Monkey Queen

Relatório Final



Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo 4:

Marta Diogo Torgal Pinto - up201407727

Telmo João Vales Ferreira Barros - up201405840

Faculdade de Engenharia da Universidade do Porto

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

1. de Novembro de 2016

## Resumo

O jogo consiste num jogo de tabuleiro, como tal foi criado um tabuleiro e guardado numa lista. Existindo 2 tipos de jogadores (humano/computador), para o humano eram pedidas as posições iniciais e finais da peça a mover. Quando um humano faz uma jogada e se esta não for considerada uma das melhores jogadas é feito um display com a melhor jogada que devia ter sido realizada.

Para o computador era gerada uma lista de movimentos possíveis e legais a realizar por cada peca do jogador presente no tabuleiro, de seguida cada movimento foi avaliado de 1 a 5, e consoante o modo de jogo uma peça era escolhida e era efetuado o movimento da mesma. Um movimento legal consiste num movimento na vertical, horizontal ou diagonal em que nenhuma peça esteja entre as posições iniciais e finais. Neste jogo como não é so possível o computador jogar contra outro computador, e este mesmo jogo acabar. Como também é possível um humano jogar contra um computador.

Para o jogo terminar um dos jogadores ou não teria mais nenhum movimento possível, ou a sua rainha teria sido comida, tais condições são verificadas e o jogo acaba dizendo qual foi o vencedor.

## Índice

[Resumo 2](#_Toc466801103)

[Índice 3](#_Toc466801104)

[Introdução 4](#_Toc466801105)

[Descrição do Jogo 5](#_Toc466801106)

[Movimentos 5](#_Toc466801107)

[Lógica do Jogo 6](#_Toc466801108)

[Representação do Estado do Jogo 6](#_Toc466801109)

[Visualização do Tabuleiro 7](#_Toc466801110)

[Lista de Jogadas Válidas 8](#_Toc466801111)

[Execução de Jogadas 8](#_Toc466801112)

[Avaliação do Tabuleiro 9](#_Toc466801113)

[Final do Jogo 9](#_Toc466801114)

[Jogada do Computador 9](#_Toc466801115)

[Interface com o utilizador 11](#_Toc466801116)

[Conclusões 12](#_Toc466801117)

[Referências 13](#_Toc466801118)

[Anexos 14](#_Toc466801119)

## Introdução

O objetivo deste trabalho passou pelo desenvolvimento de um jogo de tabuleiro, Monkey Queen, que exigiu a utilização e manipulação de listas bidimensionais assim como programação declarativa.

O relatório surge como um complemento ao trabalho prático, uma forma de suporte escrito para que qualquer um perceba o trabalho desenvolvido. Para cumprir este objetivo da melhor forma, o relatório está dividido nas seguintes secções:

- **Descrição do jogo**: breve descrição do jogo, história e regras;

- **Lógica do jogo**: explicação detalhada da lógica por detrás da implementação do jogo, representação do estado de jogo, visualização do tabuleiro e peças, validação de jogadas e listagem das jogadas válidas, avaliação do tabuleiro, verificação da condição de paragem do jogo e respetivo vencedor, escolha de jogada por parte do computador;

- **Conclusões**: conclusões retiradas da elaboração do projeto.

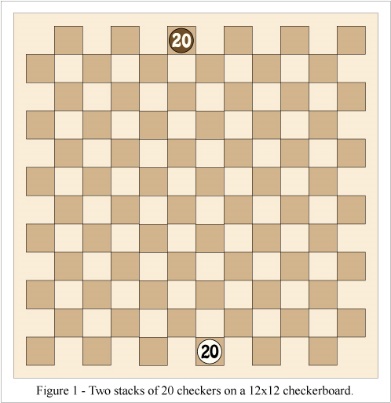
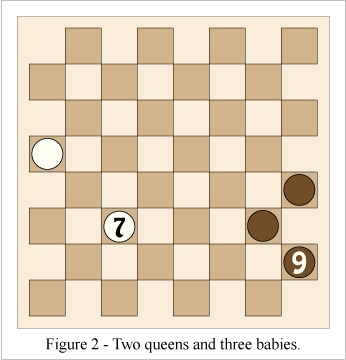
## Descrição do Jogo

Monkey Queen foi idealizado em Janeiro de 2011 por Mark Steere, sendo considerado por este um jogo abstrato, no qual dois jogadores competem num tabuleiro 12x12. O objetivo deste jogo é eliminar a rainha adversária ou deixar a mesma sem movimentos possíveis.

As jogadas são feitas alternadamente entre os jogadores, de cor Branco e Preto. O jogador 1 é o primeiro a jogar e joga com as peças brancas, o adversário joga com as peças pretas. No entanto, pode ser aplicada a *Pie Rule* após a primeira jogada, se o segundo jogador assim o entender, isto é, os jogadores trocam de posição e o jogador 2 passa a ser o jogador 1. O “novo” jogador 2 faz agora a sua “primeira” jogada no controlo das peças pretas.

O jogo começa com as duas rainhas nas colunas F e G de lados opostos do tabuleiro. Cada rainha é composta por uma pilha com 20 peças da sua cor.

No decorrer do jogo cada jogador terá exatamente uma rainha, que será a peça com pelo menos duas peças empilhadas, e poderá ter um ou mais bebés que são as peças unitárias da mesma cor que a sua.

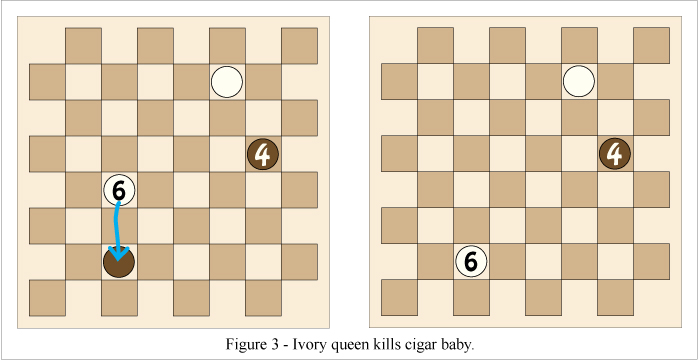


### Movimentos

Quanto à rainha, esta pode se mover no tabuleiro como num jogo de xadrez, isto é, pode mover a pilha toda que a constituí em qualquer direção (horizontal, vertical e diagonal) numa sequência de casas vazias.

Se o movimento terminar numa casa ocupada por uma peça adversária, rainha ou bebé, a mesma é capturada e substituída pela rainha. Se o movimento da rainha não originar a captura de nenhuma peça adversária a mesma deixa na sua posição anterior um bebé, reduzindo a altura da sua pilha numa unidade.

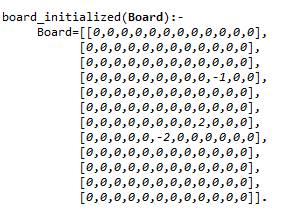
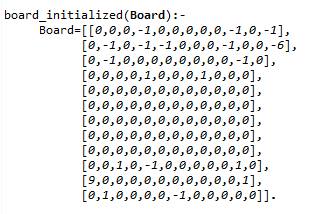
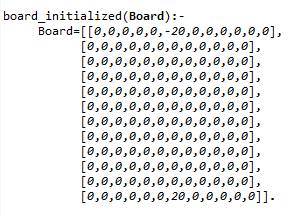
Quanto aos bebés, estes podem-se mover exatamente da mesma forma da rainha para capturar a rainha ou bebés adversários.

Quando o seu movimento não originar a captura de nenhuma peça adversária, a sua distância em linha reta em relação à rainha adversária tem de ser encurtada.

Não é obrigatório proceder à captura de peças adversárias ainda que exista essa possibilidade.

## Lógica do Jogo

### Representação do Estado do Jogo

Para representar o estado atual do jogo decidimos utilizar uma lista de listas que correspondem às diferentes linhas do tabuleiro. As peças e os espaços vazios são representados por números. Os números negativos representam as peças pretas e os positivos as peças brancas, o valor absoluto do número corresponde ao número de peças empilhadas (no caso das rainhas). Os espaços vazios são representados pelo número 0.

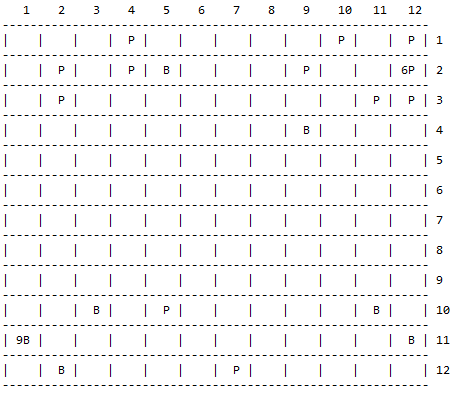
Estado Final

Estado Intermédio

Estado Inicial

### Visualização do Tabuleiro

O output do tabuleiro de jogo é o seguinte:



Para a sua visualização utilizamos os seguintes predicados:

display\_board([L1|Ls]):-

write(' A B C D E F G H I J K L'), nl,

write('-------------------------------------------------------------'), nl,

display\_lines([L1|Ls], *1*).

display\_lines([L1|Ls], Nlines):-

display\_line(L1), write('| '), write(Nlines), nl,

write('-------------------------------------------------------------'), nl,

Nextlines is Nlines + *1*,

display\_lines(Ls,Nextlines).

display\_lines([], Nlines).

display\_line([E|Es]):-

write('|'),

translate(E,T),

write(T),

display\_line(Es).

display\_line([]).

translate(*0*,' ')**.**

translate(*-20*,' 20P')**.**

translate(*-19*,' 19P')**.**

translate(*-18*,' 18P')**.**

translate(*-17*,' 17P')**.**

translate(*-16*,' 16P')**.**

translate(*-15*,' 15P')**.**

translate(*-14*,' 14P')**.**

translate(*-13*,' 13P')**.**

translate(*-12*,' 12P')**.**

translate(*-11*,' 11P')**.**

translate(*-10*,' 10P')**.**

translate(*-9*,' 9P ')**.**

translate(*-8*,' 8P ')**.**

translate(*-7*,' 7P ')**.**

translate(*-6*,' 6P ')**.**

translate(*-5*,' 5P ')**.**

translate(*-4*,' 4P ')**.**

translate(*-3*,' 3P ')**.**

translate(*-2*,' 2P ')**.**

translate(*-1*,' P ')**.**

translate(*20*,' 20B')**.**

translate(*19*,' 19B')**.**

translate(*18*,' 18B')**.**

translate(*17*,' 17B')**.**

translate(*16*,' 16B')**.**

translate(*15*,' 15B')**.**

translate(*14*,' 14B')**.**

translate(*13*,' 13B')**.**

translate(*12*,' 12B')**.**

translate(*11*,' 11B')**.**

translate(*10*,' 10B')**.**

translate(*9*,' 9B ')**.**

translate(*8*,' 8B ')**.**

translate(*7*,' 7B ')**.**

translate(*6*,' 6B ')**.**

translate(*5*,' 5B ')**.**

translate(*4*,' 4B ')**.**

translate(*3*,' 3B ')**.**

translate(*2*,' 2B ')**.**

translate(*1*,' B ')**.**

### Lista de Jogadas Válidas

valid\_moves(+Board, +Player, -ListOfMoves).

Este predicado percorre todas as posições do tabuleiro e verifica todos os movimentos possíveis para todas as peças do Player, retornando uma lista com listas de movimentos no formato [Xinicial, Yinicial, Xfinal, Yfinal].

Recorre ao predicado legal\_move que valida o movimento a efetuar.

legal\_move(+Player, +Board, +Xinitial, +Yinitial, +Xfinal, +Yfinal)

Este predicado, recorrendo a predicados auxiliares, verifica:

* se a posições inicial (Xinitial, Yinitial) e a final (Xfinal, Yfinal) estão dentro do tabuleiro
* se a orientação do movimento desejado é válida (vertical, horizontal ou diagonal)
* se a peça na posição inicial corresponde a uma peça do jogador
* se as casas entre a posição inicial e final se encontram vazias

Além das verificações enumeradas existem verificações específicas como a aproximação do bebé à rainha adversária quando este não captura nenhuma peça e a obrigação da rainha capturar uma peça quando se encontra com apenas duas peças na pilha.

### Execução de Jogadas

move(+Player, +Board, +Xinitial, +Yinitial, +Xfinal, +Yfinal, -NewBoard)

O predicado move executa a jogada da peça na posição inicial (Xinitial, Yinitial) para a posição final (Xfinal, Yfinal). Quando se trata do movimento de uma rainha e a mesma se move para uma casa vazia é deixado um bebé na sua posição inicial e o seu valor é decrementado de uma unidade.

### Avaliação do Tabuleiro

value(+PreviousBoard, +Board, +Player, -Value)

O predicado value avalia uma jogada tendo por base o tabuleiro anterior e o novo após a jogada ser efetuada. As valorizações do novo tabuleiro foram distinguidas da seguinte forma:

5- O jogo termina com a **vitória do Player**.

4- O Player **captura um bebé** adversário e deixa a sua **rainha protegida** (sem peças adversárias com a possibilidade de a capturar).

3- O Player **não captura um bebé** adversário e deixa a sua **rainha protegida**.

2- O Player **captura um bebé** adversário e deixa a sua **rainha desprotegida**.

1- O Player **não captura um bebé** adversário e deixa a sua **rainha desprotegida**.

### Final do Jogo

game\_over(+Board, -Winner)

O final do jogo é verificado pelo predicado game\_over que retorna um Winner em caso de ser true. Este predicado verifica se ambas as rainhas estão em jogo e se ambos os jogadores tem movimentos válidos, o predicado retorna um Winner quando uma destas condições não é verificada.

### Jogada do Computador

choose\_move(+Difficulty, +ListOfMoves, +ListOfValues, -Xinitial, -Yinitial, -Xfinal, -Yfinal)

Nesta implementação foram introduzidos 4 níveis de dificuldade:

1- Easy

2- Medium

3- Hard

4- Expert

A escolha do movimento a efetuar pelo computador é feita no predicado choose\_move. Em qualquer nível de dificuldade o computador irá usar um movimento com value 5 se assim tiver possibilidade. O nível Expert irá sempre efetuar a melhor jogada possível enquanto que os restantes modos apresentam as probabilidades abaixo indicadas para escolha do valor da jogada a efetuar:

**Easy**

Value 4 - 20%

Value 3 - 20%

Value 2 - 30%

Value 1 - 30%

**Medium**

Value 4 - 30%

Value 3 - 30%

Value 2 - 20%

Value 1 - 20%

**Hard**

Value 4 - 50%

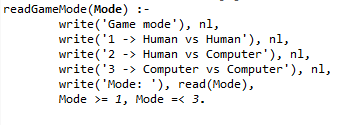
Value 3 - 30%

Value 2 - 10%

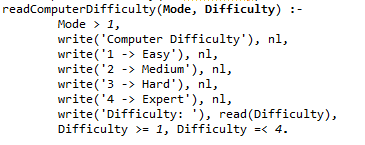
Value 1 - 10%

## Interface com o utilizador

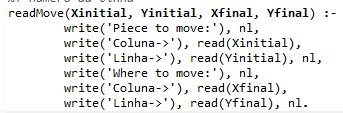
A interface com o utilizador foi implementada com o uso do predicado read. É dada ao utilizador a possibilidade de escolher o modo de jogo:



Escolher a dificuldade do computador quando o modo de jogo escolhido é 2 ou 3:



Escolher a posição da peça que quer jogar e a posição para onde quer jogar:



## Conclusões

Com este projeto é possível concluir que a linguagem Prolog se distingue claramente de outras linguagens de programação como C, C++ ou Java e que a preparação e planeamento de cada predicado é um fator muito importante. Esta linguagem é muito útil porque permite a aplicação de leis do jogo de forma clara, pelo que a passagem do modo humano para o modo computador foi rápida visto que já todos os predicados que validavam o movimento estavam implementados.

Todos os aspetos propostos foram concluídos e foi ainda introduzido um modo que permite ao jogador humano melhorar o seu jogo a partir de dicas por parte do computador da melhor jogada possível. Contudo o projeto poderia ser melhorado com uma avaliação do tabuleiro mais aprofundada que iria analisar os possíveis movimentos do adversário após a jogada testada. No entanto, este requisito iria exigir um nível de processamento muito mais elevado.

## Referências

Mark Steere. 2011. Monkey Queen. Acedido a 9 de Outubro de 2016. http://www.marksteeregames.com/Monkey\_Queen\_rules.html

## Anexos

O código fonte encontra-se num zip na mesma pasta deste relatório.