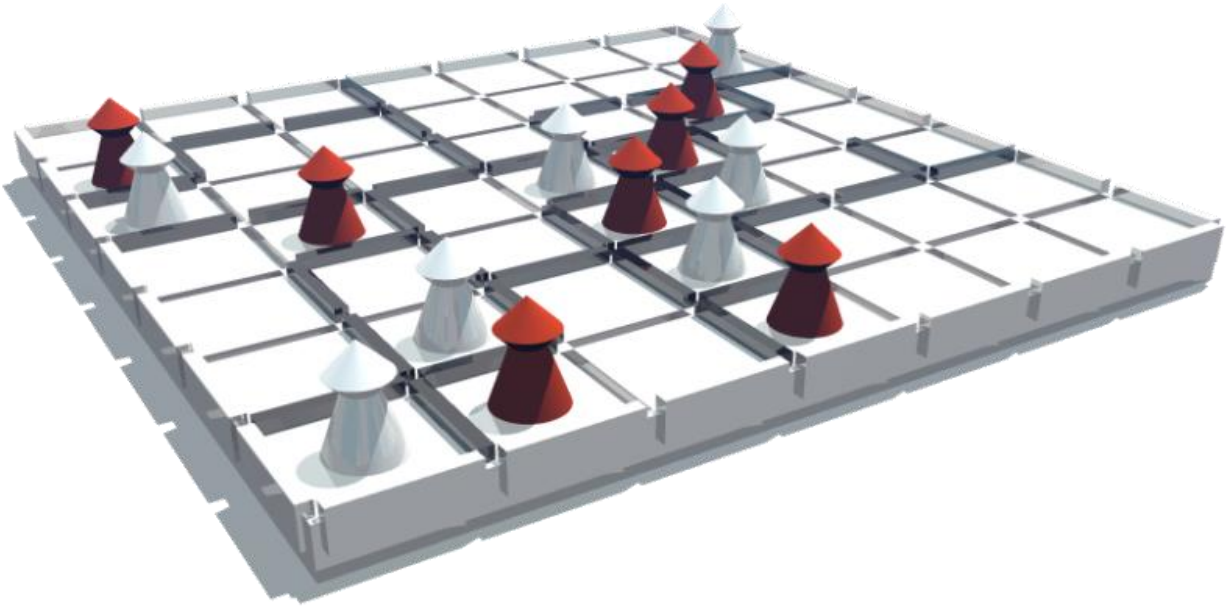
Programação em Lógica



FINES_4

3MIEIC02:

Mário André Macedo Ferreira – 201208066

Pedro Miguel Martins de Lemos da Cunha Faria – 201206054

Professor Henrique Daniel de Avelar Lopes Cardoso

Professor Carlos Manuel Milheiro de Oliveira Pinto Soares

Professor Daniel Augusto Gama de Castro Silva

Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal

11 de Novembro de 2014

RESUMO

O trabalho desenvolvido foi um jogo de tabuleiro chamado *Fines*, que envolvia dois jogadores e cujo objectivo era conseguir obter uma maior área do tabuleiro, número de quadrículas. Os jogadores iniciam o jogo com uma peça cada um e no seu turno podem adicionar uma peça (até um máximo de 7), mover uma peça e adicionar uma cerca, adicionar apenas uma cerca ou passar a vez. Para além disto outro objectivo era desenvolver uma inteligência artificial, permitindo um jogador real jogar contra o computador.

Para a implementação da lógica do jogo foram usadas principalmente listas com uma estrutura [cabeça | cauda], em que a 'cabeça' é o primeiro elemento da lista e a 'cauda' são os elementos restantes, sendo que o tratamento destas estruturas é feito recorrendo à recursividade.

Desenvolveu-se ainda uma interface com o utilizador o mais simplista possível para que este tivesse que introduzir a menor quantidade possível de informação na consola, diminuindo assim a quantidade de erros gerados pelo utilizador.

No final nem todos os objectivos foram cumpridos que se deveu à carga de trabalho de todas as unidades curriculares a que os membros do grupo estão inscritos.

ÍNDICE

CONTEÚDO

1-INTRODUÇÃO

O desenvolvimento do jogo de tabuleiro *Fines* em *Prolog* surgiu no âmbito da unidade curricular Programação em Lógica no seguimento da matéria lecionada nas aulas, principalmente listas. Estas listas em *Prolog*, à semelhança de muitas outras linguagens de programação, tem uma estrutura do tipo [cabeça | cauda], onde a 'cabeça' é o primeiro elemento da lista e a 'cauda' é o resto da lista e o tratamento destas listas é feito tendo em conta a sua estrutura, isto é, trata-se a 'cabeça' e depois a 'cauda' é novamente dividida nesta estrutura, ou seja, o primeiro elemento da 'cauda' passa a ser a 'cabeça' da lista e o resto a 'cauda'.

No caso em concreto deste trabalho, as listas foram utilizadas como forma de representar o tabuleiro de jogo, e a informação sobre cada jogador, de forma a facilitar a implementação de toda a lógica do jogo: movimento das peças, adicionar peças ao tabuleiro, verificar se algum dos jogadores terminou, etc.

Relativamente ao relatório, este divide-se em cinco tópicos diferentes. O primeiro tópico fala um pouco sobre o jogo, isto é, sobre quando surgiu e regras do jogo. No segundo tópico é discutida a implementação da lógica do jogo, nomeadamente, representação do tabuleiro, movimento das peças e condições de vitória. De seguida, fala-se um pouco sobre a estratégia utilizada na interface com o utilizador. No quinto tópico faz-se uma conclusão acerca dos objectivos do trabalho e sobre as dificuldades encontradas na realização do mesmo. Por fim é feita referência à consulta utilizada na realização do jogo e do relatório.

2-O JOGO *FINES*

O jogo "Fines", também conhecido como "Fendo", é um jogo de tabuleiro publicado este ano, 2014, o que faz com que não tenha muita história acerca do mesmo. *Fines* consiste num tabuleiro 7x7, no qual 2 jogadores tentam conquistar o máximo terreno possível, neste caso as próprias quadrículas do tabuleiro. Para que essa conquista seja feita, as quadriculas a conquistar têm de estar delimitadas por todos os lados com cercas e ter uma única peça no seu interior.

O jogo começa com uma peça de cada jogador no tabuleiro e apenas termina quando não existirem quaisquer zonas aberta ou um dos jogadores não se conseguir mexer. A cerca é utilizada pelos jogadores para delimitarem uma determinada área do tabuleiro. No final do jogo cada jogador conta as quadriculas que tem em seu "domínio" e vence o jogador com um maior número de quadrículas.

Cada jogador no seu turno tem três opções. Na primeira opção o jogador poderá mover uma das suas peças à escolha posicionadas no tabuleiro, segundo a horizontal e a vertical, não podendo passar por cima de cercas ou de outras peças. Caso o jogador pretenda, poderá alterar a orientação do movimento da peça, através de uma curva perpendicular à direita ou à esquerda, sendo de ressalvar que esta mudança de direção somente pode ser feita uma vez por jogada. Este movimento da peça não é limitado pelo número de quadrículas. No final do movimento da peça, o jogador é obrigado a adicionar uma cerca numa das posições adjacentes à peça movida onde ainda não haja uma cerca. Caso o jogador pretenda a peça poderá manter se na mesma posição e adicionar apenas a cerca.

Uma segunda opção é o jogador decidir adicionar uma peça, sendo a principal restrição só poderem existir um máximo de sete peças de cada jogador no tabuleiro. A peça poderá ser adicionada em qualquer posição desde que haja pelo menos um caminho possível entre a peça adicionada e uma peça existente no tabuleiro, ou um caminho possível entre uma peça existente e a peça a adicionar.

Em último caso caso, o jogador pode ainda não fazer nada na sua vez e passar a vez ao outro jogador.

3-LÓGICA DO JOGO

Neste tópico vai ser abordada a estrutura e a estratégia utilizadas na implementação de toda a lógica do jogo dividindo por tópicos de forma a ser mais fácil de explicar e perceber.

3.1- Representação do Estado de Jogo

Ao longo de todo o jogo tanto o tabuleiro como os jogadores são representados por listas. No caso do tabuleiro uma lista de listas, em que cada uma destas representa uma linha. Os jogadores são representados por uma lista em que o primeiro elemento é o seu nome, o segundo é o número de peças no tabuleiro e o terceiro o total da área que detém

<u>Fase Inicial:</u>

Nesta fase, o tabuleiro encontra-se completamente sem cercas, exluindo os limites do tabuleiro, e apenas com 2 peças, uma de cada jogador. Uma das peças é colocada na 4ª linha e 1ª coluna e a outra peça, na 4ª linha e na última coluna, isto é a 7ª coluna. Ao jogador com a peça branca (representada por um '1') é garantida a primeira jogada.

Quanto aos jogadores, nesta fase inicial, depois de lhes ter sido perguntado qual o nome que usariam durante o jogo a lista que os representa tem a estrutura [nome, 1, 0].

```
initialBoard([
            [3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3], %fences line
            [4, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 4], % pieces
            [4, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 4], %fences line
            [4, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 4], % pieces
            [4, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 4], %fences line
            [4, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 4], % pieces
            [4, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 4], %fences line
            [4, 1, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 2, 4], % pieces
            [4, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 4], %fences line
            [4, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 4], % pieces
            [4, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 4], %fences line
            [4, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 4], % pieces
            [4, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 4], %fences line
            [4, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 4], % pieces
            [4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4]  %fences line
            ]).
```

Fase Intermédia:

Durante esta fase, cada jogador faz a sua jogada à vez com o objetivo de conquistar a maior área possível do tabuleiro, relembrando que em cada jogada o jogador pode adicionar uma peça, acrescentar uma cerca ou mover-se e acrescentar uma cerca. O exemplo a seguir é o de um possível estado intermédio, sendo que nesse caso a a lista que representa o jogador 1 (peças brancas) seria algo do género [nome1, 5, 6] e a do jogador 2 (peças vermelhas) seria [nome2, 4, 6].

```
intermediateBoard([
            [3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3], %fences line
            [4, 0, 3, 0, 3, 1, 3, 0, 4, 0, 4, 0, 3, 0, 4], % pieces
            [4, 5, 6, 5, 6, 3, 6, 5, 6, 3, 6, 3, 6, 3, 4], %fences line
            [4, 0, 3, 0, 4, 0, 4, 0, 3, 0, 4, 2, 3, 0, 4], % pieces
            [4, 3, 6, 3, 6, 5, 6, 3, 6, 3, 6, 5, 6, 5, 4], %fences line
            [4, 0, 3, 0, 4, 0, 3, 0, 3, 0, 3, 0, 3, 0, 4], % pieces
            [4, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 4], %fences line
            [4, 0, 3, 0, 4, 0, 3, 2, 3, 0, 3, 1, 4, 0, 4], % pieces
            [4, 5, 6, 3, 6, 5, 6, 3, 6, 3, 6, 5, 6, 3, 4], %fences line
            [4, 0, 3, 1, 3, 2, 4, 0, 3, 0, 3, 0, 3, 0, 4], % pieces
            [4, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 6, 3, 4], %fences line
            [4, 2, 3, 0, 3, 0, 3, 1, 4, 0, 3, 0, 3, 0, 4], % pieces
            [4, 5, 6, 3, 6, 5, 6, 5, 6, 5, 6, 3, 6, 3, 4], %fences line
            [4, 0, 3, 0, 4, 0, 3, 2, 4, 1, 4, 0, 3, 0, 4], % pieces
            [4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4]  %fences line
            ]).
```

Fase Final:

O fim do jogo ocorre assim que não existam mais áreas abertas, isto é, todas as áreas têm de estar delimitadas com exclusivamente uma peça no seu interior. Ganha o jogo o jogador que tiver um maior número de quadrículas conquistadas. Seguindo o exemplo seguinte o jogador 1 seria representado por [nome1, 7, 25] e a do jogador 2 seria [nome2, 7, 24]. Neste caso o jogador 1 saiu vitorioso.

```
finalBoard([
            [3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3], %fences line
            [4, 0, 3, 0, 3, 1, 3, 0, 3, 0, 4, 0, 3, 0, 4], % pieces
            [4, 5, 6, 5, 6, 3, 6, 5, 6, 5, 6, 3, 6, 3, 4], %fences line
            [4, 0, 3, 0, 4, 0, 4, 0, 3, 0, 4, 2, 3, 0, 4], % pieces
            [4, 5, 6, 3, 6, 5, 6, 3, 6, 3, 6, 5, 6, 3, 4], %fences line
            [4, 0, 4, 1, 4, 0, 3, 0, 3, 0, 4, 0, 4, 0, 4], % pieces
            [4, 3, 6, 5, 6, 3, 6, 3, 6, 5, 6, 3, 6, 5, 4], %fences line
            [4, 0, 3, 2, 4, 0, 3, 2, 4, 0, 3, 1, 4, 0, 4], % pieces
            [4, 5, 6, 5, 6, 5, 6, 3, 6, 5, 6, 5, 6, 3, 4], %fences line
            [4, 0, 4, 0, 3, 2, 4, 0, 4, 1, 3, 0, 3, 0, 4], % pieces
            [4, 3, 6, 5, 6, 3, 6, 5, 6, 5, 6, 3, 6, 3, 4], %fences line
            [4, 2, 4, 1, 4, 0, 4, 1, 4, 2, 4, 0, 3, 0, 4], % pieces
            [4, 5, 6, 3, 6, 5, 6, 5, 6, 5, 6, 3, 6, 5, 4], %fences line
            [4, 0, 3, 0, 4, 0, 3, 2, 4, 1, 4, 0, 3, 0, 4], % pieces
            [4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4]  %fences line
            ]).
```

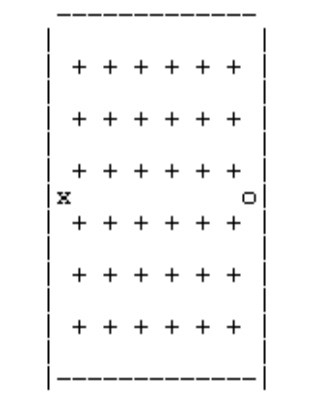
3.2-Visualização do Tabuleiro

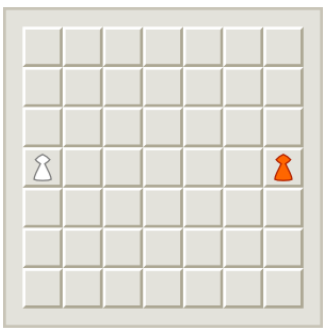
Uma vez que a interface do jogo foi feita em consola, sem recurso a nenhuma interface gráfica, - foram definidos alguns predicados para a representação do tabuleiro em diferentes fases do jogo. Essa visualização será efetuada de acordo com a seguinte correspondência, em que o caracter ' ', representa uma casa vazia, os caracteres 'x' e 'o', representam as peças dos 2 jogadores, enquanto que os caracteres restante representam as cercas que poderão ser colocadas entre as casas de jogo. Foram também implementados predicados que possibilitaram o desenho do jogo em modo de texto.

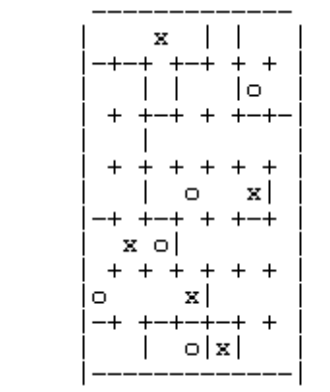
```
translateChar(0,' '). %empty space
translateChar(1,'x').
translateChar(2,'o').
translateChar(3,' '). %no fence
translateChar(4,'|').
translateChar(5,'-').
translateChar(6, '+').
```

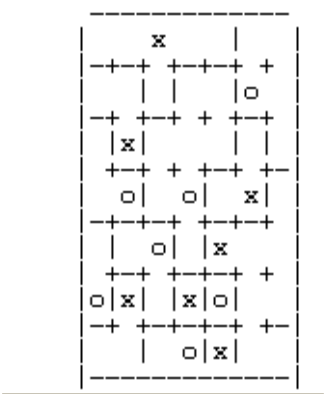
```
printBoard_line([H|T]):-
  translateChar(H,X),
  write(X),
  printBoard_line(T).

printBoard([H|T]):-
  printBoard_line(H),
  nl,
  printBoard(T).

printBoard([]).
printBoard_line([]).
```

Deste modo, no final de cada jogada o jogador poderá ver o tabuleiro de jogo idêntico a este que se encontra abaixo. Fases: inicial, intermédia e final respetivamente.









3.3- Lista de Jogadas Válidas

No seu turno, cada jogador poderá optar por uma entre as 3 jogadas válidas: o jogador poderá movimentar uma peça e adicionar uma cerca, adicionar uma peça ou passar a vez.

Caso o jogador queira mover uma peça e adicionar uma cerca, ele terá de introduzir a posição da peça que pretende mover, assim como a posição para a qual quer que o movimento seja feito. Após isto, escolhe a posição da cerca tendo como referencia a posição da peça movida.O movimento da peça, assim como a colocação da cerca só serão efetuados após a validação dos inputs do jogador.

```
movePiece(BOARD, P):-
  %ask for the piece you want to move
  write('Please choose the piece you want to move'),
  askPosition(ORI1, ORI2),
  checkPieceToMove(P, ORI1, ORI2),
  !,
  %ask for the final position of the movement
  write('Please introduce the piece final position'),
  askPosition(POS1, POS2),
  checkMovement(BOARD, ORI1, ORI2, POS1, POS2),
  %moves the piece
  movesPiece(BOARD, ORI1, ORI2, POS1, POS2, P),
  printBoard(BOARD),
  %adding a fence
  addFence(BOARD, POS1, POS2),
  endTurn(BOARD, P).
```

No caso de o jogador pretender adicionar uma peça, terá também de indicar a posição do tabuleiro na qual quer que seja adicionada a nova peça. Uma vez mais, a adição só será feita se o input for válido.

```
addPiece(BOARD, P):-
  %count number of pieces in order to check if the player can add another one
  countPieces(BOARD, P),
  !,
  % asking for a position
  write('In which line do you want to add a new piece?'),
  read(POS1),
  write('In which column do you want to add a new piece?'),
  read(POS2),
  placePiece(BOARD, P, POS1, POS2),
  printBoard(BOARD),
  endTurn(BOARD, P).
```

Por último, o jogador pode simplesmente passar a vez.

```
passTurn(BOARD, P):-
  endTurn(BOARD, P).
```

3.4- Execução de Jogadas

De modo a permitir a jogabilidade, tiveram de ser implementadas regras para a verificação de inputs do utilizador.

```
askPosition(POS1, POS2) :-
  write('Choose a line'),
  read(POS1),
  write('Choose a column'),
  read(POS2).
```

Foram também implementadas regras básicas que possibilitaram a jogabilidade.

Algumas dessas regras permitem, mover peças, verificar o movimento das mesmas para diferentes posições, assim como a colocação de cercas e de novas peças.

```
checkPieceToMove(BOARD, POS1, POS2, Line, SPACE):-
  %checks if the position is a piece position and if its empty
  R is POS1 mod 2,
  R \= 0,
  nth0(POS1, BOARD, Line),
  R is POS2 mod 2,
  R \= 0,
  nth0(POS2, BOARD, SPACE),
  SPACE = 0.


checkMovement(BOARD, ORI1, ORI2, POS1, POS2, Line, SPACE):-
  %checks if the position is a piece position and if its empty
  R is POS1 mod 2,
  R \= 0,
  nth0(POS1, BOARD, Line),
  R is POS2 mod 2,
  R \= 0,
  nth0(POS2, BOARD, SPACE),
  SPACE = 0,
  !,
  evaluatePositions(ORI1, ORI2, POS1, POS2).

checkMovement(BOARD, P):-
  write('The introduced position is not valid'),
  playingMenu(BOARD, P).
```

No caso de o jogador escolher mover uma peça e adicionar a cerca, terá de ser verificado se a posição de origem introduzida possui uma peça que pertença a esse jogador, se a posição de destino não está ocupada, e ainda se existe um caminho possível para efetuar esse movimento.

```
openFence(ORI1, ORI2, BOARD, Line):-
  nth0(ORI1, BOARD, Line),
  nth0(ORI2, Line, Elem),
  Elem = 3.

openFence(ORI1, ORI2, BOARD, Line):-
  nth0(ORI1, BOARD, Line),
  nth0(ORI2, Line, Elem),
  Elem = 6.

emptySpace(ORI1, ORI2, BOARD, Line):-
  nth0(ORI1, BOARD, Line),
  nth0(ORI2, Line, Elem),
  Elem = 0.
```

De modo a saber se existe um caminho entre as 2 posições, é utilizada a regra evaluatePositions, que recursivamente procura para as casas adjacentes segundo as regras do jogo, se é possível fazer o movimento. Para isso são necessárias outras duas regras, uma para verificar se existe uma cerca a separar as duas casas, e outra para verificar se a casa seguinte se encontra livre.

Após o movimento ser efetuado, podendo o mesmo ser feito para a posição igual à posição original, o que é sempre possível, sempre que possa o utilizador é obrigado a colocar uma cerca numa posição adjacente à posição final para a qual ele se deslocou.

```
addFence(BOARD, POS1, POS2):-
  write('Introduce where you want to add a fence'),
  ( openFence(BOARD, POS1+1, POS2) = true -> write('down')),
  ( openFence(BOARD, POS1-1, POS2) = true -> write('up')),
  ( openFence(BOARD, POS1, POS2+1) = true -> write('right')),
  ( openFence(BOARD, POS1, POS2-1) = true -> write('left')),
  read(X),
  addFenceOptions(X).

addFenceOptions(BOARD, POS1, POS2, X, Line):-
  (
    X = 'down' -> nth0(POS1+1, BOARD, Line),
                  replace(POS2, 5, Line, Line),
                  replace(POS1+1, Line, Board, Board);
    X = 'up' -> nth0(POS1-1, BOARD, Line),
                replace(POS2, 5, Line, Line),
                replace(POS1-1, Line, Board, Board);
    X = 'right' -> nth0(POS1, BOARD, Line),
                   replace(POS2+1, 4, Line, Line),
                   replace(POS1, Line, Board, Board);
    X = 'left' -> nth0(POS1, BOARD, Line),
                  replace(POS2-1, 4, Line, Line),
                  replace(POS1, Line, Board, Board);
    (write('Wrong Choice!'),nl, addFence(BOARD, POS1, POS2))
  ).
```

Num outro caso em que o jogador não pretenda mover a sua peça e adicionar uma cerca, poderá adicionar uma nova peça ao tabuleiro, para tal, o jogador não poderá mais de 7 peças no tabuleiro, e ao adicionar a nova peça terá de existir pelo menos um caminho possível entre a peça que se pretende adicionar e qualquer outra peça já existente no tabuleiro ou entre qualquer peça existente no tabuleiro e a peça que se pretende adicionar.

```
placePiece(BOARD, P, POS1, POS2):-
  checkIfPossible(BOARD, P, POS1, POS2), !,
  addCorrectPiece(BOARD, P, POS1, POS2),
  updateNumeberOfPieces(P).

updateNumeberOfPieces(P):-
  P = 1,
  nth0(1, P1, NumbP),
  NumbP is NumbP + 1,
  replace(1, NumbP, P1, P1).

updateNumeberOfPieces(P):-
  P = 2,
  nth0(1, P2, NumbP),
  NumbP is NumbP + 1,
  replace(1, NumbP, P2, P2).
```

Após todas as condições serem verificadas, a peça será adicionada na posição pretendida do tabuleiro.

```
addCorrectPiece(BOARD, P, P1, POS1, POS2, Line):-
  P = 1,
  nth0(POS1, BOARD, Line),
  replace(POS2, 1, Line, Line),
  replace(POS1, Line, Board, Board),
  write('A new piece was sucssecfully placed').

addCorrectPiece(BOARD, POS1, POS2):-
  P = 2,
  nth0(POS1, BOARD, Line),
  replace(POS2, 2, Line, Line),
  replace(POS1, Line, Board, Board),
  write('A new piece was sucssecfully placed').
```

Por fim, após cada uma das jogadas, assim como na opção de passar o turno, é chamada a regra endTurn que possibilitará o outro jogador a fazer o seu turno e assim completar a jogada.

```
endTurn(BOARD, P):-
  P = 1,
  P1 is 2,
  playingMenu(BOARD, P1).

endTurn(BOARD, P):-
  P = 2,
  P1 is 1,
  playingMenu(BOARD, P1).
```

3.5- Avaliação do Tabuleiro

O tabuleiro é avaliado sempre que se pretenda alterar posição de uma das peças ou adicionar uma nova. Para além disso, a avaliação é também feita e em maior pormenor quando se verifica se todas as áreas estão ou não fechadas.

Essa verificação das áreas é feita através da regra 'calcArea' que se baseia no algoritmo de *Floodfill*. Sem nunca sair da área contornada pelas cercas, a função procurar todas as casas do tabuleiro, verificando se existe mais do que uma peça na mesma área independentemente de pertencerem ou não ao mesmo jogador.

```
calcArea(POS1, POS2, BOARD):-
  %cria copia do Board
  copy_term(BOARD, Board2),
  %guardar a linha inicial
  floodFill(Board2, POS1, POS2).

floodFill(Board2, POS1, POS2):-
  (openFence(POS1+1, POS2)->emptySpace(POS1+2, POS2, Board2, _), POS1 is POS1 + 2, floodFill(Board2, POS1, POS2)),
  (openFence(POS1-1, POS2)->emptySpace(POS1-2, POS2, Board2, _), POS1 is POS1 - 2, floodFill(Board2, POS1, POS2)),
  (openFence(POS1, POS2+1)->emptySpace(POS1, POS2+2, Board2, _), POS2 is POS2 + 2, floodFill(Board2, POS1, POS2)),
  (openFence(POS1, POS2-1)->emptySpace(POS1, POS2-2, Board2, _), POS2 is POS2 - 2, floodFill(Board2, POS1, POS2)).
```

3.6- Final do Jogo

A verificação do final do jogo, é feita sempre antes de cada jogada. E para tal é verificado se ambos os jogadores têm movimentos possíveis a serem efetuados, uma vez que se um dos jogadores não poder efetuar mais nenhuma jogada, não faz sentido que o jogo continue, ou no caso de todas as áreas se encontrarem fechadas.

```
playingMenu(BOARD, P):-
  checkGameEnd(BOARD, P),
  !,
  write('Game Ended !'), nl,
  calculateScores(BOARD, P1, P2),
  write('Players1 pontuation: '), write(P1), nl,
  write('Players2 pontuation: '), write(P2), nl, nl,
  write('The winner is ').

playingMenu(BOARD, P):-
  nl, nl,
  write('=========================================='),nl,
  write('IS YOUR TURN'),nl,nl,
  printBoard(BOARD),
  write('1- Add Piece'), nl,
  write('2- Move Piece and Add Fence'), nl,
  write('3- Pass Turn'), nl,
  write('4- End Game'), nl,
  read(X),
  gameMenuOption(X, Board, P).
```

```
checkGameEnd(BOARD, P):-
  %se nao existirem movimentos possiveis de uma jogador
  playersCanMove(BOARD, P, 1),
  !,
  fail.

checkGameEnd(BOARD, P):-
  NPieces = 0,
  countPieces(NPieces, 0),
  %check if all areas are surrounded
  checkAllAreas(NPieces, BOARD, 0).
```

4- INTERFACE COM O UTILIZADOR

Para que o jogo se tornasse mais fácil de jogar na consola tentou-se simplicar ao máximo a quantidade de informação que o utilizador teria que inserir na consola para jogar, por isso apostou-se em menus em que o utilizador apenas escolhe o número da acção que pretende fazer, tal qual os

```
==========================================
WELCOME TO FINES

1- Player Vs Player
2- Player Vs AI
3- AI Vs AI
4- Exit
|:
```

exemplos abaixo:

```
==========================================
TIME TO CHOOSE YOUR BATTLE NAME

Player 1 -x- |: andre.

Player 2 -o- |: joao.
```

Durante o jogo sempre que necessário inserir alguma coordenada quer para deslocar uma peça ou acrescentar outra ao tabuleiro é pedido separadamente ao jogador que digite uma linha e uma coluna e caso este insira valores incorrectos, por exempo, valores menores que 1, maiores que 7 ou inválidos em termos de funcionamento do jogo é mostrada uma mensagem de erro e pede-se novos valores ao jogador.

5- CONCLUSÃO

Relativamente a este projecto percebemos que listas são realmente importantes em *prolog* , pois são a forma mais viável de guardar e alterar informação nesta linguagem. Apesar de inicialmente termos tido alguns problemas no manuseamento das listas e da própria linguagem chegamos ao final a compreender como ambas funcionam.

Quanto aos objectivos, não foram todos cumpridos, uma vez que não conseguimos corrigir todos os erros de forma a tornar o jogo funcional apesar de termos desenvolvido todas as regras e factos necessários.Outro dos objectivos não implementados foi o desenvolvimento de uma inteligência artificial, que se deveu ao facto de não termos conseguido corrigir os erros anteriormente referidos.

A não realização destes objectivos deveu-se à grande carga de trabalho de todas as unidades curriculares cujos prazos eram muito próximos e todos com linguagens e tarefas absolutamente distintas entre si. Outro dos factores que prejudicou foram ambos os elementos do grupo terem unidades curriculares do ano anterior, também com trabalhos e testes na mesma fase.

Em suma, com este trabalho aprendemos que temos de organizar ainda mais o nosso tempo para situações destas não voltarem a acontecer.

REFERÊNCIAS

- http://www.boardgamegeek.com/boardgame/159333/fendo
- http://spielstein.com/games/fendo
- http://www.swi-prolog.org/pldoc/doc/swi/library/lists.pl
- Material de apoio fornecido pelos docentes, https://moodle.up.pt/course/view.php?id=2866