**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 que escolhemos 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.

Para implementar o tabuleiro usamos uma lista de listas, em que cada elemento da primeiro representa uma linha do tabuleiro e os elementos da segunda representam as casas dentro dessa mesma linha.

As peças são identificadas por um par (ex: b-20), em que o primeiro elemento é ou “b” ou “w” representando a cor da peça (black ou white), e o segundo elemento representa, na caso de ser superior a 1, a rainha respetiva, e o número de peças da sua stack. Se a casa estiver vazia é ocupado por um “0”.

Os movimentos foram implementados da seguinte forma, é pedido ao utilizador a localização da peça que quer mover e de seguida a casa para onde a quer mover, e depois uma série de funções verifica se a jogada é valida e se o for executa-a.

**Conteúdo**

1. **Introdução**

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

Neste relatório começamos por falar do jogo, da sua história e regras, por alto, seguido da lógica que implementamos referindo-nos às funções utilizadas e esclarecendo melhor as regras. Por fim, apresentamos a interface e concluímos o relatório com a bibliografia usada.

1. **O Jogo Monkey Queen**

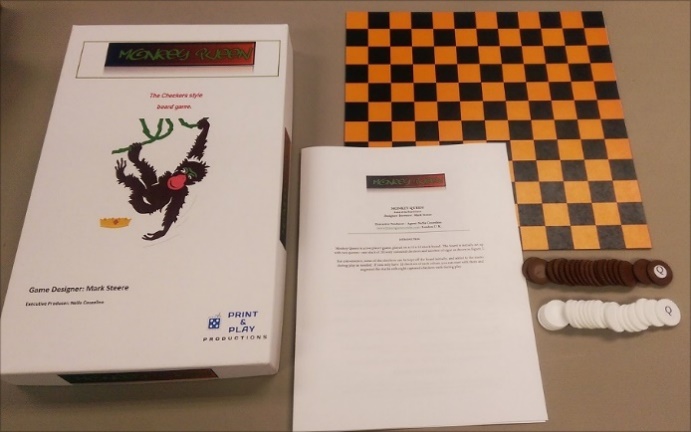
Monkey Queen é um jogo de dois jogadores, jogado num tabuleiro 12x12 e foi concebido em 2011 por Mark Steere. Inicialmente o tabuleiro tem duas rainhas (uma pilha de 20 peças pretas e outra de brancas). Os dois jogadores fazem jogadas à vez que consistem em mexer a rainha, uma pilha por turno. O objetivo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

Figura 1 - Versão Física do Jogo

1. **Lógica do Jogo**
   1. **Representação do Estado do Jogo**

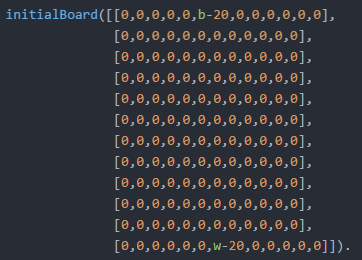
****Como referido acima o tabuleiro é quadrado, sendo que cada lado tem a largura de 12 células, assim sendo a abordagem que considerámos mais apropriada foi criar uma lista de 12 listas, em que cada lista representa uma linha do tabuleiro cada uma com 12 elementos. O tabuleiro é declarado da seguinte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

Figura 2 - Representação do tabuleiro inicial

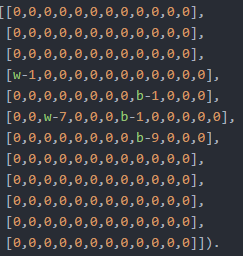
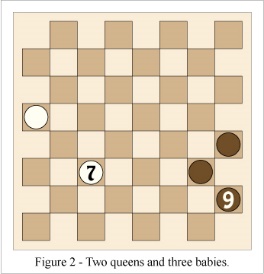
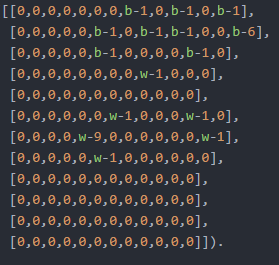
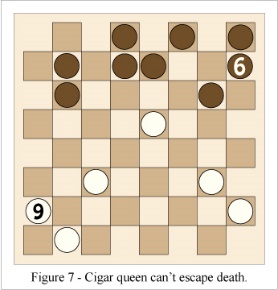
Cada posição da lista a **0** representa uma casa vazia no tabuleiro. As duas posições com o valor **20** representam as duas posições inicias das rainhas, sendo que 20 é o número de peças da pilha. Os caracteres **b** e **w** são indicativos da cor das peças, sendo preto e branco respetivamente.

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

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

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.

* 1. **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:

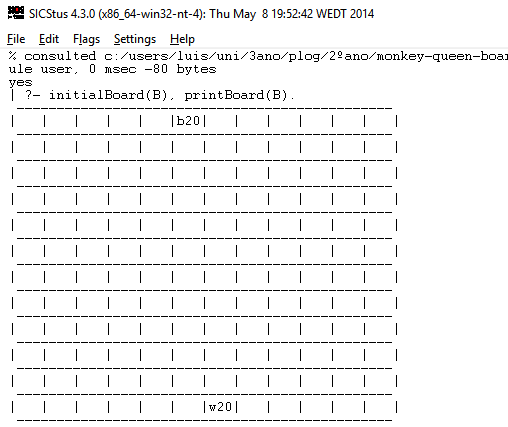


Figura 5 - Tabuleiro impresso em ASCII

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).

* 1. **Execução de Jogadas**

A jogada de cade 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 essa 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, 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, 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 substitui o que está no destino, quer seja uma peça ou uma casa vazia.

* replacePieceAtPosition(Colour-Char, [H|T], FX-FY, Board, It, FinalBoard)
  1. **Avaliação do Tabuleiro**
  2. **Final do Jogo**

Ao tentar mover uma peça com o predicado tryToMovePiece, temos uma variável chamada *Victory* que caso esta seja igual a 1, indica que o jogo chegou ao fim. Esta variável é alterado com o predicado:

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

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

* 1. **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.

1. **Interface com o Utilizador**
2. **Conclusões**

**Bibliografia**