Corrida de Reis

Relatório Final



Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Corrida\_de\_Reis\_2:

Beatriz de Henriques Martins – up201502858

Beatriz Ferreira Velho – up201700491

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

1. novembro 2017

Resumo

Este trabalho consiste na construção de um jogo, “Corrida de Reis”, utilizando uma linguagem de programação em lógica, denominada como PROLOG.

“Corrida de Reis”é uma variante mais avançada do xadrez, apenas aconselhada para jogadores experientes. É um jogo para dois jogadores e com as mesmas regras do xadrez, à exceção de não ser permitido fazer a jogada de *Xeque-Mate* e o segundo jogador necessitar de jogar mais uma vez antes do final da partida. Os jogadores começam do mesmo lado tabuleiro cada um com as seguintes peças: um Rei, uma Rainha, 2 Torres, 2 Bispos e 2 Cavalos. O objetivo do jogo é levar o Rei para a última linha do tabuleiro. Se o segundo jogador conseguir colocar o seu Reina última linha do tabuleiro quando o primeiro já o tinha feito, então é considerado empate.

Através da manipulação de predicados disponibilizados pelo *SICStus Prolog*, foi possível a total implementação do jogo de modo eficiente e prático. Também implementamos todos os objetivos pedidos na descrição do guião, tal como os modos de jogo ( Humano vs. Humano, Humano vs. Computador e Computador vs. Computador).

Podemos concluir que foi através deste projeto que consolidamos o nosso conhecimento sobre a cadeira.

Índice

[Resumo 2](#_Toc498266619)

[1 Introdução 5](#_Toc498266620)

[2 O jogo “Corrida de Reis” 5](#_Toc498266621)

[2.1 Imagens do Jogo Corrida do Reis 6](#_Toc498266622)

[3 Lógica de jogo 8](#_Toc498266623)

[3.1 Representação do estado do jogo 8](#_Toc498266624)

[3.2 Visualização do tabuleiro 9](#_Toc498266625)

[3.2.1 Lista representativa do estado inicial do jogo 9](#_Toc498266626)

[3.2.2 Lista representativa de um possível estado intermédio do jogo 9](#_Toc498266627)

[3.2.3 Lista representativa de um possível estado final do jogo 10](#_Toc498266628)

[3.3 Visualização do tabuleiro em modo de texto 10](#_Toc498266629)

[3.4 Cabeçalhos dos predicados a serem implementados 11](#_Toc498266630)

[4 Interface com o utilizador 12](#_Toc498266631)

[5 Conclusão 13](#_Toc498266632)

[6 Bibliografia 13](#_Toc498266633)

Índice de Figuras

[Figura 1: Estado inicial do tabuleiro 6](#_Toc498271407)

[Figura 2: Possíveis jogadas do cavalo 6](#_Toc498271408)

[Figura 3: Possíveis jogadas do bispo 7](#_Toc498271409)

[Figura 4: Possíveis jogadas da torre 7](#_Toc498271410)

[Figura 5: Possíveis jogadas da rainha 7](#_Toc498271411)

[Figura 6: Possíveis jogadas do rei 7](#_Toc498271412)

[Figura 7: Jogo ganho pelas peças pretas 7](#_Toc498271413)

[Figura 8: Jogo ganho pelas peças brancas 7](#_Toc498271414)

[Figura 9: Jogo que irá acabar empatado 8](#_Toc498271415)

[Figura 10: Estado inicial do tabuleiro apresentado na consola 9](#_Toc498271416)

[Figura 11: Possível estado intermédio do tabuleiro apresentado na consola 9](#_Toc498271417)

[Figura 12: Possível estado final do tabuleiro apresentado na consola 10](#_Toc498271418)

[Figura 13 Menu de início 13](#_Toc498271419)

[Figura 14 Modo de jogo Humano vs Humano (jogador branco) 13](#_Toc498271420)

[Figura 15 Modo de jogo Humano vs Humano e uma jogada (jogador preto) 14](#_Toc498271421)

[Figura 16 Modo de jogo Computador vs Humano (Humano a jogar) 14](#_Toc498271422)

[Figura 17 Modo de jogo Computador vs Humano (Computador a jogar) 14](#_Toc498271423)

[Figura 18 Modo de jogo Computador vs Computador, 1ªjogada 14](#_Toc498271424)

[Figura 19 Modo de jogo Computador vs Computador, 2ªjogada 14](#_Toc498271425)

[Figura 20 Menu de escolha de níveis para o modo de jogo de 2ª e 3ª opção 14](#_Toc498271426)

[Figura 21 Menu de apoio ao jogo 15](#_Toc498271427)

[Figura 22 Menu de fim de aplicação 15](#_Toc498271428)

# Introdução

Este projeto foi proposto no âmbito da unidade curricular de Programação em Lógica do Mestrado Integrado em Engenharia Informática e Computação. Consiste na implementação de um jogo de tabuleiro para 2 jogadores, usando uma linguagem de programação em lógica, denominada como *Prolog*. O jogo de tabuleiro selecionado foi “Corrida de Reis”, um jogo variado do Xadrez.

O nosso relatório tem a seguinte estrutura:

* Introdução
* O jogo “Corrida de Reis”
* Lógica do jogo
* Interface com o utilizador
* Conclusão
* Bibliografia

# O jogo “Corrida de Reis”

Em 1961, Vernon Rylands Parton criou o jogo Corrida de Reis, uma variante do Xadrez. Sendo uma variante do Xadrez, a Corrida de Reis tem regras diferentes, bem como um objetivo diferente. É um jogo de tabuleiro 8x8 com um total de 64 casas, realizado por dois jogadores com peças de cor diferentes: preto e branco (Rachunek, 2017). Cada jogador possui 8 peças:

* Um rei;
* Uma rainha;
* Duas torres;
* Dois bispos;
* Dois cavalos.

No início do jogo, as peças são dispostas nas duas primeiras linhas do tabuleiro como apresentado na Figura 1, pelo que os dois jogadores têm a mesma vista do jogo (Rachunek, 2017).

O objetivo da Corrida de Reis é ser o primeiro a levar o próprio rei até à última linha do tabuleiro, isto é, antes do adversário. Para mover e capturar as peças são utilizadas as regras tradicionais do Xadrez, no entanto com a seguinte alteração:

* Não é permitido atacar o rei do jogador adversário, ou seja, não é possível fazer nenhuma jogada que coloque o rei adversário em xeque – posição onde pode ser capturado;

É importante relembrar que um rei não se pode pôr a si mesmo em xeque, como não pode mover-se para uma casa em que esteja uma peça adversária. É também importante relembrar que o cavalo é a única peça que pode saltar sobre outras peças, adversárias ou não.

O jogo termina quando um jogador move o seu rei para a última linha do tabuleiro. Caso seja o rei branco a chegar primeiro à última linha e o rei preto consiga, na próxima jogada, mover-se também para a última linha, o jogo termina com um empate, isto porque o jogador com as peças brancas tem a vantagem de iniciar o jogo (Rachunek, 2017).

## Imagens do Jogo Corrida do Reis

É apresentado nas figuras abaixo o estado do tabuleiro quando se inicia o jogo Corrida de Reis, seguido das figuras que representam os possíveis movimentos de cada peça, tendo em conta que não se encontram outras peças no tabuleiro, bem como figuras que representam possíveis fins de um jogo sem desistência.

|  |  |
| --- | --- |
| Figura : Estado inicial do tabuleiro  Fonte: (Lichess, 2017) | Figura : Possíveis jogadas do cavalo  Fonte: (Lichess, 2017) |
| Figura : Possíveis jogadas do bispo  Fonte: (Lichess, 2017) | Figura : Possíveis jogadas da torre  Fonte: (Lichess, 2017) |
| Figura : Possíveis jogadas da rainha  Fonte: (Lichess, 2017) | Figura : Possíveis jogadas do rei  Fonte: (Lichess, 2017) |
| Figura : Jogo ganho pelas peças pretas  Fonte: (Lichess, 2017) | Figura : Jogo ganho pelas peças brancas  Fonte: (Lichess, 2017) |
| Figura : Jogo que irá acabar empatado  Fonte: (Lichess, 2017) |  |

# Lógica de jogo

VERIFICAR!!!!

É descrito neste capítulo o método escolhido para a representação do estado do jogo, incluindo legenda do tabuleiro e exemplos de representações durante um jogo com imagens do tabuleiro impresso na consola. É também descrito o código utilizado para a impressão do tabuleiro na consola e por fim é apresentado os cabeçalhos dos predicados que serão implementados com a responsabilidade de validar e efetuar jogadas possíveis.

## Representação do estado do jogo

Para representar o estado do jogo foi escolhida uma lista de listas. Cada lista representa uma linha do tabuleiro com oitos elementos que, por sua vez, representam as casas/colunas do tabuleiro. Uma casa pode ter um dos seguintes elementos:

* Casa vazia - (espaço vazio)
* k1 – Rei Preto
* q1 – Rainha Preta
* t1 – Torre Preta
* b1 – Bispo Preto
* c1 – Cavalo Preto
* k2 – Rei Branco
* q2 – Rainha Branca
* t2 – Torre Branca
* b2 – Bispo Branco
* c2 - Cavalo Branco

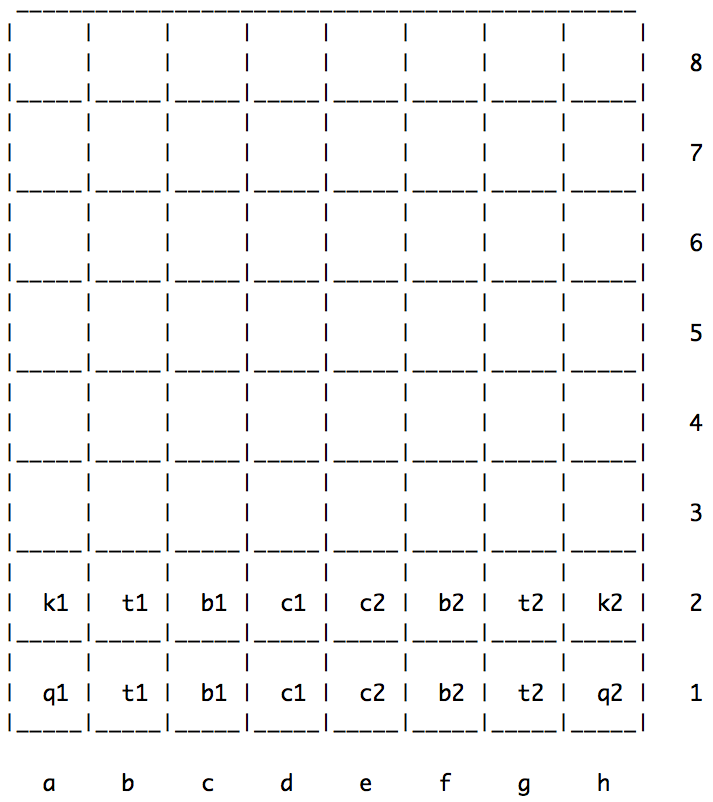
O número presente na nomenclatura atribuída define o jogador. Neste caso, o jogador com as peças pretas é identificado pelo número 1 e o jogador com as peças brancas é identificado pelo número 2.

No tabuleiro só pode haver uma única peça das referidas acima, exceto a casa vazia.

## Visualização do tabuleiro

É descrito neste subcapítulo o código utilizado para a impressão do tabuleiro na consola e por fim é apresentado os cabeçalhos dos predicados que serão implementados com a responsabilidade de validar e efetuar jogadas possíveis.

### Lista representativa do estado inicial do jogo



[ [ vazio , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],

[ vazio , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],

[ vazio , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],

[ vazio , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],

[ vazio , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],

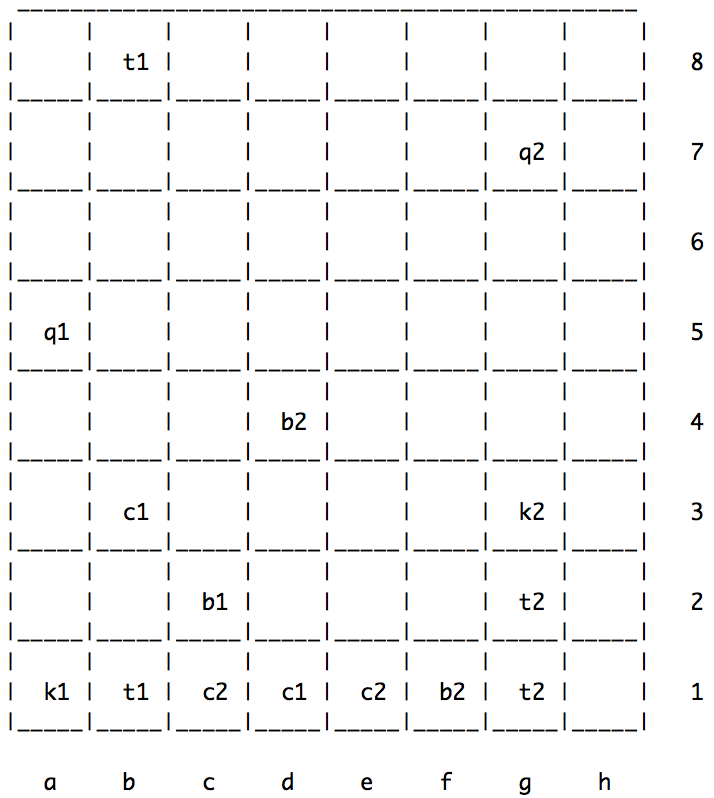
[ vazio , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],

[ k1, t1 , b1 , c1 , c2 , b2 , t2 , k2 ],

[ q1, t1 , b1 , c1 , c2 , b2 , t2 , q2 ] ]

Figura : Estado inicial do tabuleiro apresentado na consola

### Lista representativa de um possível estado intermédio do jogo



[ [ vazio , t1 , vazio , vazio , vazio , vazio , vazio , vazio ],

[ vazio , vazio , vazio , vazio , vazio , vazio , q2 , vazio ],

[ vazio , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],

[ q1 , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],

[ vazio , vazio , vazio , b2 , vazio , vazio , vazio , vazio ],

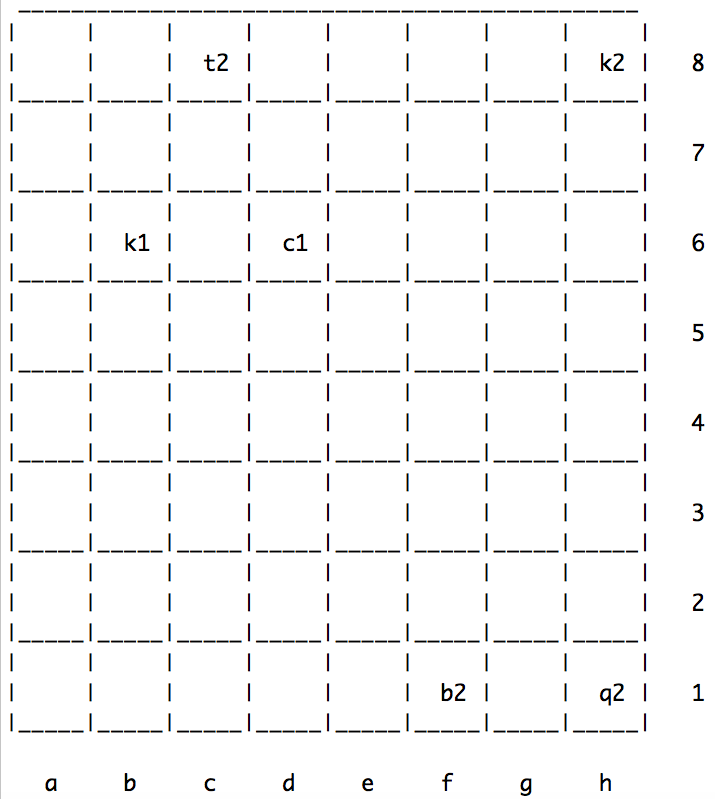
[ vazio , c1 , vazio , vazio , vazio , vazio , k2 , vazio ],

[vazio , vazio , b1 , vazio , vazio , vazio , t2 , vazio ],

[ k1 , t1 , c2 , c1 , c2 , b2 , t2 , vazio ] ]

Figura : Possível estado intermédio do tabuleiro apresentado na consola

### Lista representativa de um possível estado final do jogo



[ [ vazio , vazio , t2 , vazio , vazio , vazio , vazio , k2 ],

[ vazio , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],

[ vazio , k1 , vazio , c1 , vazio , vazio , vazio , vazio ],

[ vazio , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],

[ vazio , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],

[ vazio , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],

[ vazio , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],

[ vazio , vazio , vazio , vazio , vazio , b2, vazio , q2 ] ]

Figura : Possível estado final do tabuleiro apresentado na consola

## Visualização do tabuleiro em modo de texto

É apresentado no Excerto de Código 1 como se procede à inicialização do tabuleiro do jogo Corrida de Reis e no Excerto de Código 2 é apresentada a versão inicial do código utilizado na impressão do tabuleiro com a representação do estado do jogo.

inicializarTabuleiro(

[[vazio,vazio,vazio,vazio,vazio,vazio,vazio,vazio],

[vazio,vazio,vazio,vazio,vazio,vazio,vazio,vazio],

[vazio,vazio,vazio,vazio,vazio,vazio,vazio,vazio],

[vazio,vazio,vazio,vazio,vazio,vazio,vazio,vazio],

[vazio,vazio,vazio,vazio,vazio,vazio,vazio,vazio],

[vazio,vazio,vazio,vazio,vazio,vazio,vazio,vazio],

[k1 , t1 , b1 , c1 , c2, b2 , t2 , k2],

[q1 , t1 , b1 , c1 , c2, b2 , t2 , q2]]).

Excerto de Código : Código para inicializar o tabuleiro do jogo Corrida de Reis

% Imprime as letras que permitem identificar uma coluna do tabuleiro

impimirIdentificadoresColunas:-

        write('   a     b     c     d     e     f     g     h').

% Lista com os números que permitem identificar uma linha do tabuleiro

numeroLinhas(['8','7','6','5','4','3','2','1']).

% Imprime o limite superior do tabuleiro

imprimirSeparadorInicial:-

        write(' \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_').

% Imprime o separador de linhas do tabuleiro

imprimirSeparadorLinhas:-

        write('|\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_|  ').

% Imprime o separador de colunas do tabuleiro

imprimirSeparadorColunas:-

        write('|     |     |     |     |     |     |     |     |').

% Imprime uma casa do tabuleiro com a peca "Peca".

imprimirCasa(Peca, 2):-

        write('  '), write(Peca), write(' ').

imprimirCasa(\_, \_):-

        write('     ').

% Imprime as pecas que estão numa determinada linha do tabuleiro

imprimirPecasLinha([]).

imprimirPecasLinha([H | T]):-

        atom\_length(H, L), write('|'),

        imprimirCasa(H, L),

        imprimirPecasLinha(T).

% Imprime a linha numero "NLinha" do tabuleiro

imprimirLinha([] , []).

imprimirLinha(Linha , NLinha):-

        imprimirSeparadorColunas, nl,

        imprimirPecasLinha(Linha), write('|   '), write(NLinha), nl,

        imprimirSeparadorLinhas, nl.

% Imprime todas as linhas do tabuleiro

imprimirLinhas([],[]).

imprimirLinhas([Linha | T], [NLinha | ListaLinhas]):-

        imprimirLinha(Linha, NLinha),

        imprimirLinhas(T, ListaLinhas).

% Imprime o tabuleiro com o estado atual do jogo

imprimirTabuleiro([H | T]):-

        imprimirSeparadorInicial, nl,

        numeroLinhas(ListaLinhas),

        imprimirLinhas([H | T], ListaLinhas), nl,

        impimirIdentificadoresColunas, nl.

Excerto de Código : Código para a impressão do tabuleiro com o estado do jogo

A Figura 10, Figura 11 e Figura 12 são três resultados diferentes do output do predicado imprimirTabuleiro representado no Excerto de Código 2, tendo como argumento a lista junto a cada figura.

## Cabeçalhos dos predicados a serem implementados

Numa fase posterior irão ser implementados os seguintes predicados:

* moverPeca(Tabuleiro, Jogador, Linha, Coluna, LinhaDest, ColunaDest, TabuleiroF)

Predicado que irá mover uma peça de um jogador da posição atual para uma posição destino. Este predicado é executado depois de o movimento ser validado.

* validarJogada(Tabuleiro, Jogador, Linha, Coluna, LinhaDest, ColunaDest, TabuleiroF)

Predicado que irá validar o movimento da peça desde a casa onde se encontra até à casa destino.

Depois de ser validado, o movimento pode ser efetuado ou pode ser negado, sendo neste caso pedido um novo movimento ao jogador.

* validarCasaComPeca(Tabuleiro, Jogador, LinhaI, ColunaI, Peca)

Predicado que irá validar se a posição/casa origem inserida pelo jogador contém uma peça. Caso a casa tenho uma peça, ela é devolvida em “Peca”, caso não tenha é pedida uma nova posição ao jogador.

* validarPecaJogador(JogadorEmJogo, JogadorPeca)

Predicado que irá validar se a peça na posição/casa origem inserida pelo jogador lhe pertence. Caso não pertença ao jogador em jogo é apresentada uma mensagem com essa informação e é pedida uma nova posição. Este predicado irá ser chamado depois do predicado anterior: validarCasaComPeca.

# Interface com o utilizador

A interface foi desenhada de modo proporcionar ao utilizador uma experiência agradável e bastante intuitiva.

O menu inicial do jogo permite ao utilizador escolher o modo de jogo que deseja (*1.* ou *2.* ou *3.* seguido de *Enter*), consultar as instruções de jogo (*4.* seguido de *Enter*) ou sair do jogo (*5.* seguido de *Enter*).

O jogador pode escolher um dos 3 modos de jogo:

* Modo Jogador *vs* Jogador;
* Modo Jogador *vs* Computador;
* Modo Computador *vs* Computador.

Quando o jogador escolhe a primeira opção (*1. Start Game Player vs Player*) é desenhado de imediato o tabuleiro. Debaixo do tabuleiro existe uma mensagem onde está especificado o jogador e como tem de executar a jogada, ou seja, introduzir as coordenadas para mover a peça, linha/coluna.

Quando o utilizador escolhe a segunda opção (*2. Start Game PC vs Player*) está a jogar contra o próprio computador. A interação neste modo é bastante semelhante à opção anterior (*1. Start Game Player vs Player*)*,* à exceção de que é necessário escolher um nível de jogo. Os níveis disponíveis são:

1. Uma partida “normal”, semelhante à primeira opção, sem qualquer tipo de ataque ou captura;
2. Uma partida mais emocionante, em que é permitido atacar e capturar as peças dos adversários.

Na terceira opção (*2. Start Game PC vs PC*), a última opção de jogo, é o modo automático, ou seja, o computador joga sozinho contra si próprio. Em semelhança à opção anterior (*2. Start Game PC vs Player*), o utilizador também tem de escolher o nível de jogo que deseja assistir.

A quarta opção (*4. How to play*) é um menu onde está definido o objetivo de jogo, as regras mais importantes e as peças de cada jogador.

A última opção (*5. Exit*) é apenas para terminar a aplicação em segurança.

|  |  |
| --- | --- |
| /Users/beatrizdehenriquesmartins/git/FEUP---PLOG/Entrega Final/Imagens/menu_Incio_Jogo.png  Figura Menu de início | /Users/beatrizdehenriquesmartins/git/FEUP---PLOG/Entrega Final/Imagens/Tabuleiro-inicial_JJ.png  Figura Modo de jogo Humano vs Humano (jogador branco) |
| /Users/beatrizdehenriquesmartins/git/FEUP---PLOG/Entrega Final/Imagens/Tabuleiro_JJ_preto.png  Figura Modo de jogo Humano vs Humano e uma jogada (jogador preto) | Figura Modo de jogo Computador vs Humano (Humano a jogar) |
| Figura 17 Modo de jogo Computador vs Computador, 1ªjogada |  |
| /Users/beatrizdehenriquesmartins/git/FEUP---PLOG/Entrega Final/Imagens/nivel_JC.png  Figura 18 Menu de escolha de níveis para o modo de jogo de 2ª e 3ª opção | /Users/beatrizdehenriquesmartins/git/FEUP---PLOG/Entrega Final/Imagens/menu_ajuda.png  Figura 19 Menu de apoio ao jogo |
| /Users/beatrizdehenriquesmartins/git/FEUP---PLOG/Entrega Final/Imagens/menu_exit.png  Figura Menu de fim de aplicação |

# Conclusão

# Bibliografia

Lichess. (2017). Racing Kings. Retrieved from <https://lichess.org/study/7qOrZwG6>

Rachunek, F. (2017). BrainKing - Regras do jogo (Corrida de Reis). Retrieved from <https://brainking.com/pt/GameRules?tp=125>

**Resumo** - Resumo sucinto do trabalho com 150 a 250 palavras (problema abordado, objetivo, como foi o problema resolvido/abordado, principais resultados e conclusões).

1. **Introdução** - Descrever os objetivos e motivação do trabalho. Descrever num parágrafo breve a estrutura do relatório.
2. **O jogo XXX** - Descrever sucintamente o jogo, a sua história e, principalmente, as suas regras. Devem ser incluídas imagens apropriadas para explicar o funcionamento do jogo. (Pode ser idêntico ao texto do relatório intercalar.)
3. **Logica do Jogo** - Descrever o projeto e implementação da lógica do jogo em Prolog, incluindo a forma de representação do estado do tabuleiro e sua visualização, execução de movimentos, verificação do cumprimento das regras do jogo, determinação do final do jogo e cálculo das jogadas a realizar pelo computador utilizando diversos níveis de jogo. Sugere-se a estruturação desta secção da seguinte forma:
   1. **Representação do estado do jogo –** pode ser idêntico descrito no relatório intercalar
   2. **Visualização do tabuleiro –** pode ser idêntico ao descrito no relatório intercalar
   3. **Lista de jogadas validas -** Obtenção de uma lista de jogadas possíveis. Exemplo: valid moves(+Board, -ListOfMoves).
   4. **Execução de jogadas -** Validação e execução de uma jogada num tabuleiro, obtendo o novo estado do jogo. Exemplo: move(+Move, +Board, -NewBoard).
   5. **Avaliação do tabuleiro** - Avaliação do estado do jogo, que permitirá comparar a aplicação das diversas jogadas disponíveis. Exemplo: value(+Board, +Player, -Value).
   6. **Final do jogo** - Verificação do fim do jogo, com identificação do vencedor. Exemplo: game over(+Board, -Winner).
   7. **Jogada do Computador** - Escolha da jogada a efetuar pelo computador, dependendo do n ́ıvel de difi- culdade. Por exemplo: choose move(+Level, +Board, -Move).
4. **Interface com o utilizador** – Descrever o módulo de interface com o utilizador em modo de texto.
5. **Conclusões** - Que conclui deste projecto? Como poderia melhorar o trabalho desenvolvido?