
Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

# Quantick

## Relatório Final

3MIEIC06 - Quantik4

Andreia Gouveia up201706430@fe.up.pt
João Filipe Carvalho de Araújo up201705577@fe.up.pt

Novembro de 2019

# Índice

# 1.Introdução

O presente trabalho tem como propósito a implementação de um tabuleiro de jogo na linguagem Prolog, aplicando os conhecimentos aprendidos nas aulas teóricas e prática de PLOG.

O nosso grupo escolheu o jogo Quantik , que se baseia no Quatro em linha e no Sudoku. Implementamos o jogo de três formas distintas, permitindo ao jogador decidir se pretende jogar contra outro jogador, contra o computador, ou colocar o computador a jogar contra outro computador. É de notar que em qualquer destes 3 modos, as regras foram implementadas com sucesso.

Neste relatório encontra-se a descrição do jogo, tal como uma explicação do modo de implementação escolhido pelo nosso grupo.

# 2.Quantik

## 2.1 História do jogo

Quantik foi inventado por Nouri Khalifa . O jogo foi publicado e distribuído pela Gigamic em setembro de 2019

## 2.2 Constituição do jogo

É um jogo para 2 pessoas. Cada jogador tem à sua disposição 8 peças (2 cubos, 2 cilindros, 2 cones e 2 esferas), sendo que cada jogador é diferenciado pela cor das suas peças (podendo ser brancas ou pretas). O jogo também inclui um tabuleiro,dividido em 4 quadrantes, que possui 16 posições onde os jogadores vão poder colocar as suas peças.

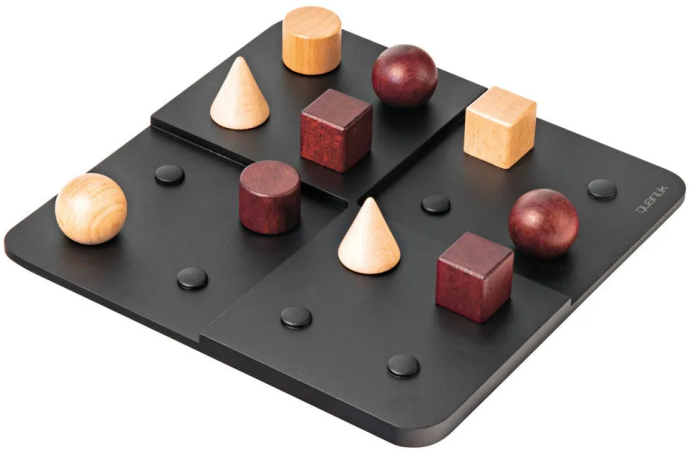


**Figura 1:** Imagem do tabuleiro

## 2.3 Regras e objectivos do jogo

As regras do jogo são relativamente simples. A cada ronda, os jogadores vão colocar uma das suas peças no tabuleiro em espaços vazios. É proibido colocar uma peça de determinada forma, numa linha, coluna ou quadrante onde já exista uma peça com essa mesma forma do adversário. Caso a peça repetida seja do mesmo jogador, então a jogada é válida. O primeiro jogador a conseguir ter uma fila, coluna ou quadrante do tabuleiro com todas as formas diferentes (independentemente se forem ou não da cor do jogador)  ganha o jogo imediatamente.

# 3.Lógica do jogo

## 3.1 Representação interna

### 3.1.1 Tabuleiro

O nosso tabuleiro é representado por uma matriz de 4x4. Ou seja, uma lista de listas de inteiros.

```
initialBoard(
            [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]).
```

Inicialmente o tabuleiro está vazio, sendo que o vazio (ou seja, sem a presença de peças) é representado pelo inteiro 0.

### 3.1.2 Peças

Existem no total 16 peças neste jogo, todas representadas por chars. Estas 16 dividem-se em 8 pretas (representadas por letras minúsculas) e 8 brancas (representadas por letras maiúsculas). Cada jogador terá 2 cones(‘p’ ou ‘P’ dependendo da cor) , 2 cubos(‘c’ ou ‘C’ dependendo da cor), 2 esferas(‘e’ ou ‘E’ dependendo da cor) e 2 cilindros(‘l’ ou ‘L’ dependendo da cor).

Segue-se o código que representa esta atribuição:

```
% Tradution
piece(0, V) :- V = '*'.
piece(1, V) :- V ='p'.
piece(6, V) :- V ='P'.
piece(2, V) :- V ='c'.
piece(7, V) :- V ='C'.
piece(3, V) :- V ='l'.
piece(8, V) :- V ='L'.
piece(4, V) :- V ='e'.
piece(9, V) :- V ='E'.
```

Relativamente à cor das peças, as brancas são representadas pelo conjunto de número [6,7,8,9] e as pretas pelo conjunto [1,2,3,4].

### 3.1.3 Jogadores

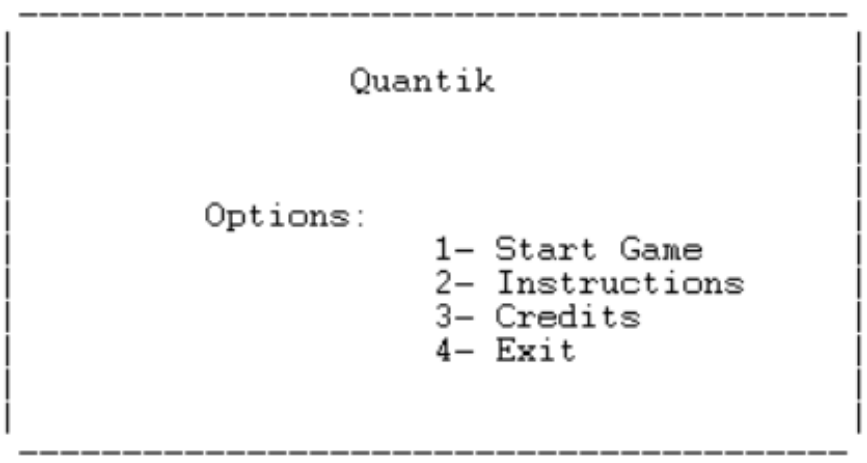
Os jogadores são representados por átomos denominados de white e black, com respectivo valor de 1 e 2. Por “default”, colocamos o jogador branco a ser o primeiro a jogar.

```
player(white , V) :- V = 1.
player(black , V) :- V = 2.
```

## 3.2 Visualização

Quando o jogo é inicializado, surge um menu principal que dá ao jogador 4 opções:

- Começar o jogo
- Ver as instruções do jogo
- Ver os créditos do jogo
- Sair do jogo



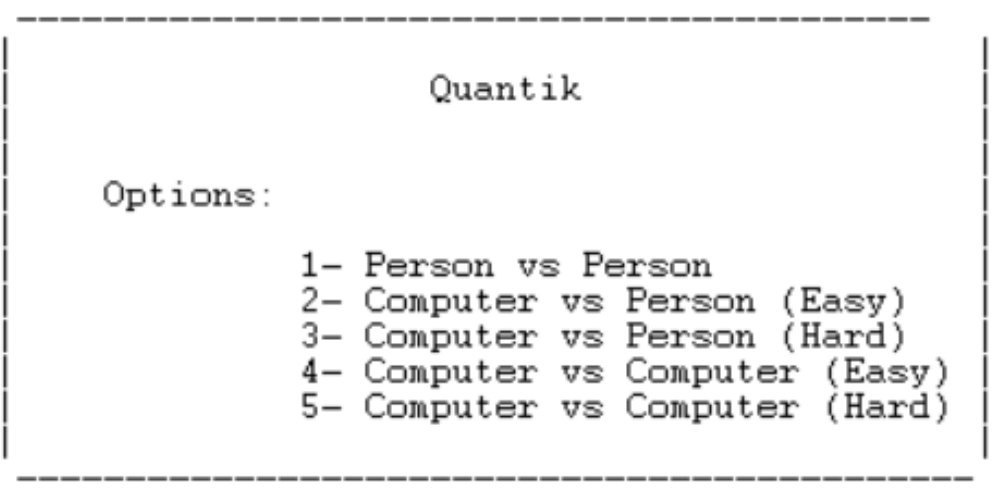

Segue-se o código responsável pelo display do menu principal:

```
menus:- % First Menu
        repeat,
        displayMainMenu,
        nl,
        read_line([H|T]),
        length(T,0),
        !,
        Input is H-48,
        menuChoice(Input).
```

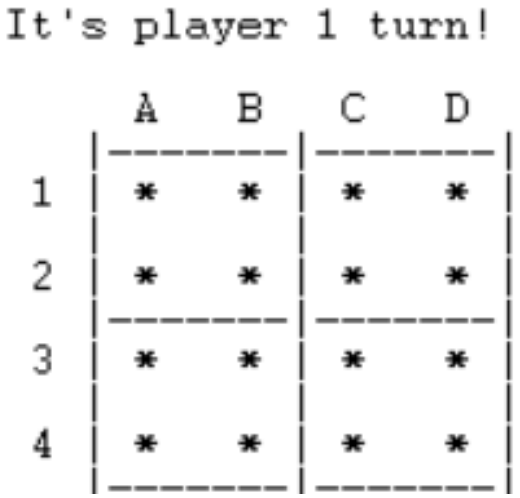
```
displayMainMenu:-
        write('\n ---------------------------------------\n'),
        write('|                                       |\n'),
        write('|               Quantik                 |\n'),
        write('|                                       |\n'),
        write('|                                       |\n'),
        write('|                                       |\n'),
        write('|        Options:                       |\n'),
        write('|                  1- Start Game        |\n'),
        write('|                  2- Instructions      |\n'),
        write('|                  3- Credits           |\n'),
        write('|                  4- Exit              |\n'),
        write('|                                       |\n'),
        write('|                                       |\n'),
        write(' ---------------------------------------\n').
```

Caso a opção escolhida seja começar o jogo, é apresentado um segundo menu que permite ao jogador escolher entre 5 opções:

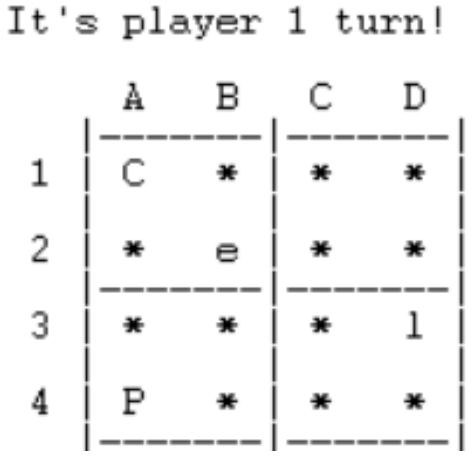
- Person VS Person
- Computer VS Person (Easy)
- Computer VS Person (Hard)
- Computer VS Computer (Easy)
- Computer VS Computer (Hard)



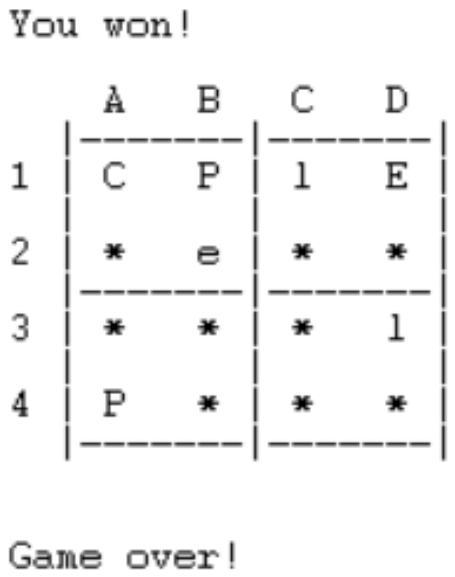

Ambos 3 modos possuem o mesmo modo de começar, é apresentado ao jogador um tabuleiro vazio. Em cima do tabuleiro, o programa indica de quem é a vez de jogar.



A imagem em cima, refere-se ao tabuleiro num estado inicial. Num estado mais avançado do jogo, iremos encontrar um tabuleiro do género:




No final do jogo, neste caso, numa situação de vitória, o tabuleiro apresenta-se da seguinte maneira:



Segue-se o código responsável pelo display do tabuleiro:

```
printBoard(Board):-
        write('\n  '),
        length(Board, X),
        printTopBoard(X,0),
        write('\n   ') ,
        printEmptyLine(X,0),
        printBoard(Board, 1).
```

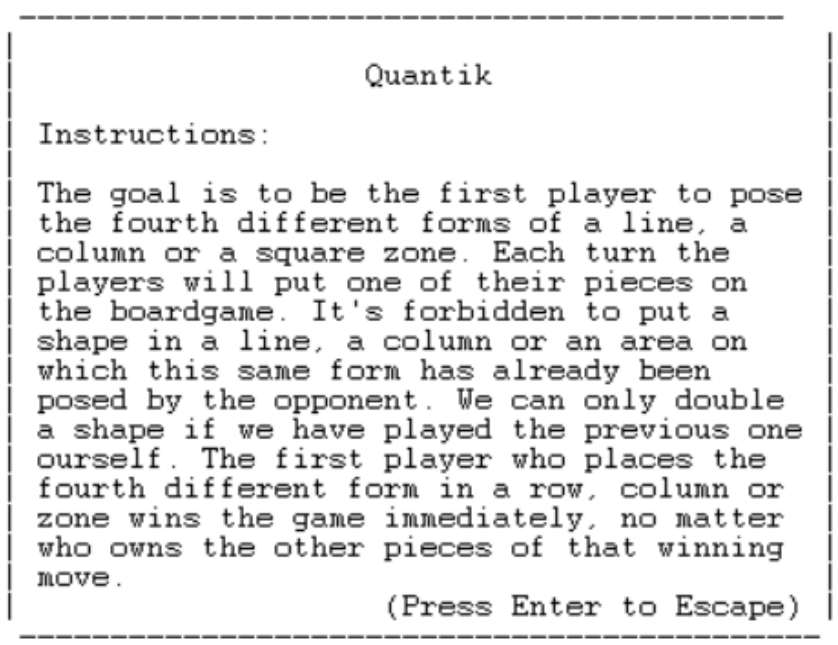
A função `printTopBoard`, vai ser responsável por imprimir o topo do tabuleiro, ou seja, a parte alfabética das coordenadas.

```
printBoard([],_):-
        nl,
        nl.

printBoard([H|T],Num):-
        write('\n '),
        write(Num),
        printLine(H),
        length(H,X),
        write('\n   '),
        Num1 is Num mod 2,
        printEmptyLine(X,Num1),
        Num2 is Num+1,
        printBoard(T,Num2).
```

A função `printBoard`(com 2 argumentos) vai imprimir o restante tabuleiro.

Caso a opção escolhida seja as instruções do jogo, é apresentado na consola o seguinte ecrã:

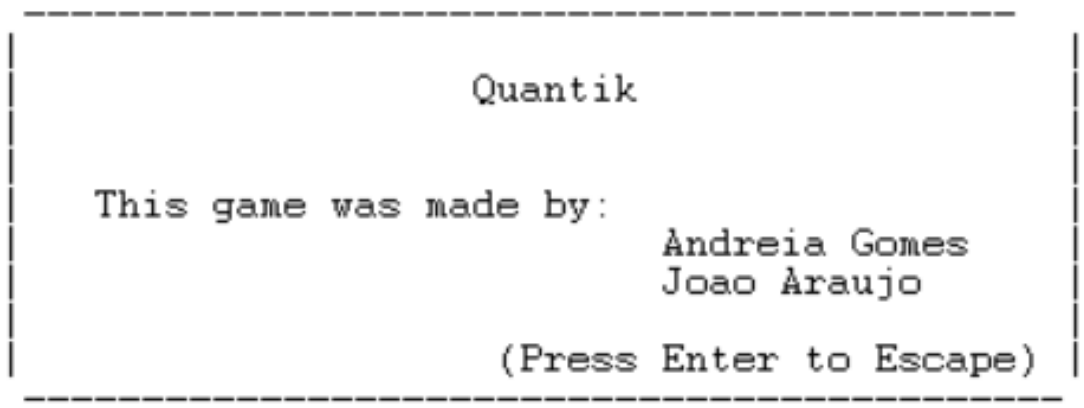



O código responsável por este ecrã é o seguinte:

```
menuChoice(2):- % instructions
        displayInstructions,
        read_line(_),
        menus.
```

```
displayInstructions:-
        write('\n ----------------------------------------\n'),
        write('|                                        |\n'),
        write('|                 Quantik                |\n'),
        write('|                                        |\n'),
        write('| Instructions:                          |\n'),
        write('|                                        |\n'),
        write('| The goal is to be the first player to pose |\n'),
        write('| the fourth different forms of a line, a    |\n'),
        write('| column or a square zone. Each turn the     |\n'),
        write('| players will put one of their pieces on    |\n'),
        write('| the boardgame. It\'s forbidden to put a     |\n'),
        write('| shape in a line, a column or an area on    |\n'),
        write('| which this same form has already been      |\n'),
        write('| posed by the opponent. We can only double  |\n'),
        write('| a shape if we have played the previous one |\n'),
        write('| ourself. The first player who places the   |\n'),
        write('| fourth different form in a row, column or  |\n'),
        write('| zone wins the game immediately, no matter  |\n'),
        write('| who owns the other pieces of that winning  |\n'),
        write('| move.                                  |\n'),
        write('|                  (Press Enter to Escape) |\n'),
        write(' ----------------------------------------\n').
```

Caso a opção escolhida seja os créditos, é apresentada na consola uma o seguinte ecrã:

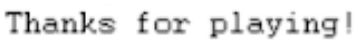



Segue-se o código responsável pela apresentação deste ecrã:

```
menuChoice(3):- % credits
        displayCredits,
        read_line(_),
        menus.
```

```
displayCredits:-
        write('\n ----------------------------------------\n'),
        write('|                                        |\n'),
        write('|                 Quantik                |\n'),
        write('|                                        |\n'),
        write('|                                        |\n'),
        write('|   This game was made by:               |\n'),
        write('|                           Andreia Gomes|\n'),
        write('|                           Joao Araujo  |\n'),
        write('|                                        |\n'),
        write('|                 (Press Enter to Escape)|\n'),
        write(' ----------------------------------------\n').
```

Caso a opção escolhida seja sair do jogo, é apresentada na consola uma mensagem a agradecer ao jogador por ter jogado.




Segue-se o código responsável pela apresentação desta mensagem:

```
menuChoice(4):- % exitgame
        write('\nThanks for playing!\n').
```

## 3.3 Lista de Jogadas Válidas

A nossa lista de jogadas válidas é obtida pela função `valid_moves`. Esta função vai escolher aleatoriamente uma peça e ver o conjunto de coordenadas onde será válido(de acordo com as regras do jogo) a colocar.

Segue-se o código desta função:

```
valid_moves(_Board, ListOfMoves, Pieces , FinalList):-
    isEmpty(Pieces),
    copy(ListOfMoves,FinalList).

valid_moves(Board, ListOfMoves, [Piece|T] , FinalList):-
    findall(V-X-Y-Piece, (pieceRuleValidation(Board , X , Y , Piece),
        isEmptyCell(X,Y,Board),
        finishMove(Board , X , Y , Piece , [Piece|T] , _NewPieces , NewBoard),
        value(NewBoard , V)), List),
    append(ListOfMoves, List, NewList),
    valid_moves(Board , NewList , T , FinalList).
```

A função vai receber um tabuleiro, `Board` e uma lista de peças disponíveis , `Pieces`. No fim, esta função vai retornar uma lista com as jogadas possíveis para todas as peças possíveis através da variável, `ListOfMoves`.

Esta função irá , com a ajuda da função `findall`, obter todas as combinações de coordenadas possíveis, conjuntamente com o valor do tabuleiro associado(sob a forma de Value-X-Y-Piece).

Faz isto ao chamar 3 funções:

- `pieceRuleValidation`:

  Esta função vai verificar se a peça pode ser colocada na posição pretendida. Primeiramente executa a função `pieceCheckR` que vai verificar se na linha pretendida, há alguma peça igual à do inimigo. A função `pieceCheckC` vai transpor o tabuleiro, sendo que deste modo, permite a reutilização da função `pieceCheckR`. Finalmente é executada a função `getSquare`, que coloca numa lista o quadrante desejado,

permitindo assim também utilizar o mesmo procedimento que a função `pieceCheckR`.

```
pieceRuleValidation(Board , X , Y , Piece):-
    pieceCheckR(Board , Y , Piece),
    pieceCheckC(Board , X , Piece),
    getSquare(Board,X,Y,List),
    pieceCheck(List, Piece).
```

- `isEmptyCell`:

  Verifica se o lugar está vazio no tabuleiro.

```
isEmptyCell(X , Y , List):-
    nth1(Y,List,NewList),
    nth1(X,NewList,0).
```

- `finishMove`:

  Devolve um tabuleiro com a nova jogada, para tal, remove a peça da lista de peças disponíveis e finalmente repõe o antigo tabuleiro com o novo.

```
finishMove(Board , X , Y , Piece , AvailablePieces , UpdatedPieces , NewBoard):-
    removePiece(Piece , AvailablePieces , UpdatedPieces),
    replace(Board, X , Y, Piece, NewBoard).
```

- `value`:

  Devolve o valor do estado do jogo (esta função é melhor descrita no tópico 3

## 3.4 Execução de Jogadas

As nossas jogadas são validadas e executadas, no caso do jogo pessoa contra pessoa, por uma função denominada por `move(Board, X, Y,Piece, AvailablePieces, UpdatedPieces,NewBoard).`

```
move( Board ,  _Player , X , Y , Piece , AvailablePieces , UpdatedPieces , NewBoard):-
    pieceRuleValidation(Board , X , Y , Piece),
    !,
    finishMove(Board , X , Y , Piece , AvailablePieces , UpdatedPieces , NewBoard).

move( _Board ,  _Player , _X , _Y , _Piece , _AvailablePieces , _UpdatedPieces):-
    write("\n Invalid Play\n").
```

Esta função vai receber o tabuleiro atual, `Board`, as coordenadas, (`X, Y`), e a peça, `Piece`, a colocar no novo tabuleiro, `NewBoard`. Recebe também as peças disponíveis, `AvailablePieces` de modo a verificar que a peça escolhida é válida.

A validação da jogada é feita pela função `pieceRuleValidation` ,sendo que esta função foi descrita em detalhe na seção 3.3. Após averiguar se a peça pode ser colocada no tabuleiro, invoca a função `finishMove` que devolve um tabuleiro com a nova jogada, para tal, remove a peça da lista de peças disponíveis e finalmente repõe o antigo tabuleiro com um novo.

É de notar que validação de uma peça numa determinada linha/coluna/quadrante é feita ao verificar se há alguma peça adversária com a mesma forma na mesma linha/coluna/quadrante. Caso tal se verifique, o programa não permite a execução da jogada e volta a pedir uma nova jogada.

## 3.5 Final do Jogo

O final do jogo é feito com a função `game_over ( Board , _Counter, NewCounter)`, que recebe o tabuleiro no seu estado atual ( `Board` ), um contador ( `Counter`) e o contador com o novo valor ( `NewCounter` ).

```
game_over(Board,_Counter,NewCounter):-
    checkWin(Board,1,1),
    !,
    NewCounter is 15,
    write('\n You won! \n').

game_over(Board,_Counter,NewCounter):-
    checkWin(Board,2,3),
    !,
    NewCounter is 15,
    write('\n You won! \n').

game_over(Board,_Counter,NewCounter):-
    checkWin(Board,3,2),
    !,
    NewCounter is 15,
    write('\n You won! \n').

game_over(Board,_Counter,NewCounter):-
    checkWin(Board,4,4),
    !,
    NewCounter is 15,
    write('\n You won! \n').

game_over(_Board,Counter,NewCounter):-
    NewCounter is Counter,
    write('\n Keep playing! \n').
```

Esta função vai chamar a função `checkWin(Board , X , Y)`, que verifica se o tabuleiro está num estado de vitória.

```
checkWin(Board , _X , Y):- % win from row
    nth1(Y,Board,Row),
    winList(Row, 1).

checkWin(Board , X , _Y ):- % win from column
    transpose(Board, Board1),
    nth1(X,Board1,Row),
    winList(Row, 1).

checkWin(Board , X , Y):- % win from square
    getSquare(Board,X,Y,List),
    winList(List, 1).
```

Esta recebe o tabuleiro e a posição a analisar. Verifica se no determinado X, se cumpre a condição de vitória usando a função `winList(Row, 1)`, que verifica se a multiplicação de todos os elementos dessa lista (com mod 5 em todos os elementos) resulta em 24.

Caso a vitória na linha não se verifique, passa a analisar a coluna, usando a mesma função, mas transpondo o tabuleiro. Permitindo assim a reutilização de código e uma verificação mais eficiente.

Caso a vitória na coluna também não se verifique, passamos a verificar o quadrante. O quadrante é colocado numa lista com a ajuda da função `getSquare`, permitindo reutilizar o código de verificação anteriormente referido.

Finalmente, caso nenhuma destas verificações obtenha resultado positivo, podemos afirmar que o jogo pode prosseguir, visto que ainda não houve vitória e contador incrementa uma unidade.

No caso de alguma destas condições se verificar positivamente, o contador passa a ser 15, terminando assim o jogo pois no fim da ronda o contador ficará a 16, que é o número máximo de jogadas.

## 3.6 Avaliação do Tabuleiro

A avaliação do tabuleiro é feita com a função `value`. Esta função vai percorrer as colunas, linhas e quadrantes do tabuleiro.

```
value(Board , Value):-
    %lines
    checkLines(Board , 0 , Counter),
    %rows
    transpose(Board , TransposedBoard),
    checkLines(TransposedBoard , 0 , Counter1),
    atributeBigger(Counter , Counter1 , Value1),
    %squares
    getSquare(Board,1,1,List1),
    getSquare(Board,4,4,List2),
    getSquare(Board,2,3,List3),
    getSquare(Board,3,2,List4),
    checkLines([List1,List2,List3,List4] , 0 , Counter2),
    atributeBigger(Value1 , Counter2 , Value).
```

Para cada linha/coluna/quadrado, este vai verificar o número de zeros existentes. Classificamos as jogadas com 15, 10, 6, 5, 1, 0, -1, sendo que quanto maior é o valor melhor é a jogada, usando a função `atributeValue`, sendo que Line é a lista a avaliar, Value é o valor a devolver e Counter é o número de peças na lista.

Para saber se há peças repetidas, somamos os elementos da linha e se o valor da soma não pertencer à lista de hipóteses possíveis, então haverá peças repetidas.

```
atributeValue(_Line, 0 , 0):- !. % if row is isEmpty ignores value

atributeValue(_Line, 5 , 1):- !. % 1 piece - middle option

atributeValue(Line, 6 , 2):- % 2 different pieces - middle option
    list_sum(Line , Sum),
    member(Sum , [3,4,5,6,7]),
    !.

atributeValue(_Line, 1 , 2):- !. % 2 pieces with repetitions - bad option

atributeValue(Line, -1 , 3):- % 3 different pieces - worst option
    list_sum(Line , Sum),
    member(Sum , [6,7,8,9]),
    !.

atributeValue(_Line, 1 , 3):- !. % 3 pieces with repetitions - bad option

atributeValue(Line, 15 , 4):- % 4 different pieces - best option
    winList(Line,1),
    !.

atributeValue(_Line, 1 , 4):- !. % 4 pieces with repetition - bad option


atributeValue(_Line , 0 , _Counter). % if anything else
```

Depois de todo o tabuleiro avaliado, é vista qual a maior pontuação, sendo esse o valor atribuído à resposta.

```
attributeValue(_Line, 0 , 0):- !. % if row is empty ignores value

attributeValue(_Line, 5 , 1):- !. % 1 piece - middle option

attributeValue(Line, 6 , 2):- % 2 different pieces - middle option
    list_sum(Line , Sum),
    member(Sum , [3,4,5,6,7]),
    !.

attributeValue(_Line, 1 , 2):- !. % 2 pieces with repetitions - bad option

attributeValue(Line, -1 , 3):- % 3 different pieces - worst option
    list_sum(Line , Sum),
    member(Sum , [6,7,8,9]),
    !.

attributeValue(_Line, 1 , 3):- !. % 3 pieces with repetitions - bad option

attributeValue(Line, 15 , 4):- % 4 different pieces - best option
    winList(Line,1),
    !.

attributeValue(_Line, 1 , 4):- !. % 4 pieces with repetition - bad option
```

## 3.7 Jogada do Computador

A jogada do computador é feita pela função `choose_move`, tem dois modos de implementação diferentes, dependendo da decisão do jogador ao escolher o nível de dificuldade.

Caso o nível escolhido seja:

- Aleatório

```
choose_move(_Board , random , _X , _Y , _Piece, []):-
    write('\n ---- No available moves ----').

choose_move(Board , random , X , Y, Piece, ListOfMoves):-
    random_select(Elem,ListOfMoves,_),
    getMove(Elem , X , Y , Piece),
    validMove(X , Y , Board),
    !,
    write('\n Column: '),
    X1 is X + 64,
    char_code(C,X1),
    write(C),
    write('\n Line: '),
    write(Y),
    write('\n Piece: '),
    piece(Piece,C1),
    write(C1),
    nl.
```

Aqui, a escolha da peça e a posição, é feita aleatoriamente com a ajuda da função random_select, que vai escolher um movimento aleatoriamente da lista de movimentos válidos.

- Inteligência Artificial

```
choose_move(_Board , smart , _X , _Y, _Piece, []):-
    write('\n ---- No available moves ----').

choose_move(Board , smart , X , Y, Piece, List):-
    getBestMove( List , _Move, BestMove),
    getMove(BestMove , X , Y , Piece),
    validMove(X , Y , Board),
    !,
    write('\n Column: '),
    X1 is X + 64,
    char_code(C,X1),
    write(C),
    write('\n Line: '),
    write(Y),
    write('\n Piece: '),
    piece(Piece,C1),
    write(C1),
    nl.
```

Já esta função escolhe a jogada que tiver o maior valor, ou seja, maior probabilidade de não levar à derrota. Isto é feito com a função `getBestMove`.

```
bestMove([] , Move, Move).

bestMove([H|T], Move, BestMove):-
    getValue(H ,Value),
    getValue(Move , Value2),
    Value > Value2, % if new move is better than the last
    !,
    bestMove(T , H, BestMove).

bestMove([_H|T] , Move, BestMove):-
    bestMove(T , Move, BestMove).

getBestMove(List , Move, BestMove):-
    getfirstelement(List, Move),
    bestMove(List , Move, BestMove).

% === get move
getMove(_V-X-Y-Piece, X , Y, Piece).
```

# 4.Conclusão

O objetivo deste projeto foi dar a conhecer o funcionamento da linguagem Prolog, tal como a implementação de um jogo à nossa escolha.

Sentimos algumas dificuldades inicialmente a perceber o funcionamento desta nova linguagem, como também a validação da jogada e o como obter a melhor jogada possível e a realização da inteligência artificial, mas com a prática e com ajuda das aulas conseguimos superá-las.

Achamos que o nosso trabalho poderia ser melhorado na parte da interface, como também na parte da inteligência artificial. Devido à falta de tempo, não nos foi possível melhorar estas partes.

Por fim, achamos que este projecto foi essencial para melhorar a nossa compreensão sobre a linguagem, tal como sobre a matéria lecionada nesta disciplina.

# 5.Bibliografia


- https://en.1jour-1jeu.com/people/nouri-khalifa/designer
- https://en.1jour-1jeu.com/boardgame/2019-quantik/
- https://en.gigamic.com/game/quantik