efectuar jogadas e de avaliação do tabuleiro para comparar as diferentes jogadas possíveis, a condição de terminação do jogo e a jogada do computador considerando a dificuldade. De seguida a interface com o utilizador em modo de texto e, por fim, a conclusão do projecto.

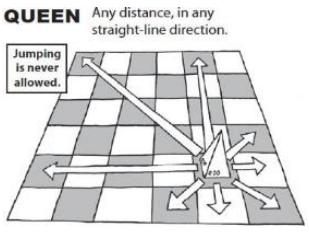
Martian Chess

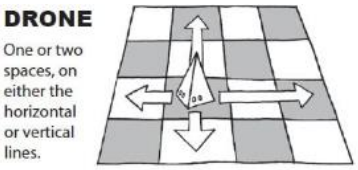
Martian Chess é um jogo abstracto de estratégia para 2 ou 4 jogadores inventado por Andrew Looney. É jogado com peças em forma de pirâmides num tabuleiro de xadrez.

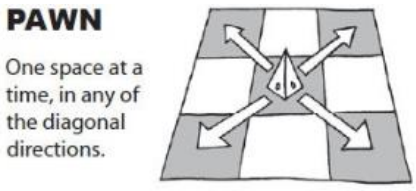


Regras:

- Cada jogador começa com 9 peças: 3 pequenas (pawns), 3 médias (drones) e 3 grandes (queens). A cor das peças é irrelevante para este jogo.
- Os jogadores colocam as suas peças nos cantos do tabuleiro, num jogo entre 2 jogadores, apenas metade do tabuleiro é usado.
- São os jogadores que decidem quem começa e as jogadas alternam.
- O tabuleiro é dividido em duas partes (no caso de 2 jogadores) ou em quatro partes (4 jogadores). Cada jogador apenas controla as peças que se encontram no seu território (lado do tabuleiro).
- Pawns movem-se um espaço diagonalmente, em todas as direcções.
- Drones movem-se um ou dois espaços na horizontal ou vertical.
- Queens movem-se em qualquer distância e em qualquer direcção.







- Cada espaço não pode conter mais do que uma peça
- Uma peça é capturada quando uma peça inimiga se move para o espaço onde esta se encontra.
- Uma peça capturada é eliminada do tabuleiro e no final do jogo será contabilizada para contagem do resultado e determinação do vencedor.
- As peças são possuídas de acordo com o território onde se encontram, logo um jogador do qual a sua peça é capturada ganha controlo da peça que a capturou.
- Promoções: Se um jogador não tiver Queens pode criar uma movendo um Drone para a posição de um Pawn (ou vice versa) e juntando-os. Da mesma maneira podem ser usados dois Pawns para formar um Drone.
- Um jogador não pode reverter a jogada anterior do adversário (exemplo: não pode mover uma peça para lá da linha de divisão dos territórios para o espaço de partida, na jogada seguinte).
- Pontuação:
    - Pawn: 1 ponto.
    - Drone: 2 pontos.
    - Queen: 3pontos.
- O jogo termina quando um jogador fica sem peças (o seu território fica vazio)
- O jogador com maior pontuação ganha a partida.
- Num jogo de 4 jogadores, em que é jogado por duas equipas de 2 jogadores as pontuações dos 2 jogadores é somada.

Representação do Estado do Jogo

O tabuleiro de jogo é representado através de uma lista de listas em que “s” significa espaço vazio, “q “significa Queen, “d “significa Drone e” p” significa Pawn.

Representação em Prolog do estado de jogo inicial:

```
board([ [q,q,d,s],
        [q,d,p,s],
        [d,p,p,s],
        [s,s,s,s],
        [s,s,s,s],
        [s,p,p,d],
        [s,p,d,q],
        [s,d,q,q]]).
```

Posição intermédia:

```
board([ [q,s,q,s],
        [d,d,s,s],
        [p,s,p,s],
        [s,s,p,s],
        [s,q,d,s],
        [s,p,s,s],
        [s,s,d,s],
        [s,s,q,q]]).
```

Posição final:

```
board([ [s,s,s,s],
        [s,s,s,s],
        [s,s,s,s],
        [s,s,s,s],
        [s,p,s,s],
        [s,s,d,s],
        [s,s,s,q]]).
```

Visualização do Tabuleiro

Para a visualização em modo de texto é usado o predicado **display_full_board()** que chama **display_board_letter()** para imprimir as letras que correspondem a cada coluna do tabuleiro. **display_board()** imprime a numeração de cada uma das linhas através de **display_line_number(),** bem como o conteúdo do tabuleiro através do predicado **display_line()** e escreve, de seguida, os caracteres para construir os espaços do tabuleiro.

```
display_full_board(B, X, Y):-
display_board_letter,
display_board(B,X,Y, 1).

%%%%%%%%%%%%%%Display board predicates%%%%%%%%%%%%%%%%%%%%%%%%%%
display_board_letter:-
    write('    a   b   c   d'),
    nl,
    write('   ---------------- '),
    nl.

display_board([L1|Ls],X,Y,N):-

    display_line_number(N),
    N1 is N+1,
    write('|'),
    display_line(L1),
    nl,
    ( N=4 -> write('  =================');write('   ----------------') ),
    nl,
    display_board(Ls,X,Y,N1).

display_board([],_,_,_).
```

```
display_line_number(N):-
    write(N),
    write(' ').

display_line([E|Es]):-
    translate(E,V),
    write(' '),
    write(V),
    write(' |'),
    display_line(Es).

display_line([]).


translate(s,' ').
translate(p,'*').
translate(d, 'x').
translate(q, 'X').
```

O predicado **display_line()** mostra a linha, uma a uma fazendo também a tradução dos símbolos utilizados para representação das peças em prolog, para símbolos mais representativos das peças do tabuleiro (**'*'** representa o pawn, ' **x'** representa o drone e ' **X'** a queen).

```
    a   b   c   d
  ----------------
1 | X | X | x |   |
  ----------------
2 | X | x | * |   |
  ----------------
3 | x | * | * |   |
  ----------------
4 |   |   |   |   |
  =================
5 |   |   |   |   |
  ----------------
6 |   | * | * | x |
  ----------------
7 |   | * | x | X |
  ----------------
8 |   | x | X | X |
  ----------------
```

```
    a   b   c   d
  ----------------
1 |   |   |   |   |
  ----------------
2 |   |   |   | * |
  ----------------
3 |   |   |   |   |
  ----------------
4 |   |   | X |   |
  =================
5 |   |   |   |   |
  ----------------
6 |   |   |   |   |
  ----------------
7 |   |   |   |   |
  ----------------
8 |   |   |   |   |
  ----------------
```

Lista de jogadas válidas

Para verificar se uma jogada é válida, dadas as coordenadas da peça a mover e das coordenadas para a nova célula do tabuleiro criaram-se diferentes predicados (pelo facto de existirem 3 peças diferentes, com movimentos possíveis também diferentes) que verificam se cada uma pode, ou não, mexer-se que são **pawn_can_move(), drone_can_move()** e **queen_can_move(),** estes recebem como argumentos o tabuleiro (B), as coordenadas da peça a mover(L,C), as novas coordenadas(Nl, Nc), o nº da jogada(J), ,o elemento que se encontra nas novas coordenadas (Elem2) e o novo elemento a colocar nas novas coordenadas (NewElem).

```
%%%%%%%%%%%%%%%%Chceks the move of the pawn%%%%%%%%%%%%%%%%
pawn_can_move(B,L,C,Nl,Nc, J, Elem2, NewElem):-
  (check_jogada(B, L, C, Nl, Nc, J, Elem2, NewElem) ->
    DL is abs(Nl-L),
    DC is abs(Nc-C),
    (DC=1,DL=1 -> nl ; (write('Jogada invalida_pawn\n') ,false)); write('fail check jogada'), false).


%%%%%%%%%%%%%%%%Chceks the move of the drone%%%%%%%%%%%%%%%%
drone_can_move(B,L,C,Nl,Nc, J, Elem2, NewElem):-
  (check_jogada(B, L, C, Nl, Nc, J, Elem2, NewElem) ->
    AbsDL is abs(Nl-L),
    AbsDC is abs(Nc-C),
    DL is Nl-L,
    DC is Nc-C,
    (AbsDC=0,AbsDL=0 ->  (write('Jogada invalida_drone\n') ,false );
    ((AbsDL=1 ;AbsDL=2),AbsDC=0 );((AbsDC=1;AbsDC=2),AbsDL=0) -> nl;  write('Jogada invalida_drone\n'),false),

    (
        DL = 0, DC < 0 -> check_path_col(B, Nl, C, Nc, -1);
        DL = 0, DC > 0 -> check_path_col(B, Nl, C, Nc, 1);

        DC = 0, DL < 0 -> check_path_line(B, L, Nl, Nc, -1);
        DC = 0, DL > 0 -> check_path_line(B, L, Nl, Nc, 1);
        nl
    ); false).
```

Para testar se os movimentos do pawn é chamado o predicado **check_jogada()** com os argumentos tabuleiro, linha, coluna, nova linha, nova coluna, numero da jogada elemento nas novas coordenadas e o elemento a colocar nas novas cordenadas**,** para testar se a peça escolhida corresponde ao jogador que a esta a tentar mover e se o pode mover.

Em relação à verificação do movimento do drone e como este se pode mover mais do que uma posição é necessário testar também se o caminho até à nova célula não contem peças (porque este não pode saltar) para isso são chamados os predicados **check_path_line()** no caso do movimento ser vertical e **check_path_col()** se o movimento for na horizontal, com os argumentos tabuleiro, linha, coluna, nova linha, nova coluna e 1 ou -1 se o movimento for horizontal ou vertical, respectivamente.

Para a avaliação do movimento da queen são utilizados os predicados referidos anteriormente acrescidos do **check_line_col()** pelo facto do movimento poder ser também na diagonal, os argumentos são os mesmos das anteriores com o a adição de um 1 ou -1 dependendo do movimento ser na diagonal para ascendente ou descendente, horizontal ou vertical.

```
%%%%%%%%%%%%%%%Chceks the move of the queen%%%%%%%%%%%%%%%%
queen_can_move(B, L, C, Nl, Nc, J, Elem2, NewElem):-
  (check_jogada(B, L, C, Nl, Nc, J, Elem2, NewElem) ->
    DL is Nl-L,
    DC is Nc-C,
    (
        DC=0,DL=0 -> write('Jogada invalida_queen\n') , false;

        DL = 0, DC < 0 -> check_path_col(B, Nl, C, Nc, -1);
        DL = 0, DC > 0 -> check_path_col(B, Nl, C, Nc, 1);

        DC = 0, DL < 0 -> check_path_line(B, L, Nl, Nc, -1);
        DC = 0, DL > 0 -> check_path_line(B, L, Nl, Nc, 1);

    AbsL is abs(DL),
    AbsC is abs(DC),

    (AbsL \= AbsC -> write('Jogada invalida abs_queen\n'), false; nl),

    (
        DL < 0, DC < 0 -> check_path_line_col(B, L, C, Nl, Nc, -1, -1).
        DL < 0, DC > 0 -> check_path_line_col(B, L, C, Nl, Nc, -1, 1);
        DL > 0, DC < 0 -> check_path_line_col(B, L, C, Nl, Nc, 1, -1);
        DL > 0, DC > 0 -> check_path_line_col(B, L, C, Nl, Nc, 1, 1);
        nl
    )
  );
  false).
```

Execução de jogadas

Para a execução de jogadas é invocado o predicado **move_piece()** que recebe como argumentos o tabuleiro a linha, a coluna , a nova linha, a nova coluna, o novo tabuleiro, o numero da jogada, o antigo score e o novo score do jogador 1 e do jogador 2 respetivamente. Depois de saber qual é a peça naquele espaço (através de **getelem()**) testa se esta se pode mover para as coordenadas pretendidas, em caso afirmativo começa por alterar o espaço original por 's' (espaço), através do predicado **replace()**, descobre a peça na célula para onde pretende mover, para que saiba se está a eliminar alguma peça, actualiza o score com **update_score()** e altera o espaço final pela peça a mover.

```
%%%%%%%%%%%%%%%The player tries to make a new move%%%%%%%%%%%%%%%%%
move_piece(B,L,C,Nl,Nc,Nr, J, Os1, Ns1, Os2, Ns2):-
    getelem(B,L,C,Elem),
  write('prev: '), write(Elem),
  nl,
  getelem(B, Nl, Nc, Elem21),
  write('next: '), write(Elem21),
  nl,
    (
        Elem = 's' -> write('peca invalida\n'), false;
        Elem = 'p' -> F is 0;
        Elem = 'd' -> F is 1;
        Elem = 'q' -> F is 2
    ),

        Li is L - 1,
        Ci is C - 1,

        NLi is Nl - 1,
        NCi is Nc - 1,

    (
        F = 0 -> (pawn_can_move(B, L, C, Nl, Nc, J, Elem3, NewElem) -> replace(B, Li, Ci, 's', N), getelem(B, Nl, Nc, Elem2),
        update_score(Nl, Elem2, Os1, Ns1, Os2, Ns2, J), replace(N,NLi, NCi,NewElem, Nr); false);
        F = 1 -> (drone_can_move(B, L, C, Nl, Nc, J, Elem3, NewElem) -> replace(B, Li, Ci, 's', N),
        getelem(B, Nl, Nc, Elem2), update_score(Nl, Elem2, Os1, Ns1, Os2, Ns2, J), replace(N,NLi, NCi,NewElem, Nr); false);
        F = 2 -> (queen_can_move(B, L, C, Nl, Nc, J, Elem3, NewElem) -> replace(B, Li, Ci, 's', N),
        getelem(B, Nl, Nc, Elem2), update_score(Nl, Elem2, Os1, Ns1, Os2, Ns2, J), replace(N,NLi, NCi,NewElem, Nr); false)
    ).
```

Avaliação do Tabuleiro

Para processar jogadas é chamado o predicado **process_pieces**(), com os argumentos tabuleiro, linha coluna, numero da jogada, novo tabuleiro e score a antigo score dos dois jogadores que tem como argumentos o tabuleiro o numero da jogada o novo tabuleiro e o actual e novo score dos dois jogadores. Este predicado percorre o território do jogador que se encontra a jogar à procura de jogadas possíveis.

```
process_pieces(B, J, Nb, S1, Ns1, S2, Ns2):-
  (is_par(J) -> process_pieces_top(B, 1, 1, J, Nb, S1, Ns1, S2, Ns2); process_pieces_bot(B, 5, 1, J, Nb, S1, Ns1, S2, Ns2)).

process_pieces_top(B, X, Y, J, Nb, S1, Ns1, S2, Ns2):-
  (X = 5 -> false;
    getelem(B, X, Y, Elem),
    (Elem \= 's', find_next_board_bot(B, X, Y, 5, 1, J, Nb, S1, Ns1, S2, Ns2);
      (Y = 4 -> Y1 is 1, X1 is X + 1; Y1 is Y + 1, X1 is X),
      process_pieces_top(B, X1, Y1, J, Nb, S1, Ns1, S2, Ns2))).

process_pieces_bot(B, X, Y, J, Nb, S1, Ns1, S2, Ns2):-
  (X = 9 -> false;
    getelem(B, X, Y, Elem),
    (Elem \= 's', find_next_board_top(B, X, Y, 1, 1, J, Nb, S1, Ns1, S2, Ns2);
      (Y = 4 -> Y1 is 1, X1 is X + 1; Y1 is Y + 1, X1 is X),
      process_pieces_bot(B, X1, Y1, J, Nb, S1, Ns1, S2, Ns2))).
```

Final do Jogo

Como referido anteriormente o jogo termina quando um dos "territórios" do tabuleiro fica vazio, mesmo que possa não ser o proprietário dessa metade do tabuleiro a ganhar o jogo, já que a determinação do vencedor é feita pelo score de cada um, que é aumentado eliminando peças adversarias. Consequentemente os predicados necessários para verificar o fim do jogo vão testar se alguma das metades do tabuleiro está vazia, **end_game_p1()** para verificar o território do jogador 1 e **end_game_p2()** para o jogador 2, estes predicados são chamados ao fim de cada ciclo de jogo, quando estas retornam false o jogo prossegue e os seus argumentos são apenas o tabuleiro e um contador.

```
%%%%%%%%%%%%%%%Check if each side of the board is empty%%%%%%%%%%%%%%%%
end_game_p1(B, C):-
    C = 5 -> true;
    nth1(C, B, Elem),
    (
    Elem \= ['s', 's', 's', 's'] -> false;
    C1 is C + 1,
    end_game_p1(B, C1)).

end_game_p2(B, C):-
    C = 9 -> true;
    nth1(C, B, Elem),
    (
        Elem \= ['s', 's', 's', 's'] -> false;

    C1 is C + 1,
    end_game_p2(B, C1)).
```

Jogada do Computador

Modo Fácil

A inteligência **Fácil** joga aleatoriamente través do predicado **rand_play(),**que recebe o tabuleiro, o novo tabuleiro, o numero da jogada, o score e novo score dos jogadores 1 e 2 e gera um movimento aleatório para uma peça do jogador que está a jogar, para isso chama predicados que fazem uma procura no tabuleiro desse mesmo jogador **(process_pieces_rand()**).

Os detalhes deste movimento são depois passados ao predicado responsável por realizar o movimento (**move_piece()**).

```
%%%%%%%%%%%%%%%Goes through the board and finds a possible play%%%%%%%%%%%%%%%%
rand_play(B, Nb, J, S1, Ns1, S2, Ns2):-
  nl, write('random'), nl,
  process_pieces_rand(B, J, Nb, S1, Ns1, S2, Ns2).

process_pieces_rand(B, J, Nb, S1, Ns1, S2, Ns2):-
  (is_par(J) -> process_pieces_top_rand(B, 1, 1, J, Nb, S1, Ns1, S2, Ns2); process_pieces_bot_rand(B, 5, 1, J, Nb, S1, Ns1, S2, Ns2)).
```

Modo Difícil

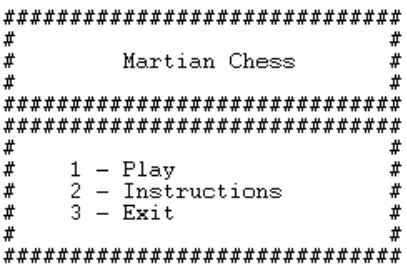
Neste modo de jogo é chamado o predicado **best_play()** que avalia para cada peça do território do tabuleiro do jogador que se encontra a jogar e percorre o território do adversário à procura de oportunidades para eliminar uma peça adversária, no caso de não haver faz um movimento aleatório.

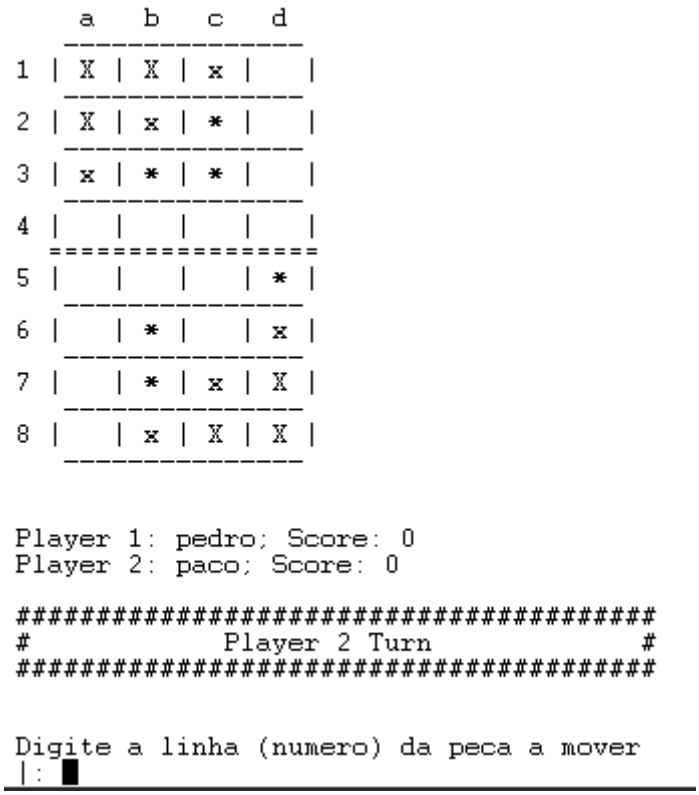
```
%%%%%%%%%%%%%%%Generates a play in which the PC captures a piece and if it isn't possible, makes a random move%%%%%%%%%%%%%%%%%
best_play(B, Nb, J, S1, Ns1, S2, Ns2):-
  process_pieces(B, J, Nb, S1, Ns1, S2, Ns2).

process_pieces(B, J, Nb, S1, Ns1, S2, Ns2):-
  (is_par(J) -> process_pieces_top(B, 1, 1, J, Nb, S1, Ns1, S2, Ns2);
                process_pieces_bot(B, 5, 1, J, Nb, S1, Ns1, S2, Ns2)).
```

Interface com utilizador

Inicializar o jogo requer apenas correr o predicado **startGame**, a partir daqui encontramos o menu principal. Este menu conte 3 opções, jogar, ler as instruções do jogo, sair. O menu Jogar apresenta 3 escolhas: Jogar (entre duas pessoas), Jogar contra computador (com hipótese de modo Fácil ou Difícil) e Computador contra Computador.





Quando um dos territórios do tabuleiro ficar vazio, o jogo termina, anunciando qual o jogador venceu, e depois mostrando de novo o tabuleiro, agora no seu estado final.

```
    a   b   c   d
   ---------------
1 |   |   |   |   |
   ---------------
2 |   |   |   | * |
   ---------------
3 |   |   |   |   |
   ---------------
4 |   |   | X |   |
  =================
5 |   |   |   |   |
   ---------------
6 |   |   |   |   |
   ---------------
7 |   |   |   |   |
   ---------------
8 |   |   |   |   |
   ---------------

##########################################
#                                        #
#              Game Over!                #
#                                        #
##########################################


##########################################
#                                        #
#           Player 2 Wins!!              #
#                                        #
##########################################
```

Conclusão

A realização deste projeto contribuiu irrefutavelmente para a consolidação dos assuntos abordados nas aulas teóricas e práticas da cadeira e, como tal, permitiram obter um conhecimento mais profundo de como trabalhar com a linguagem PROLOG e do seu poder enquanto linguagem.

Completamos todos os objectivos propostos, já que todos os modos foram feitos, embora o modo de Inteligencia Artificial de modo Dificil pudesse ser melhorado.

Bibliografia

Sterling, Leon S.; Shapiro, Ehud Y. - **The Art of Prolog**: Advanced Programming Techniques

http://www.looneylabs.com/rules/martian-chess

http://www.swi-prolog.org/

Anexo

Em anexo seguem os ficheiros auxiliar.pl, board.pl,cli.pl,computer.pl, play.pl e main.pl
No Sicstus deve ser consultado o ficheiro main.pl e para correr o jogo deve ser usada a instrução: startGame.