# Q!nto

## Relatório Final


Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

**Grupo Q!nto_1:**
João Miguel Fidalgo Esteves Nogueira - up201303882
Luís Miguel da Costa Oliveira - up201304515

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

8 de Novembro de 2015

# 1 Resumo

O projecto consiste num jogo de tabuleiro para duas pessoas em que cada jogador pode, no seu turno, ter uma de três acções: sendo elas jogar, trocar cartas por *wildcards* no tabuleiro ou passar (e trocar cartas da sua mão com as do baralho). O vencedor é o que ficar sem cartas em primeiro lugar. O nosso objectivo era fazer uma representação do jogo tão fiel ao original quanto possível, para isso decidimos criar um tabuleiro dinâmico e representado por uma lista de listas de pares. As cartas que têm uma cor e um naipe associado são representadas por um par.

No final obtivemos um projecto com o qual ficamos contentes e nos ajudou a consolidar e colocar em prática os conhecimentos adquiridos desde o início do semestre. De facto deparámo-nos com algumas dificuldades, particularmente com a complexidade em aplicar as regras do jogo, o que foi encarado como um desafio a superar.

# Conteúdo

# 2 Introdução

Este projecto surge no âmbito da unidade curricular Programação em Lógica do 3º ano do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto e tem como objectivo a implementação de um jogo de tabuleiro em *Prolog* com interface em *"txt"*. Para efeitos de testes utilizamos o software recomendado, o *SicStus-Prolog*.

O nosso grande objectivo ao partir para a realização deste projecto era, como já foi dito no resumo inicial, conseguirmos obter no final uma representação fiel ao jogo original. Para isso tentamos desenvolver uma interface simples e agradável para o utilizador, garantimos que todas as regras na colocação eram cumpridas e fazendo-o conseguimos aprofundar e consolidar os nossos conhecimentos relativamente a operações sobre listas, entrar em contacto com outros paradigmas de programação e desenvolver o nosso pensamento lógico.

# 3 O Jogo - Q!nto

## 3.1 História

"Q!nto" é um jogo de tabuleiro recente, criado no ano de 2014 no qual podem participar entre 2 e 4 jogadores. Foi criado por Gene Mackles e lançado pela PDG games

Este jogo faz parte de uma coleção de três jogos chamada Triple Play que é uma coleção de jogos de tabuleiro.

## 3.2 Conteúdo do Jogo

Estão incluídas no Jogo 60 cartas divididas como mostra a figura à direita.

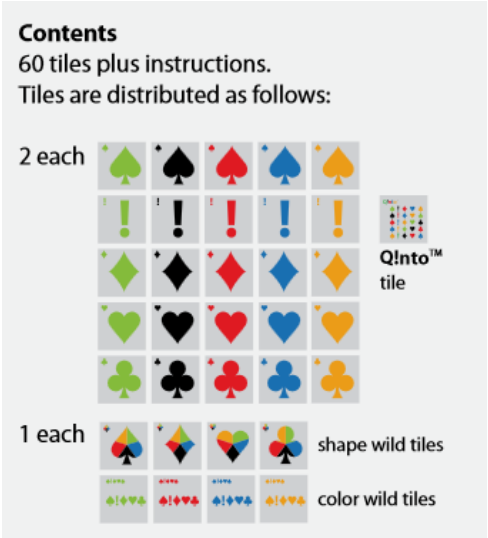


Figura 1: Baralho

## 3.3 Regras de Jogo

Existem três versões para jogar o Q!into, sendo elas o *Classic*, *Plus* e *Light*. Neste projeto usaremos apenas a versão Light que passamos a explicar a seguir:

Inicialmente cada jogador retira uma carta do baralho para escolher quem começa a jogar. O jogo começa dividindo igualmente o baralho pelos jogadores presentes.

As cartas que cada jogador recebeu serão a pilha de cartas com que ele vai poder jogar. Estas ficarão viradas para baixo e cada jogador retira da pilha de cartas que lhe foi atribuída as primeiras 5.

**Noção base:** Uma linha são 2, 3, 4 ou 5 cartas numa linha ou coluna na qual as cartas têm a mesma côr ou cores diferentes e na qual as cartas têm a mesma forma ou formas diferentes. Uma linha com 5 cartas é um Q!nto. O primeiro jogador começa o jogo. A seguir joga o jogador à sua esquerda. Quando na vez de um jogador, este pode fazer uma de três coisas:

1. Adicionar 1, 2, 3, 4 ou 5 peças à grelha numa única linha reta e depois retirar do topo da sua pilha de cartas o número necesário para voltar a ficar com 5 na mão*. Se, numa das jogadas, o jogador completar um ou mais Q!ntos, este deve voltar a completar a sua mão* e deve jogar outra vez.
2. Trocar uma ou mais cartas da sua mão por wildcards já jogadas e/ou a carta Q!nto de forma a que as cartas trocadas continuem a funcionar naquela posição.
3. Passar a vez... e trocar algumas, todas ou nenhuma das susas cartas colocando-as no fundo da sua pilha e retirando do topo o número de cartas necessário para voltar a ficar com 5*.

*ou menos, se não houver cartas suficientes para perfazer 5.

O jogo termina quando houver 1 jogador que fica sem cartas, sendo este o vencedor.

Nota: Uma carta "wild" ou "Q!nto" pode representar cartas diferentes em diferentes direções. No caso de representar coisas diferentes e não ter na mão uma carta que também as consiga representar, então, essa carta é insubstituível.

# 4 Lógica de Jogo

## 4.1 Representação do estado de jogo

Embora não haja tabuleiro neste jogo, existe sempre a necessidade de saber a posição relativa entre as cartas que estão em jogo. Como tal, implementamos um tabuleiro na forma de lista de listas de cartas que funciona de forma dinâmica, ou seja, sempre que é introduzida uma nova carta é feita a verificação aos quatro lados do tabuleiro e adicionada uma linha/coluna se assim for necessário, por forma a garantir que existem sempre casas nas bordas do tabuleiro que estão livres para que o jogador as possa utilizar.

```
%%%%%%%%%%%%%%%%%%%%%%%% Check Board %%%%%%%%%%%%%%%%%%%%%%%%
checkBoardSize(Board, ResultBoard4) :-
    if(checkUpperLine(Board), addLineTop(Board, ResultBoard1), ResultBoard1 = Board),
    if(checkLowerLine(ResultBoard1), addLineBottom(ResultBoard1, ResultBoard2, _),
        ResultBoard2 = ResultBoard1),
    if(checkLeftColumn(ResultBoard2), ResultBoard3 = ResultBoard2, addColumnLeft(
        ResultBoard2, ResultBoard3)),
    if(checkRightColumn(ResultBoard3), ResultBoard4 = ResultBoard3, addColumnRight(
        ResultBoard3, ResultBoard4)).
```

Código - 1

O estado do jogo não é apenas representado pelo tabuleiro. A cada jogada é apresentada no ecrã a mão de que o jogador dispõe para jogar. Como sabemos ele apenas tem acesso a cinco das cartas do seu baralho, e durante a sua jogada não lhe são dadas mais cartas para além daquelas cinco.

```
%%%%%%%%%%% Prints N First Cards of the Deck %%%%%%%%
print5Cards(_, MaxNum, MaxNum, MaxNum).
print5Cards([], _, NumberOfCards, NRes) :-
    !, NRes is NumberOfCards.

print5Cards([DeckHead | DeckTail], MaxNum, NumberOfCards, NRes) :-
    write(NumberOfCards),
    write(' - '),
    printCard(DeckHead),
    nl,
    M is NumberOfCards + 1,
    print5Cards(DeckTail, MaxNum, M, NRes).

%%%%%%%%%%% Prints the first 5 cards %%%%%%%%%%
printFirstFive(Deck, NumberOfCardsLeft) :-
    write('\nFirst Five Cards are:\n\n'),
    print5Cards(Deck, 5, NumberOfCardsLeft).
```

Código - 2

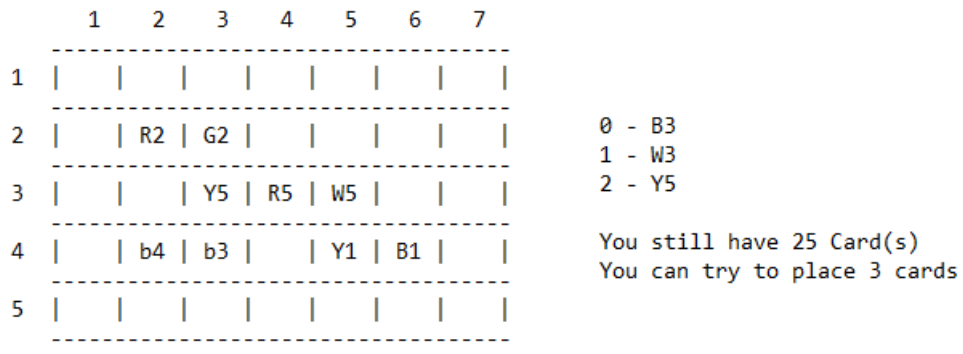


Figura 2: Print - 1

## 4.2 Visualização do Tabuleiro

Sendo o tabuleiro uma lista de listas de tamanho variável, os predicados responsáveis pela visualização do tabuleiro de jogo terminam quando a cauda da lista é uma lista vazia nas chamadas recursivas como mostrado abaixo.

```
%######################print all cells#####################%
printAllCells([A|[]], BoardHeight, I) :-
    I < BoardHeight,
    I1 is I + 1,
    startPrintLine(A, I1),
    nl,
    length(A, BoardLength),
    startPrintDivisor(1, BoardLength),
    nl.
printAllCells([A|B], BoardHeight, I) :-
    I < BoardHeight,
    I1 is I + 1,
    startPrintLine(A, I1),
    nl,
    length(A, BoardLength),
    startPrintDivisor(1, BoardLength),
    nl,
    printAllCells(B, BoardHeight, I1).
printAllCells(_,_,_).
getLength([L | _], BoardLength) :-
    length(L, BoardLength).

%######################printBoard##########################%
printBoard(Board) :-
    length(Board, BoardHeight),
    getLength(Board, BoardLength),
    printIdsOfColumns(BoardLength),
    nl,
    printAllCells(Board, BoardHeight, 0).
```

Código - 3

Alguns dos predicados utilizados não estão mostrados acima, como por exemplo o predicado responsável pela impressão dos índices das linhas, no entanto estes estão incluídos no código fonte em anexo a este relatório.

## 4.3 Execução de Jogadas

No início de cada jogada é perguntado ao jogador que tipo de jogada pretende executar. Como sabemos acima o jogador pode optar por adicionar cartas, trocar cartas já presentes no tabuleiro ou trocar cartas que tem na mão.

```
Choose what to do in your move

    1 - Add Cards
    2 - Swap with Wild Cards
    3 - Pass... and trade
```

Figura 3: Print - 2

### 4.3.1 Adicionar Cartas

Depois de optar por adicionar cartas no tabuleiro, é mostrado no ecrã o estado atual do tabuleiro e as cartas que o jogador tem na mão. O jogador opta pelas cartas que quer jogar bem como a casa

do tabuleiro onde pretende que as cartas sejam colocadas. Depois de introduzir esses dados é feita a verificação das condições que têm de ser cumpridas:

- Verifica se a casa está vazia;
- Verifica se tem pelo menos uma carta nas casas adjacentes (excepto no caso de ser a primeira jogada);
- Verifica se, com a carta colocada, a linha fica com menos de seis cartas;
- Verifica também se as regras relacionadas com as cores e naipes se verificam na linha;
- Verifica se, com a carta colocada, a coluna fica com menos de seis cartas;
- Verifica também se as regras relacionadas com as cores e naipes se verificam na coluna.

```
1  (checkFirstPlay(Board), Check is 0);
2          (
3              checkIfACardIsInTheSpot(SelectedLine, SelectedColumn, Board),
4              getCoordinatesAroundSpot(Line1, Line2, Column1, Column2, SelectedLine,
                   SelectedColumn, Board),
5              (
6                  \+((checkIfACardIsInTheSpot(Line1, SelectedColumn, Board)));
7                  \+((checkIfACardIsInTheSpot(SelectedLine, Column1, Board)));
8                  \+((checkIfACardIsInTheSpot(Line2, SelectedColumn, Board)));
9                  \+((checkIfACardIsInTheSpot(SelectedLine, Column2, Board)))
10             ),
11             checkIfLessThanFiveInLine(SelectedLine, SelectedColumn, Board, RemovedCard),
12             checkIfLessThanFiveInColumn(SelectedLine, SelectedColumn, Board, RemovedCard
                   )
13         ) -> write('\nPosition Accepted\n'), Col is SelectedColumn, Line is SelectedLine
               , SelectedCard is SelectedCardTemp, Check is 0;
14         write('\nPosition unaccepted\n'),
```

Código - 4

Acima encontra-se parte do predicado responsável pelas verificações mencionadas que garantem que todas as regras são cumpridas.

### 4.3.2 Trocar Cartas

Depois de optar por alterar cartas presentes no tabuleiro, tal como no caso acima, é mostrado no ecrã o estado atual do tabuleiro bem como as cartas de que o jogador dispõe. Depois de optar pelas cartas e casas nas quais querem jogar são feitas as seguintes verificações:

- Verifica se a carta escolhida pelo jogador é *Wild*, seja de côr ou naipe;
- Verifica se a casa escolhida pelo utilizador está, de facto, ocupada;
- Verifica se as regras referentes ao número, cores e naipes da sequência são cumpridas na linha;
- Verifica se as regras referentes ao número, cores e naipes da sequência são cumpridas na coluna.

```
1  (checkFirstPlay(Board), Check is 1);
2          (
3              checkIfWild(CardOnPlace),
4              \+((checkIfACardIsInTheSpot(SelectedLine, SelectedColumn, Board))),
5              checkIfLessThanFiveInLine(SelectedLine, SelectedColumn, Board, RemovedCard),
6              checkIfLessThanFiveInColumn(SelectedLine, SelectedColumn, Board, RemovedCard
                   )
7          ) -> write('\nSwap Position Accepted\n'), Col is SelectedColumn, Line is
               SelectedLine, SelectedCard
```

Código - 5

Acima encontra-se parte do predicado responsável pelas verificações mencionadas que garantem que todas as regras são cumpridas.

### 4.3.3 Trocar cartas

Ao optar por trocar cartas é dada oportunidade ao jogador para escolher quais as cartas (dentro daquelas que estão nesse momento na sua mão) que este pretende trocar. Após a escolha estas cartas passam para o fim do baralho e são repostas na mão por aquelas que estão imediatamente no topo do baralho.

```
swapCardsInDeck(Deck_1, ResultDeck_1) :-
    chooseToSwap(Deck_1, ResultDeck_1, 5).

chooseToSwap(Deck, ResultDeck_1, 0) :-
    ResultDeck_1 = Deck.
chooseToSwap(Deck, ResultDeck_1, N) :-
    print5Cards(Deck, N, 0, NumberOfCards),
    getSelection0(SelectedCardTemp, NumberOfCards),
    removeCardFromDeck(Deck, ResultDeck, SelectedCardTemp, 0, RemovedCard),
    append(ResultDeck, [RemovedCard], TempDeck),

    write('\nDo you want to try swap more (0. - Yes | 1. - No)? '),
    get_char(Answer),
    get_char(_),
    if(Answer == '0', N1 is N - 1, N1 is 0),
    chooseToSwap(TempDeck, ResultDeck_1, N1).
```

Código - 6

## 4.4 Avaliação do estado do Jogo

Tendo em conta que nesta modalidade o jogo termina quando um dos jogadores termina o seu baralho, podemos avaliar o estado do jogo através do número de cartas restantes nos baralhos dos jogadores que são impressos nas jogadas.

## 4.5 Final do Jogo

O jogo termina mal um dos jogadores termine com o seu baralho, tendo sido bem sucedido em deixar todas as suas cartas no tabuleiro sem infringir nenhuma das regras. No final de cada jogada é feita a verificação de se o baralho do jogador está, ou não, vazia. No caso do baralho estar vazio termina o jogo imediatamente.

```
checkVictory([Deck | _], Check) :-
    chec(Deck, Check).
chec([C | _], Check) :-
    if(nonvar(C), Check is 1, Check is 0).
```

Código - 7

# 5 Interface com o Utilizador

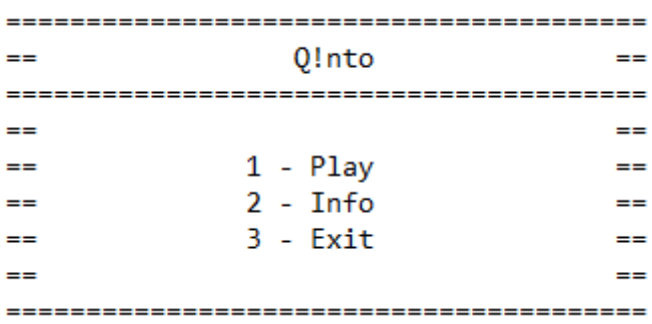


Figura 4: Print - 3 - Menu Principal

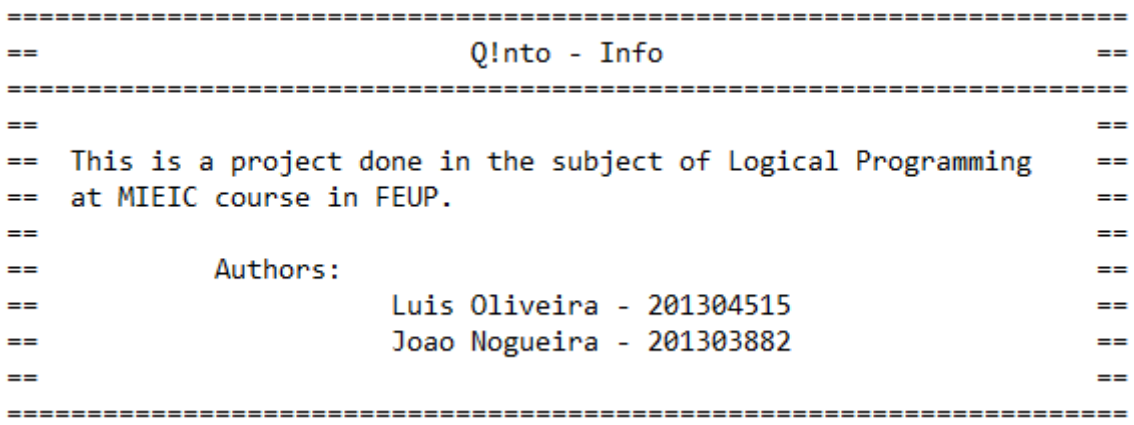


Figura 5: Print - 4 - Info

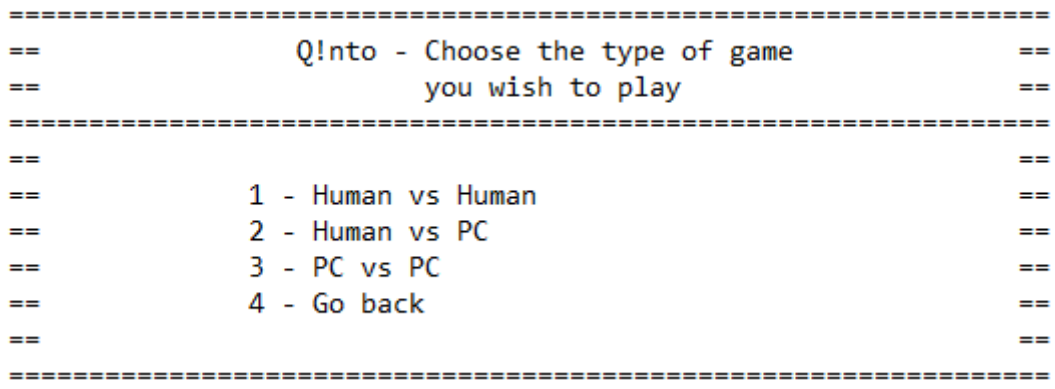


Figura 6: Print - 5 - Menu de Jogo


Figura 7: Print - 6 - Início do Jogo

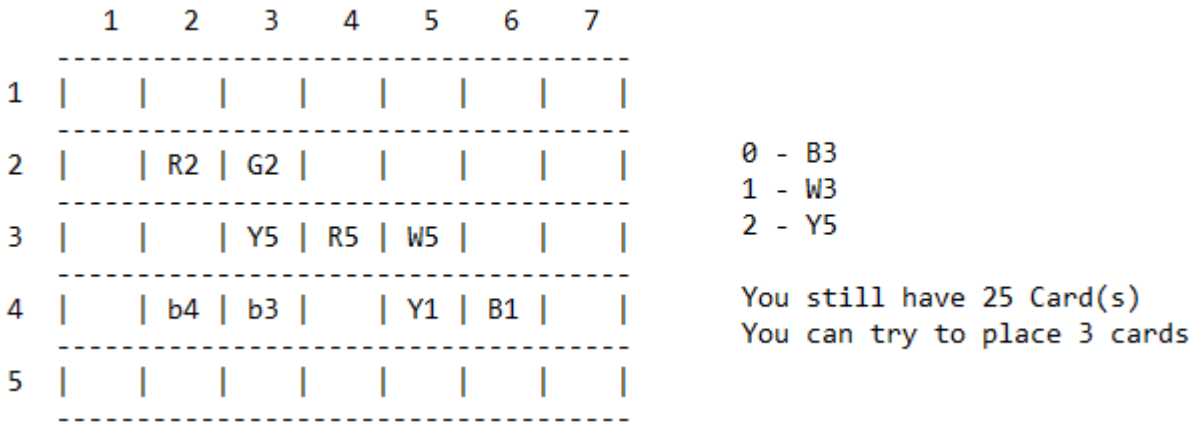


Figura 8: Print - 7 - Meio do Jogo

```
0 - B1

You still have 1 Card(s)
You can try to place 1 cards

Choose the Card to Play (end the number with"."): 0.
In What Line do you want to put it: 17.
In What Column do you want to put it: 1.

Position Accepted

No more cards to play. Turn ended.
```

Figura 9: Print - 8 - Final do Jogo

```
================================================================
==                        PLAYER 1 WINS                       ==
================================================================
```

Figura 10: Print - 9 - Vitória do Jogador 1

# 6 Conclusões

Este trabalho realizado no âmbito da Unidade Curricular de Programação em Lógica serviu positivamente para que ganhássemos bastantes conhecimentos ao nível da linguagem de PROLOG. Achamos que, tendo um pouco mais de tempo para a conclusão do trabalho, este seria concluído de melhor forma. Mesmo assim o balanço que fazemos é bastante positivo.

Surpreendeu-nos a dificuldade de algumas das regras exigidas por este jogo, bem como a necessidade de fazer um tabuleiro de forma dinâmica, no entanto, encaramos estas dificuldades como desafios que fizemos por superar e fizemo-lo com sucesso. Infelizmente não tivemos tempo para implementar o *bot*, melhoria que proporíamos num futuro em que pudéssemos melhorá-lo.