# Monkey Queen

## Relatório Final


Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

**Monkey_Queen_1:**

José Miguel Costa – 201402717

Luís Miguel Gonçalves – 201207141

Faculdade de Engenharia da Universidade do Porto

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

13 de Novembro de 2016

# Resumo

O jogo escolhido pelo grupo foi o “Monkey Queen”. Neste jogo temos um tabuleiro quadrado com uma dimensão de 12x12, e as peças possíveis são: a rainha (uma stack de peças que começa com 20) e os bebés. As peças podem ser pretas ou brancas. Os movimentos de todas as peças são como os da rainha no xadrez (qualquer número de casas em qualquer direção), e se a rainha se move sem capturar deixa para trás uma das peças da sua stack, dando origem a um bebé. As peças são capturadas por substituição, como no xadrez, e se uma rainha captura um bebé inimigo este vai para a sua stack. O jogo acaba quando uma das rainhas for capturada ou ficar sem peças.

Implementamos um jogo, principalmente concentrado no modo de jogador contra jogador com uma interface simples e fácil de perceber. Apesar de algumas dificuldades tiramos um balanço positivo do resultado final, mas mesmo assim com espaço para melhorias.

# Conteúdo

# 1. Introdução

Na unidade curricular Programação em Lógica foi-nos proposta a realização de um projeto que consiste na implementação de um jogo de tabuleiro, a escolher de uma lista predefinida, em ProLog. Escolhemos o jogo Monkey Queen por ser um jogo que nos parece familiar e interessante, fácil de aprender e divertido de jogar.

Neste relatório iremos começar por falar do jogo, da sua história e regras, seguido da lógica que implementámos referindo-nos às funções utilizadas e esclarecendo melhor as regras. Por fim, apresentaremos a interface e concluímos o relatório com a conclusão e bibliografia usada.

# 2. O Jogo Monkey Queen

Monkey Queen é um jogo de dois jogadores, jogado num tabuleiro 12x12 e foi concebido em 2011 por Mark Steere. Inicialmente o tabuleiro tem duas rainhas (uma pilha de 20 peças pretas e outra de brancas). Os dois jogadores fazem jogadas à vez que consistem em mexer a rainha, uma pilha por turno. O objetivo do jogo é matar a rainha inimiga ou deixar o adversário sem movimentos possíveis.

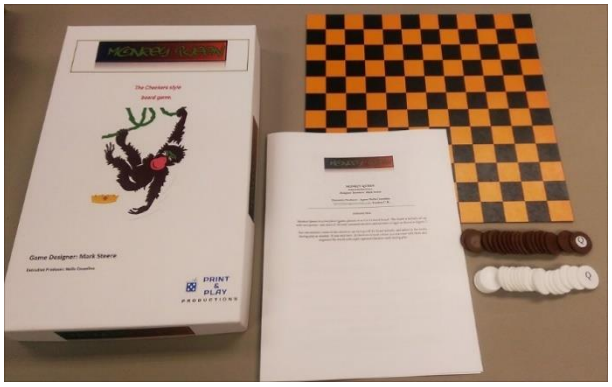


*Figura 1 - Versão Física do Jogo*

# 3. Lógica do Jogo

## a. Representação do Estado do Jogo

Como referido acima o tabuleiro é quadrado, sendo que cada lado tem a largura de 12 células, assim sendo a abordagem que considerámos mais apropriada foi criar uma lista de 12 listas, em que cada lista representa uma linha do tabuleiro cada uma com 12 elementos. O tabuleiro é declarado da seguinte maneira, representando o seu estado inicial:

```
initialBoard([[0,0,0,0,0,b-20,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,w-20,0,0,0,0,0]]).
```

*Figura 2 - Representação do tabuleiro inicial*

Cada posição da lista a **0** representa uma casa vazia no tabuleiro. As duas posições com o valor **20** representam as duas posições inicias das rainhas, sendo que 20 é o número de peças

```
[[0,0,0,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,0,0],
 [w-1,0,0,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,b-1,0,0,0],
 [0,0,w-7,0,0,0,b-1,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,0,b-9,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,0,0]]).
```

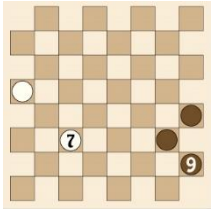

*Figura 4 - Visualização Prolog e representação real (Posição Intermédia)*

```
[[0,0,0,0,0,0,0,b-1,0,b-1,0,b-1],
 [0,0,0,0,0,b-1,0,b-1,b-1,0,0,b-6],
 [0,0,0,0,0,b-1,0,0,0,0,b-1,0],
 [0,0,0,0,0,0,0,0,w-1,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,w-1,0,0,0,w-1,0],
 [0,0,0,0,w-9,0,0,0,0,0,0,w-1],
 [0,0,0,0,0,w-1,0,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,0,0]]).
```

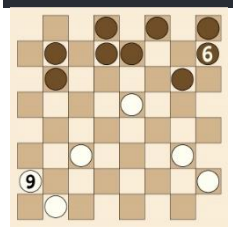

*Figura 3 - Visualização Prolog e representação real (Posição Final)*

da pilha. Os caracteres **b** e **w** são indicativos da cor das peças, sendo preto e branco respetivamente.

A figura da esquerda representa um estado intermédio do tabuleiro. As posições onde **1** é o número na segunda parte do par, representam um bebé deixado pela rainha, e as outras duas posições com um número superior a 1 representam a rainha sendo o número a quantidade de peças na pilha.

A figura da direita representa um estado final do tabuleiro, onde o jogador que usa as peças pretas irá perder na próxima jogada, pois não pode fazer mais movimentos com a rainha sem esta ser capturada na jogada do oponente.

## b. Visualização do Tabuleiro

Por forma a imprimir a lista foram declarados predicados em Prolog e chegámos a uma representação que achamos adequada e percetível para jogador:

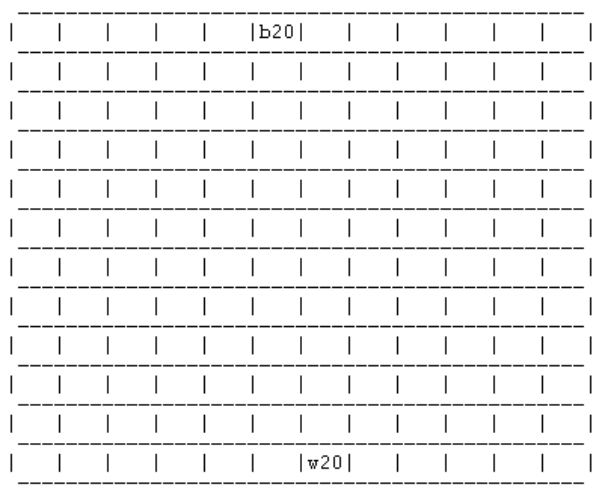


*Figura 5 - Tabuleiro impresso em ASCII*

Principais predicados para imprimir o tabuleiro:

- printBoard([]):-

  printLine(x).

  printBoard([H|T]):-

  printLine(x),

  printSpaces(H),

  printBoard(T).

Em que a lista é passada como argumento ([H-T]) e a função printLine é responsável por desenhar as linhas horizontais do tabuleiro e printSpaces por desenhar as linhas verticais e por imprimir as peças no tabuleiro.

Por cada elemento da lista recebida por printSpaces é chamada a função translatePrint em que consoante o tamanho e cor da peça representa-a no tabuleiro com uma dimensão constante de 3 caracteres, como é possível visualizar abaixo:

- translatePrint(0):-

  write('  ').

  translatePrint(Colour-Char):-

  Char < 10,

  write('  '),

  write(Char).

  translatePrint(Colour-Char):-

  Char >= 10,

  write(Colour),

  write(Char).

## c. Execução de Jogadas

A jogada de cada jogador é obtida com o predicado:

- playerMove(Board, Player, Victory, NextBoard)

Em que o jogador dá as coordenadas da peça que pretende mover e as coordenas do destino e com essas coordenadas executamos o predicado:

- tryToMovePiece(PlayerChar, Board, FX-FY, TX-TY, NextBoard, Victory)

Que verifica se as coordenadas que o utilizador inseriu, tanto da posição em que a peça está como a posição para onde vai, e se a peça escolhida são válidas com os predicados:

- validateFromPosition(Player, BoardState, FX-FY)
- validateToPosition(Player, BoardState, FX-FY, TX-TY)
- checkIfBelongsToPlayer(Player, Piece)

Onde começamos por validar a posição inicial, depois a posição final e se a peça pertence ao jogador certo. Ao validar a posição de destino, verificamos os três movimentos possíveis (horizontal, vertical e diagonal) com os predicados:

- horizontalMovement(Player, BoardState, FX-FY, TX-TY, FXNew-FYNew)
- verticalMovement(Player, BoardState, FX-FY, TX-TY, FXNew-FYNew)
- diagonalMovement(Player, BoardState, FX-FY, TX-TY, FXNew-FYNew)

Dependendo se a peça se movimenta para capturar ou não e se se trata da rainha, pode deixar para trás um bebé ou não.

- dropChild(DropPiece, Piece, FX-FY, FinalBoard2, BoardWithChild)

Se capturar come a peça na posição de destino.

- eatPiece(Player, Piece, [H|T], FX-FY, TX-TY, Board, It, Victory, FinalBoard, DropPiece)

E em qualquer situação, em caso de jogada válida, substitui o que está no destino, quer seja uma peça ou uma casa vazia.

- replacePieceAtPosition(Colour-Char, [H|T], FX-FY, Board, It, FinalBoard)

### d. Final do Jogo

O final do jogo, como referido acima, é alcançado quando a rainha inimiga é comida. Ao tentar mover uma peça com o predicado tryToMovePiece, temos uma variável *Victory* que caso esta seja igual a 1, indica que o jogo chegou ao fim. Esta variável é alterada com o predicado:

- checkEndGame(Colour-Char, Piece, Victory,DropPiece, NewPiece)

Ao executar o predicado eatPiece se a rainha inimiga for capturada o valor de *Victory* é alterado para 1 e o jogo acaba com uma mensagem indicativa.

### e. Jogada do Computador

A jogada do computador está pouco elaborada, visto termo-nos concentrado mais na vertente jogador contra jogador do projeto. Assim sendo a jogada do computador é obtida com o predicado:

- pcMove(Board, Player, Victory, NextBoard)

Semelhante ao predicado anterior playerMove, mas as coordenadas são obtidas aleatoriamente, tanto as da posição da pesa como da posição de destino. Isto gera o problema de poder demorar infinitamente a encontrar uma jogada.

## 4. Interface com o Utilizador

A interface está implementada de forma intuitiva para o utilizador. Começa por mostrar um menu ao utilizador com as opções de jogador vs jogador, jogador vs pc ou sair do jogo. Após o utilizador escolher é mostrado o tabuleiro inicial, indicado qual dos dois jogadores é a jogar e são pedidas as coordenadas da peça que se pretende mover e as coordenadas do seu destino. Até ao final do jogo é este o sistema e quando o jogo termina o jogador vencedor é congratulado e o jogo termina.

```
  Welcome to Monkey Queen

Select any of the options bellow:
1) 1 vs 1
2) 1 vs PC
0) Exit

Option:
```

*Figura 6 - Menu Inicial*

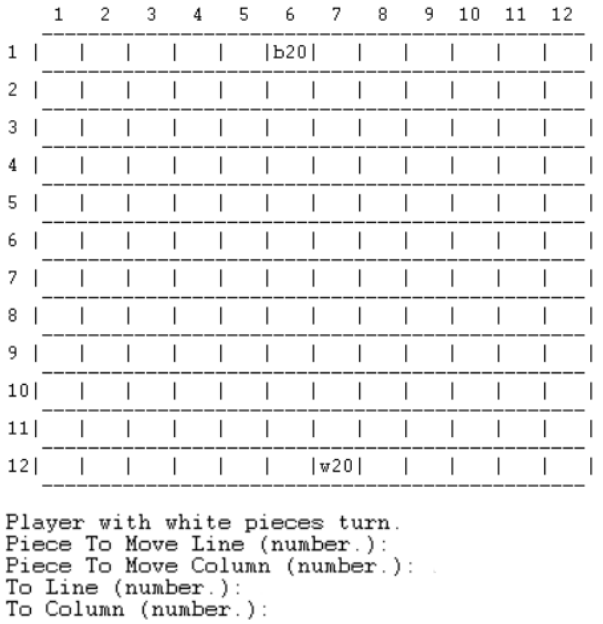


*Figura 7 - Inserir Coordenadas*

A interface está implementada nos seguintes predicados:

- game(Board)
    - Responsável por chamar a função de menu e, consoante o input do utilizador, chama o controlador do jogo. Este, por sua vez,
- mainMenu(Option)
    - Imprime o menu e espera pelo input do utilizador.

## 5. Conclusões

Com a realização deste projeto foi-nos possível interiorizar melhor alguns conteúdos aprendidos nas aulas teóricas e práticas de PLOG, relativamente à linguagem ProLog. Com um pouco mais de tempo o trabalho teria sido concluído de melhor forma, mas o balanço que tiramos é bastante positivo.

Relativamente a melhorias, estas seriam principalmente na interface e na implementação das jogadas do computador. Sendo que a segunda implicaria encontrar todas as peças do computador, encontrar todas as jogadas possíveis para cada peça, dar uma pontuação a cada jogada consoante o resultado de efetuar essa mesma jogada e por fim executar a jogada com melhor pontuação.