U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Quartetto

Relatório Final

Programação em Lógica-PLOG

2018-2019

Francisco Ademar Freitas Friande **up201508213**@fe.up.pt

João Pedro Bandeira Fidalgo **up201605237**@fe.up.pt

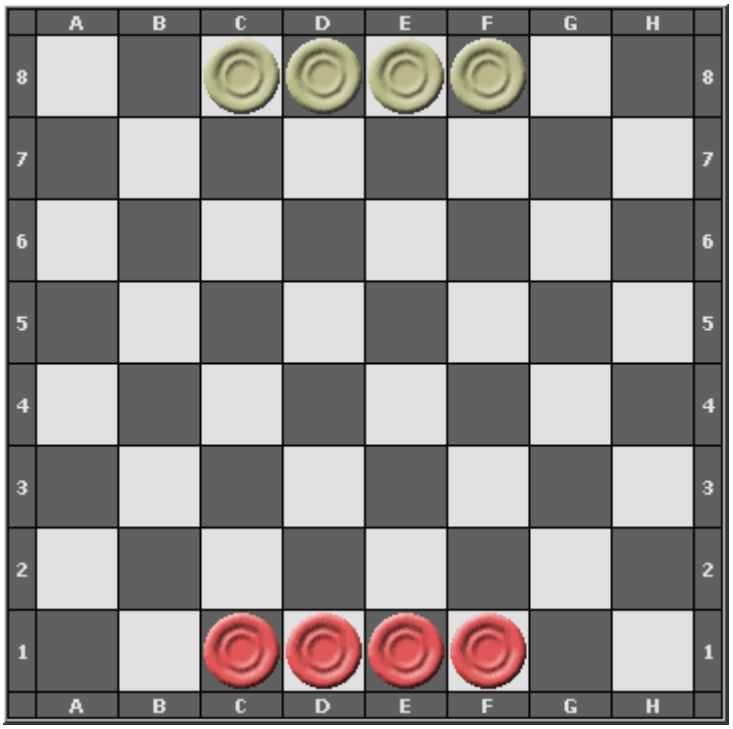
Introdução

O trabalho tem como objetivos a aprendizagem da linguagem Prolog, como base de programação mais declarativa, utilização do SICStus e introdução a um nível baixo de inteligência aritificial, através da implementação de um jogo de tabuleiro que é caracterizado pelas suas listas de jogadas possíveis, condições de vitória, escolha de peças etc.

Descrição do jogo

Quartetto é um jogo de tabuleiro que foi criado em 2008, mas que tem um conceito de jogabilidade estratégica semelhante aos jogos Mondrago, de Adrian Schacker (1992) e Teeko, de John Scarne (1945). Cada jogador começa com quatro peças, as quais se manterão até ao fim.

A condição vencedora é um dos jogadores posicionar as suas quatro peças em disposição quadrangular com arestas não paralelas aos limites do tabuleiro. Sendo semelhante aos jogos previamente mencionados, difere no facto de a área do quadrado, em termos de colunas e linhas englobadas ter de ser maior do que 5x5.



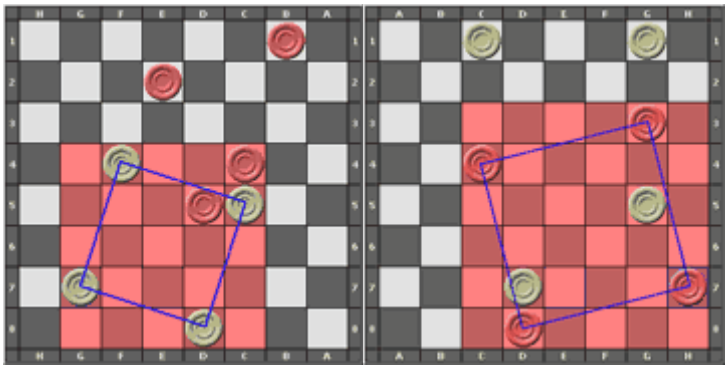
O tabuleiro é quadrado com 8x8 espaços, começando as peças dispostas da maneira apresentada acima.

Objetivo

O objectivo do jogo é um jogador posicionar as suas peças da seguinte forma:

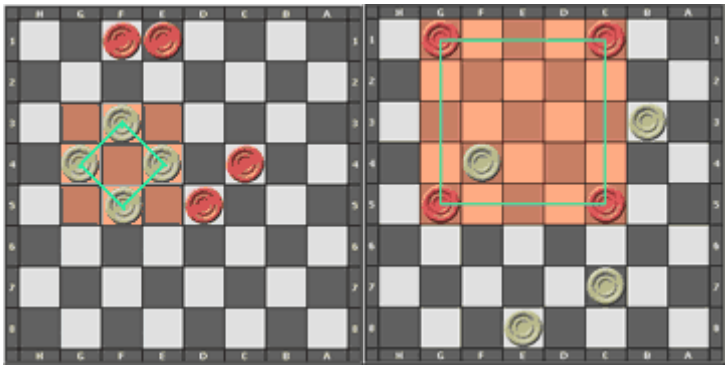
1. Os vértices devem formar um quadrado rodado em relação ao tabuleiro.
2. A “caixa” delimitada pelos vértices terá de ser maior ou igual a 5x5.

Vejamos exemplos do descrito.



Peças brancas ganham
(quadrado com “caixa” de 5x5)

Peças vermelhas ganham
(quadrado com caixa de 6x6)



Peças brancas não ganham,
apesar de fazerem um quadrado rodado,
mas apenas com 3x3

Peças vermelhas não ganham,
apesar de fazerem uma “caixa” de 5x5
a caixa é paralela às arestas

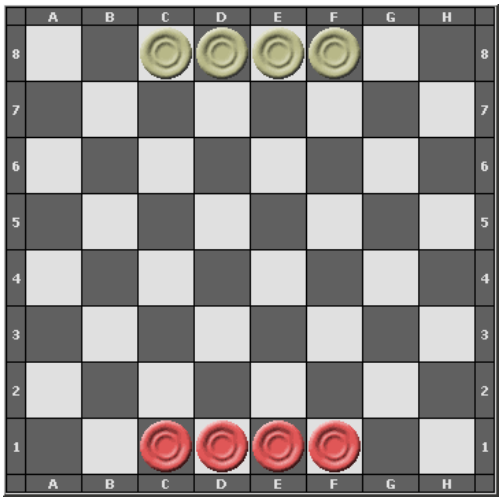
Movimentação

São consideradas jogadas válidas qualquer jogada na horizontal ou vertical e que não irá sobrepor nenhuma outra peça (aliada ou adversária). O jogo acaba quando um jogador atinge o objetivo e ganha.

Lógica do jogo

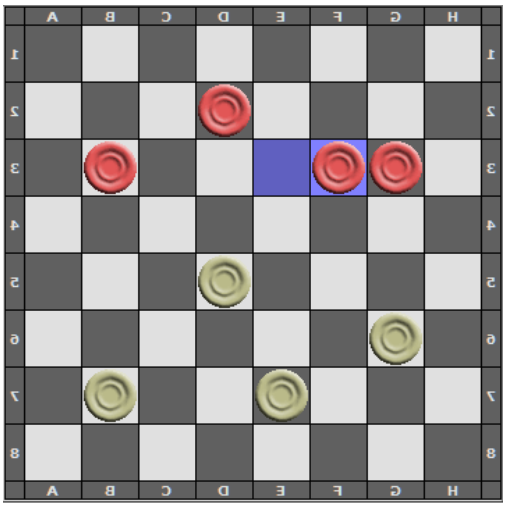
Representação do tabuleiro

Estado Inicial



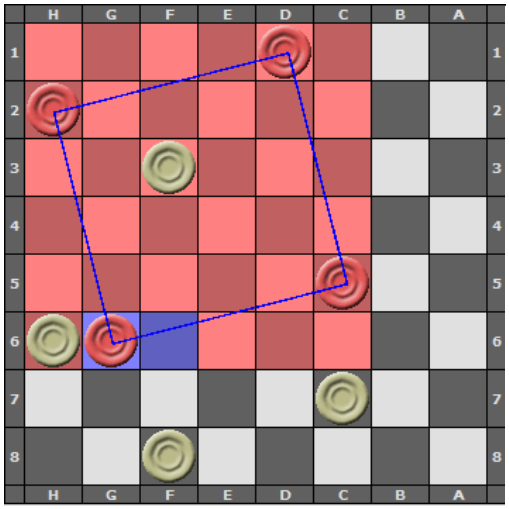
```
  | H | G | F | E | D | C | B | A |
 ---------------------------------
1|   |   | R | R | R | R |   |   |1
 ---------------------------------
2|   |   |   |   |   |   |   |   |2
 ---------------------------------
3|   |   |   |   |   |   |   |   |3
 ---------------------------------
4|   |   |   |   |   |   |   |   |4
 ---------------------------------
5|   |   |   |   |   |   |   |   |5
 ---------------------------------
6|   |   |   |   |   |   |   |   |6
 ---------------------------------
7|   |   |   |   |   |   |   |   |7
 ---------------------------------
8|   |   | W | W | W | W |   |   |8
 ---------------------------------
  | H | G | F | E | D | C | B | A |
```

Estado Intermédio



```
  | H | G | F | E | D | C | B | A |
 ---------------------------------
1|   |   |   |   |   |   |   |   |1
 ---------------------------------
2|   |   |   |   |   | R |   |   |2
 ---------------------------------
3|   |   | R | R |   |   |   | R |3
 ---------------------------------
4|   |   |   |   |   |   |   |   |4
 ---------------------------------
5|   |   |   |   |   | W |   |   |5
 ---------------------------------
6|   |   | W |   |   |   |   |   |6
 ---------------------------------
7|   |   |   |   | W |   |   | W |7
 ---------------------------------
8|   |   |   |   |   |   |   |   |8
 ---------------------------------
  | H | G | F | E | D | C | B | A |
```

Estado Final



```
  | H | G | F | E | D | C | B | A |
 ---------------------------------
1|   |   |   |   |   | R |   |   |1
 ---------------------------------
2| R |   |   |   |   |   |   |   |2
 ---------------------------------
3|   |   |   | W |   |   |   |   |3
 ---------------------------------
4|   |   |   |   |   |   |   |   |4
 ---------------------------------
5|   |   |   |   |   |   | R |   |5
 ---------------------------------
6| W |   | R |   |   |   |   |   |6
 ---------------------------------
7|   |   |   |   |   |   | W |   |7
 ---------------------------------
8|   |   |   | W |   |   |   |   |8
 ---------------------------------
  | H | G | F | E | D | C | B | A |
```

Visualização do Tabuleiro

O tabuleiro é imprimido no ecrã através da função **display_game** que recebe o **board** como argumento. Começa por imprimir as letras das coordenadas e depois chama o **print_tab** que dá print ao tabuleiro todo recursivamente e no início e fim de cada linha imprime os números para as coordenadas.

```
printLetters:-write('  | H | G | F | E | D | C | B | A | '), nl.

printSeparator:-write('_____________________________________'), nl.

display_game(Board):-
    printLetters,
    printSeparator,
    print_tab(Board,['1','2','3','4','5','6','7','8']),
    printLetters,
    nl.
```

Lista de jogadas válidas

A lista de jogadas válidas é obtida pela função **valid_moves(+Board,+Row,+NumColumn,-ListOfMoves)**, que dá uma lista de jogadas válidas e imprime apenas a lista de jogadas para a peça que fora escolhida pelo jogador. Este predicado chama 4 funções diferentes , cada uma trata de ver as jogadas para cada sentido da peça, cima, baixo, esquerda, direita. No fim dá *append* de todas as listas obtidas e a lista final é guardada na variável **ListOfMoves**.

```
valid_moves(Board,Row,NumColumn,MovesList):-
    listColumnDown(Board,MovesList1,Row,NumColumn,TempMovesList),
    listColumnUp(Board,MovesList2,Row,NumColumn,TempMovesList1),
    listRowRight(Board,MovesList3,Row,NumColumn,TempMovesList2),
    listRowLeft(Board,MovesList4,Row,NumColumn,TempMovesList3),
    append(MovesList1,MovesList2,MovesListAux),
    append(MovesListAux,MovesList3,MovesListAux1),
    append(MovesListAux1,MovesList4,MovesList),!.
```

Execução de Jogadas

O predicado que trata da execução de jogadas é o **move(+Board,-NewBoard,+Symbol,+Player,+Level)** que tem duas declarações , uma em que o **Player** é 'P', que trata da movimentação de um jogador, e outra que é 'C', que trata da movimentação do computador.

No caso de ser um jogador, começa por pedir a peça que o jogador quer mover e, enquanto esta não for válida (**checkCell**), ou caso esteja rodeada por peças e não seja possível movê-la (**validPiece)**, volta a pedir. Depois chama a função **chooseDest** que trata de perguntar o destino da peça escolhida e, enquanto o jogador não escolher um local que esteja na lista de jogadas válidas, repete o pedido de input das coordenadas.

```
move(Board,NewBoard,Symbol,'P',_) :-
    repeat,
    write('Which piece you would like to move?\n'),
    chooseCell(Row,NumColumn),
    ((checkCell(Board,Row,NumColumn,Symbol),
    valid_moves(Board,Row,NumColumn,MovesList),
    validPiece(MovesList));
    (write('Not a valid piece or with no moves! Choose again.\n\n'),
    fail)),!,
    replaceRows(Board,Row,NumColumn,0,BoardAux),
    chooseDest(BoardAux,NewBoard,Symbol,Row,NumColumn,MovesList),!.
```

No caso do computador apenas chama o predicado que trata da movimentação do bot para cada dificuldade.

```
move(Board,NewBoar,Symbol,'C',Level) :-
    choose_move(Board,NewBoar,Level,Symbol).
```

Final do jogo

O perdicado **game_over(+PiecesPositionsList,+Player)** é responsável por verificar se as peças de um jogador satisfazem a condição de vitória. As posições das peças estão guardadas na variável **PiecesPositionsList** e através desta lista, primeiramente, verifica-se se as peças com a maior, menor, linha e coluna tem diferença de 5 para caso formem um quadrado, o quadrado que as envolve seja maior ou igual a 5x5, **checkSpan**, depois verifica-se se formam um quadrado entre si, **isSquare**, em seguida, se o quadrado formado não está paralelo ao tabuleiro, **isRotated**.

```
game_over(PiecesPositionsList, Player):-
    checkSpan(PiecesPositionsList,0,0,8,8,RowSpan,ColumnSpan),!,
    (RowSpan>4,
    ColumnSpan>4,
    distanceBetween2(1,2,PiecesPositionsList,DistanceList,_),!,
    (isSquare(DistanceList),
    isRotated(PiecesPositionsList)),
    print_win(Player)
   ).
```

Avaliação do tabuleiro

O computador avalia as jogadas de cada peça com as seguintes condições: se é possível vencer o jogo com alguma das jogadas; ou se alguma das jogadas levará a que a peça forme um triângulo retângulo com duas outras peças. Caso a vitória seja possível, o *bot* vai optar por essa jogada, caso contrário, verifica a jogada mais próxima da vitória. O predicado responsável por isso é o **value(+PiecesPositionsList,+Coords,+Index,-Value)**.

```
value(PiecesPositionsList,Coords,Index,Value):-
    ((valueWin(PiecesPositionsList,Index,Coords),
    Value is 3);
        ((check_for_triangle(PiecesPositionsList,Coords,Index),
        Value is 2);
            Value is 0)).
```

Já o predicado que é responsável por verificar se uma certa jogada pode dar a vitória é o **valueWin(+PiecesPositionsList,+Index,+Coords)** que falha caso não aconteça e dá sucesso caso se satisfaça.

```
valueWin(PiecesPositionsList,Index,Coords) :-
    getRowNumColumn(Coords,Row,NumColumn),
    Coords1=[Row,NumColumn],
replaceColumns(PiecesPositionsList,Index,Coords1,PiecesPositionsListTemp),!,
    game_over(PiecesPositionsListTemp).
```

Enquanto que o predicado **check_for_triangle(+PiecesPositionsList, +Move, +PieceIndex)** é bem sucedido se encontrar um triângulo retângulo entre a peça com coordenadas **Move** e todas as combinações com as outras três peças. É pelo motivo de ser facilitada o cálculo das combinações 3 a 3, que a peça a ser movida é inserida na primeira posição da lista a ser verificada.

```
check_for_triangle(PiecesPositionsList,Move,PieceIndex):-
   nth1(PieceIndex,PiecesPositionsList,CurrentPiece),
   delete(PiecesPositionsList,CurrentPiece,PiecesPositionsListTemp),
   getRowNumColumn(Move,R,NC),
append([[R,NC]],PiecesPositionsListTemp,PiecesPositionsList_2Bchecked),!,
   recursiveCheck_for_triangle(PiecesPositionsList_2Bchecked, 5).
```

Esta função recorre, em última instância, a **isTriangle(+TriCoords)**, que verifica se as três peças a ser verificadas perfazem um triângulo retângulo, rodado em relação ao mapa e para o qual irá existir a possibilidade de ser formado um quadrado em jogadas futuras.

```
isTriangle(TriCoords):-
   distanceAmong3(1,2,TriCoords,DistanceList,_),!,
   isSquare(DistanceList),
   isRotated(TriCoords),
   tryAllCombos(TriCoords, 0).
```

Jogada do Computador

A jogada do computador é tratada pelo predicado **choose_move(+Board,-NewBoard,+Level,+Symbol)** que tem duas declarações diferentes conforme a dificuldade do *bot*. Caso seja nível 1 chama **choose_move(+Board,-NewBoard,1,+Symbol)**, em que *bot* que joga aleatoriamente; no caso do *bot* de nível 2, **choose_move(+Board,-NewBoard,2,+Symbol)**, em que o *bot* que escolhe a melhor jogada possível naquele turno.

No caso do *bot* de nível 1 o predicado começa por obter a posição das peças, **getPiecesList**, e depois chama o predicado **make_randomMove**, que escolhe uma das peças de forma aleatória, depois encontra as jogadas possíveis para a peça escolhida, **valid_moves**, e escolhe uma dessas jogadas de forma aleatória.

```
make_randomMove(Board,Symbol,PiecesPositionsList,NewBoard):-
    repeat,
    random(1, 5, PieceIndex),
    nth1(PieceIndex,PiecesPositionsList,PieceCoords),
    getRowColumn(PieceCoords,Row,NumColumn),
    valid_moves(Board,Row,NumColumn,MovesList),
    validPiece(MovesList),!,
    replaceRows(Board,Row,NumColumn,0,BoardAux),
    length(MovesList, L),
```

```
    Length is L+1,
    random(1,Length,MoveIndex),
    nth1(MoveIndex,MovesList,MoveCoords),
    getRowNumColumn(MoveCoords,Row1,NumColumn1),
    replaceRows(BoardAux,Row1,NumColumn1,Symbol,NewBoard).
```

```
choose_move(Board, NewBoard, 1, Symbol):-
    getPiecesList(Board, Symbol,PiecesPositionsList, _,0),
    make_randomMove(Board,Symbol,PiecesPositionsList,NewBoard).
```

No caso do *bot* de nível 2, o predicado é **choose_move(+Board,-NewBoard,+Level,+Symbol)** que começa por obter uma lista com as coordenadas das peças, **getPiecesList**, depois obtém, através de **listAllValidMoves**, as jogadas possíveis para todas essas peças e em seguida chama **recursiveValue**, que trata de dar um valor a todas as jogadas possíveis. Logo a seguir é obtido o maior valor de todas as jogadas, **getBestValue**, e com ele as melhores jogadas, **getBestsMoves**. Por fim entre as melhores jogadas é escolhida uma de forma aleatória e são feitas todas as alterações no **NewBoard**.

```
choose_move(Board, NewBoard, 2, Symbol):-
    getPiecesList(Board, Symbol,PiecesPositionsList, _,0),
    listAllValidMoves(Board,PiecesPositionsList, AllMovesList),
    recursiveValue(PiecesPositionsList, AllMovesList,0,ValuesList),
    getBestValue(BestValue,ValuesList,0),
    getBestsMoves(AllMovesList,ValuesList,BestsMoves,BestValue,_),
    repeat,

choose_randomMovesList(BestsMoves,ChosenList,PieceIndex,ChosenLength),
    ChosenLength > 1,!,
    nth1(PieceIndex,PiecesPositionsList,PieceCoords),
    getRowColumn(PieceCoords,Row,NumColumn),
    replaceRows(Board,Row,NumColumn,0,BoardAux),
    random(1,ChosenLength,MoveIndex),
    nth1(MoveIndex,ChosenList,MoveCoords),
    getRowNumColumn(MoveCoords,Row1,NumColumn1),
    replaceRows(BoardAux,Row1,NumColumn1,Symbol,NewBoard).
```

Conclusões

Com o trabalho conseguimos ficar a entender melhor a linguagem de Prolog, tal como o paradigma de programação em lógica. O trabalho poderia ser melhorado de forma a que o *bot* tivesse mais condições de movimentação, pois seria mais rápido a concluir o jogo e mais imune a falhas.

Bibliografia

Fotografias na Descrição e Objetivo:

http://www.iggamecenter.com/info/en/quartetto.html

Fotografias visualização do tabuleiro:

http://www.iggamecenter.com

Algumas bases para aprendizagem da linguagem:

https://sicstus.sics.se/sicstus/docs/latest4/html/sicstus.html/