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

12 novembro 2017

Resumo

O presente trabalho prático consiste na construção de um jogo, "Corrida de Reis", utilizando uma linguagem de programação em lógica, denominada como *Prolog*.

O jogo "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*. Os jogadores começam do mesmo lado do tabuleiro e cada jogador tem as seguintes peças: um Rei, uma Rainha, duas Torres, dois Bispos e dois Cavalos. O objetivo do jogo é levar o Rei para a última linha do tabuleiro.

Através da manipulação de predicados implementados e também alguns nativos do *SICStus Prolog*, foi possível a implementação do jogo de modo eficiente e prático. Foram 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 matéria lecionada na unidade curricular.

Índice

Índice de Figuras

1 Introdução

Este projeto foi proposto no âmbito da unidade curricular de Programação em Lógica do Mestrado Integrado em Engenharia Informática e Computação. O trabalho 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", uma variante do Xadrez. O relatório tem a seguinte estrutura:

- O jogo "Corrida de Reis"
- Lógica do jogo
- Interface com o utilizador
- Conclusão
- Bibliografia

2 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, todas as peças estão dispostas nas duas 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).

2.1 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.

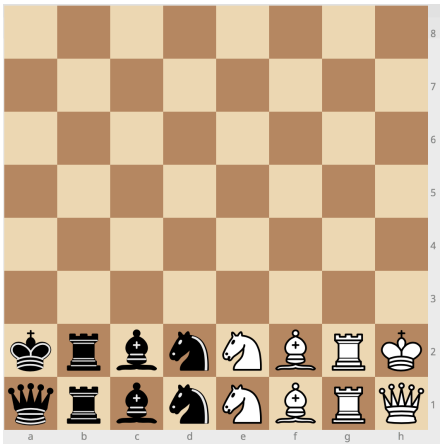


Figura 1: Estado inicial do tabuleiro

Fonte: (Lichess, 2017)

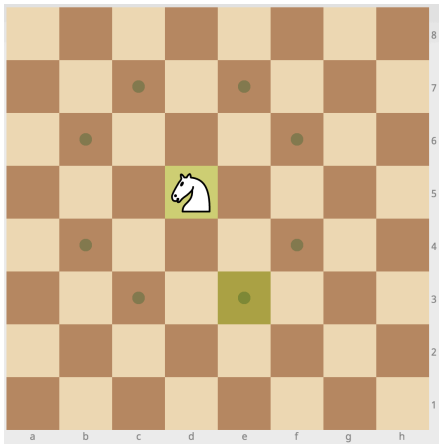


Figura 2: Possíveis jogadas do cavalo

Fonte: (Lichess, 2017)

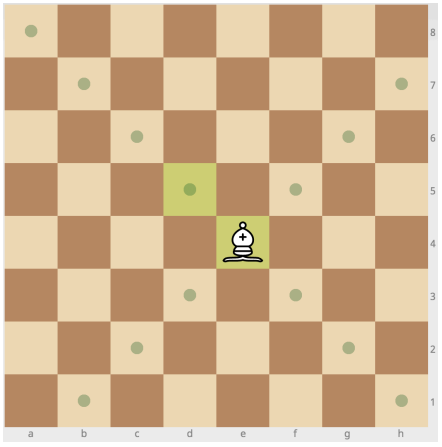


Figura 3: Possíveis jogadas do bispo

Fonte: (Lichess, 2017)

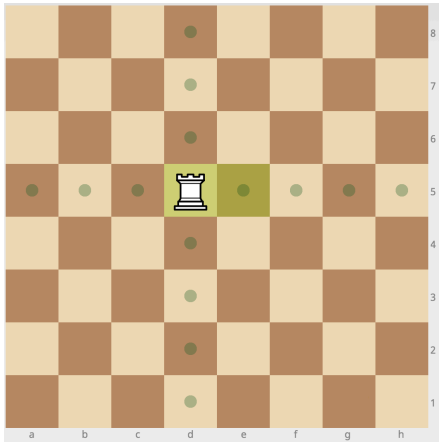


Figura 4: Possíveis jogadas da torre

Fonte: (Lichess, 2017)

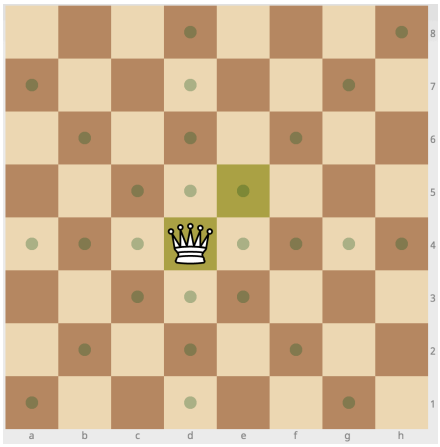


Figura 5: Possíveis jogadas da rainha

Fonte: (Lichess, 2017)

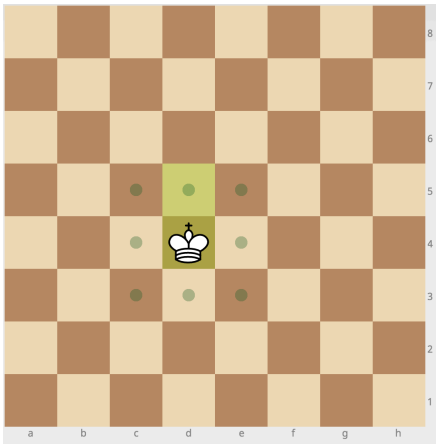


Figura 6: Possíveis jogadas do rei

Fonte: (Lichess, 2017)

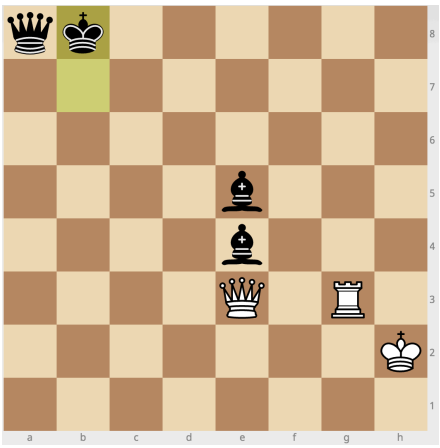


Figura 7: Jogo ganho pelas peças pretas

Fonte: (Lichess, 2017)

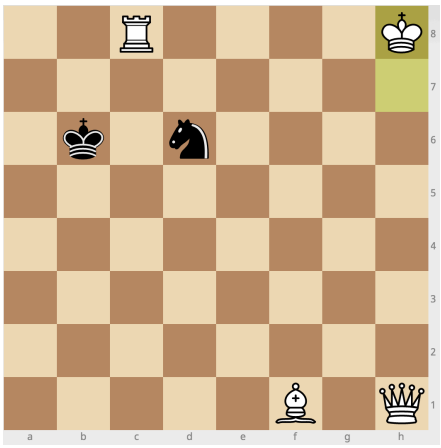


Figura 8: Jogo ganho pelas peças brancas

Fonte: (Lichess, 2017)

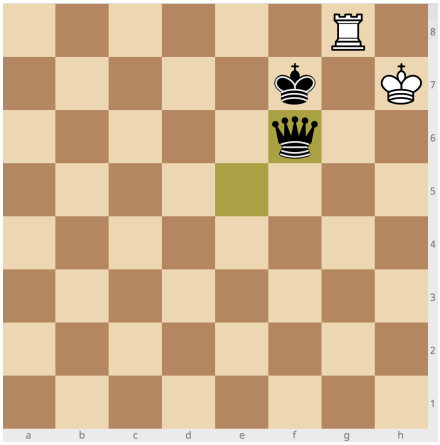


Figura 9: Jogo que irá acabar empatado

Fonte: (Lichess, 2017)

3 Lógica de jogo

Neste capítulo é descrita a implementação da lógica do jogo em *Prolog*. Inclui a representação do estado do tabuleiro e a sua visualização, a execução dos movimentos das peças, tendo em atenção todas as regras do jogo, a validação do fim do jogo e a execução das jogadas pelo computador utilizando dois níveis.

3.1 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)
- K – Rei Preto
- Q – Rainha Preta
- T – Torre Preta
- B – Bispo Preto
- C – Cavalo Preto
- k – Rei Branco
- q – Rainha Branca
- t – Torre Branca
- b – Bispo Branco
- c - Cavalo Branco

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

3.2 Visualização do tabuleiro

É descrito nesta subsecção como se procede para a impressão do estado do tabuleiro, bem como imagens que representam estados possíveis do jogo.

3.2.1 Impressão do tabuleiro

De forma a simplificar a implementação da lógica do jogo o tabuleiro utilizado na implementação do jogo é diferente do tabuleiro utilizado na impressão. O tabuleiro apresentado de seguida é o tabuleiro utilizado na implementação da lógica de jogo:

```
[ [ {} , {} , {} , {} , {} , {} , {} , {} ] ,
[ {} , {} , {} , {} , {} , {} , {} , {} ] ,
[ {} , {} , {} , {} , {} , {} , {} , {} ] ,
[ {} , {} , {} , {} , {} , {} , {} , {} ] ,
[ {} , {} , {} , {} , {} , {} , {} , {} ] ,
[ {} , {} , {} , {} , {} , {} , {} , {} ] ,
[{7,1,k,preto},{7,2,t,preto},{7,3,b,preto},{7,4,c,preto},{7,5,c,branco},{7,6,b,branco},{7,7,t,branco},{7,8,k,branco}],
[{8,1,q,preto},{8,2,t,preto},{8,3,b,preto},{8,4,c,preto},{8,5,c,branco},{8,6,b,branco},{8,7,t,branco},{8,8,q,branco}]]
```

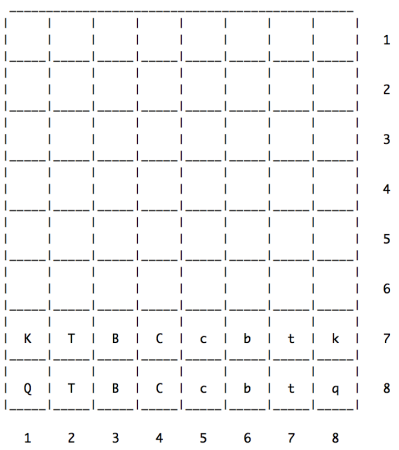
Quando é necessário imprimir o estado do tabuleiro é utilizado um predicado que converte o tabuleiro acima apresentado para o seguinte formato:

[[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],
[K, T , B , C , c , b , t , k],
[Q, T , B , C , c , b , t , q]]

As peças do jogador preto são convertidas para letras maiúsculas e as peças do jogador branco são convertidas para minúsculas. Para a conversão de caracter é utilizado o predicado nativo `char_code`. O predicado utilizado para a conversão é o seguinte:

```
converterTabuleiro([],[]).

converterTabuleiro([H|T],[TAB2|TAB1]):-
        converterTabuleiro(T,TAB1),
        converterTabuleiroRecurse(H,TAB2).

converterTabuleiroRecurse([],[]).

converterTabuleiroRecurse([H|T],[TAB2|TAB1]):-
        converterTabuleiroRecurse(T,TAB1),
        converterCasa(H,TAB2).

converterCasa({},vazio).

converterCasa({_,_,TIPO,branco},TIPO).

converterCasa({_,_,TIPO,preto},CASA):-
        char_code(TIPO,CODE),
        CODE1 is CODE-32,
        char_code(CASA,CODE1).
```

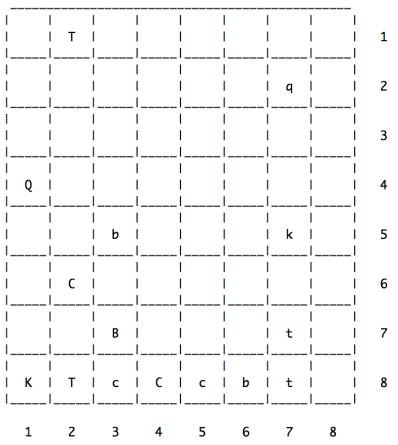
3.2.2 Lista representativa do estado inicial do jogo




```
[ [ 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 ],
                          [ K, T , B , C , c , b , t , k ],
                          [ Q, T , B , C , c , b , t , q ] ]
```

Figura 10: Estado inicial do tabuleiro apresentado na consola

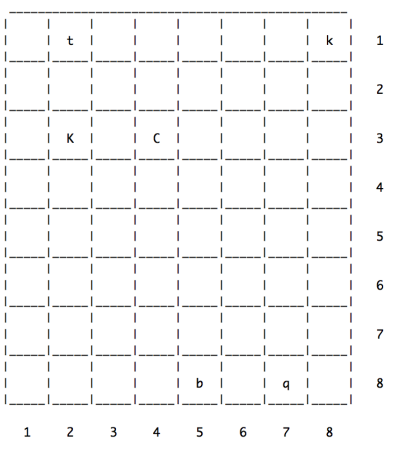
3.2.3 Lista representativa de um possível estado intermédio do jogo




```
[ [ vazio , T , vazio , vazio , vazio , vazio , vazio , vazio ],
  [ vazio , vazio , vazio , vazio , vazio , vazio , q , vazio ],
  [ vazio , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],
  [ Q , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],
  [ vazio , vazio , vazio , b , vazio , vazio , k , vazio ],
  [ vazio , C , vazio , vazio , vazio , vazio , vazio , vazio ],
  [vazio , vazio , B , vazio , vazio , vazio , t , vazio ],
  [ K , T , c , C , c , b , t , vazio ] ]
```

Figura 11: Possível estado intermédio do tabuleiro apresentado na consola

3.2.4 Lista representativa de um possível estado final do jogo




```
[ [ vazio , vazio , t , vazio , vazio , vazio , vazio , k ],
  [ vazio , vazio , vazio , vazio , vazio , vazio , vazio , vazio ],
  [ vazio , K , vazio , C , 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 , b , vazio, q , vazio ] ]
```

Figura 12: Possível estado final do tabuleiro apresentado na consola

3.3 Validação da posição inicial introduzida

Introduzida a posição da peça que o jogador pretende mover, são realizadas as duas validações. Primeiro é verificado se a posição inserida se encontra dentro do tabuleiro, e posteriormente é verificado se a posição tem uma peça e se pertence ao jogador que está no momento a jogar. Sendo a segunda validação mais relevante, é apresentado o código relativo à mesma.

```
validarPecaIntroduzidaJogador(TAB,PL,PC,JOGADOR,
                                {PL,PC,TIPO,JOGADOR}):-
        obterPecasTabuleiro(TAB,PECAS),
        member({PL,PC,TIPO,JOGADOR},PECAS).
```

O predicado `validarPecaIntroduzidaJogador` tem como argumentos o tabuleiro `TAB`, a linha `PL` e coluna `PC` introduzidas pelo utilizador, o jogador `JOGADOR` a jogar no momento e a posição escolhida na forma `{PL,PC,TIPO,JOGADOR}`, em que o `TIPO` representa o tipo de peça naquela posição. Neste predicado é obtida uma lista com as peças presentes no tabuleiro na forma `{PL,PC,TIPO,JOGADOR}`. Se a linha, a coluna e o jogador pertencerem à lista de peças a posição é aceite.

3.4 Validação da posição final introduzida

Nesta subsecção encontra-se a grande parte da lógica do jogo. Após a introdução da posição para a qual pretende mover a peça selecionada são realizadas várias validações que pretendem responder às regras do jogo. As validações realizadas são:

- Dentro do tabuleiro;
- Validar movimento da peça;
- Validar interseção entre peças;
- Mexer/consumir peça;
- Verificar xeque;
- Validar fim do jogo;

O predicado responsável pelas validações referidas anteriormente é o seguinte:

```
validarPosicaoFinalIntroduzida(TAB,FL,FC,CASA,GANHOU,NEWTAB):-
        validarDentroTabuleiro(FL,FC),
        validarMovimento(CASA,{FL,FC}),
        validarIntersecaoEntrePecasBase(CASA,{FL,FC},TAB),
        mexerPecaBase(CASA,{FL,FC},TAB,NEWTAB),
        verificarXeque(NEWTAB),
        validarFimJogo(NEWTAB,GANHOU).
```

Nas subsecções seguintes serão descritas com um nível de granularidade fina cada uma das validações referidas.

3.4.1 Validar o movimento da peça

Nesta fase do jogo é apenas verificado se a peça que o jogador selecionou pode efetuar o movimento para a posição final introduzida.

```
validarMovimento({PL,PC,TIPO,_},{FL,FC}):-
        PL\=FL,
        validarMovimentoSub({PL,PC,TIPO},{FL,FC}).

validarMovimento({PL,PC,TIPO,_},{FL,FC}):-
        PC\=FC,
        validarMovimentoSub({PL,PC,TIPO},{FL,FC}).
```

Nos predicados acima o primeiro argumento representa a posição inicial da peça, contendo a linha e a coluna, bem como o tipo de peça e o segundo argumento representa a posição final para a qual a peça se deverá mover. É retornado *true* se o movimento for válido.

São elaborados dois predicados para verificar se a posição inicial é diferente da posição final, caso seja a posição final é rejeitada. Caso a linha inicial seja diferente da linha final ou a coluna inicial diferente da coluna final é chamado um dos seguintes predicados:

```
validarMovimentoSub({PL,PC,t},{FL,FC}):-
        validarMovimentoTorre(PL,PC,FL,FC).

validarMovimentoSub({PL,PC,q},{FL,FC}):-
        validarMovimentoRainha(PL,PC,FL,FC).
```

```
validarMovimentoSub({PL,PC,k},{FL,FC}):-
        validarMovimentoRei(PL,PC,FL,FC).

validarMovimentoSub({PL,PC,b},{FL,FC}):-
        validarMovimentoBispo(PL,PC,FL,FC).

validarMovimentoSub({PL,PC,c},{FL,FC}):-
        validarMovimentoCavalo(PL,PC,FL,FC).
```

Tendo em conta o tipo de peça, é chamado o predicado correspondente à validação do movimento. Sabendo que os possíveis movimentos das peças apenas podem ser do tipo verticais, horizontais ou diagonais, foram elaborados predicados para estes possíveis movimentos, os quais são depois chamados nos movimentos das peças adequadas.

É apresentado a seguir os predicados responsáveis por verificar se, consoante uma posição inicial e uma posição final, o movimento é vertical, horizontal ou diagonal:

```
validarMovimentoVerticalHorizontal(PL,PC,FL,FC):-
        X is FC-PC,
        Y is FL-PL,
        validarMovimentoVH(X,Y).

validarMovimentoVH(X,Y):-
        X=0,Y\=0.
validarMovimentoVH(X,Y):-
        X\=0,Y=0.

validarMovimentoDiagonal(PL,PC,FL,FC):-
        X is FC-PC,
        Y is FL-PL,
        AX is abs(X),
        AY is abs(Y),
        AX=AY.
```

3.4.1.1 Movimento do Rei

Visto que o rei pode mover-se apenas uma casa em cada direção foram elaborados dois predicados, um que valida se o movimento é vertical ou horizontal e o outro que valida se é diagonal. Os últimos três predicados verificam se o rei está a mover-se apenas uma casa.

```
validarMovimentoRei(PL,PC,FL,FC):-
        validarMovimentoVerticalHorizontal(PL,PC,FL,FC),
        AX is abs(FC-PC),
        AY is abs(FL-PL),
        validarVHRei(AX,AY).

validarMovimentoRei(PL,PC,FL,FC):-
        validarMovimentoDiagonal(PL,PC,FL,FC),
        AX is abs(FC-PC),
        AY is abs(FL-PL),
        validarDGRei(AX,AY).

validarDGRei(AX,AX):-AX=1.

validarVHRei(AX,AY):-AX=0,AY=1.

validarVHRei(AX,AY):-AX=1,AY=0.
```

3.4.1.2 Movimento da Rainha

Os predicados abaixo apresentados verificam se o movimento da rainha é vertical ou horizontal, caso não seja, verifica se é diagonal.

```
validarMovimentoRainha(PL,PC,FL,FC):-
        validarMovimentoVerticalHorizontal(PL,PC,FL,FC).

validarMovimentoRainha(PL,PC,FL,FC):-
        validarMovimentoDiagonal(PL,PC,FL,FC).
```

3.4.1.3 Movimento da Torre

Visto que a torre pode mover-se na vertical ou horizontal, é apenas validado se o movimento corresponde a um dos possíveis.

```
validarMovimentoTorre(PL,PC,FL,FC):-
        validarMovimentoVerticalHorizontal(PL,PC,FL,FC).
```

3.4.1.4 Movimento do Bispo

O predicado abaixo valida se o movimento é diagonal visto ser o único movimento possível de um bispo.

```
validarMovimentoBispo(PL,PC,FL,FC):-
        validarMovimentoDiagonal(PL,PC,FL,FC).
```

3.4.1.5 Movimento do Cavalo

O cavalo pode mover-se na forma de L, ou seja, para o movimento do cavalo os predicados que verificam os movimentos verticais, horizontais e diagonais não são aplicados.

```
validarMovimentoCavalo(PL,PC,FL,FC):-
        AX is abs(PC-FC),
        AY is abs(PL-FL),
        validarMovimentoCavaloSub(AX,AY).

validarMovimentoCavaloSub(AX,AY):-AX=1,AY=2.

validarMovimentoCavaloSub(AX,AY):-AX=2,AY=1.
```

3.4.2 Validar a interseção entre peças

Validado o movimento da peça da posição inicial para a posição final, é necessário verificar se não existem peças ao longo do caminho que a peça percorre. Esta regra não se aplica ao cavalo pois é a única peça que pode "saltar" sobre outras peças no seu movimento.

O predicado seguinte é o predicado base para a validação em questão e devolve *true* se não existir interseção. De modo a facilitar o processo é utilizado uma lista com todas as casas do tabuleiro que têm uma peça. Os elementos da lista estão no formato `{PL,PC,TIPO,JOGADOR}`, bem como o argumento `PECA`.

```
validarIntersecaoEntrePecasBase(PECA,PF,TAB):-
        obterPecasTabuleiro(TAB,PECAS),
        validarIntersecaoEntrePecas(PECA,PF,PECAS).
```

Os predicados seguintes são responsáveis por verificar se uma peça, representada pelos argumentos `CL` e `CC`, se encontra na linha entre a posição inicial e a posição final.

```
validarPontoNaLinha(PL,PC,CL,CC,FL,FC):-
        validarLinhaDiagonal(PL,PC,CL,CC,FL,FC).

validarPontoNaLinha(PL,PC,CL,CC,FL,FC):-
        validarLinhaHorizontal(PL,PC,CL,CC,FL,FC).

validarPontoNaLinha(PL,PC,CL,CC,FL,FC):-
        validarLinhaVertical(PL,PC,CL,CC,FL,FC).
```

As validações para cada tipo de linha (diagonal, horizontal e vertical) são realizadas nos seguintes predicados:

```
validarLinha(PL,PC,CL,CC,FL,FC):-
        NUMERADOR1 is CC-PC, % Linha Horizontal
        DIVISOR1 is FC-PC,   % Linha Horizontal
        NUMERADOR is CL-PL,
        DIVISOR is FL-PL,
        number(DIVISOR),
        DIVISOR\=0,
        DIVISOR1\=0,
        ALPHA1 is NUMERADOR1/DIVISOR1,
        ALPHA is NUMERADOR/DIVISOR,
        ALPHA1=ALPHA,
        ALPHA>0,
        ALPHA<1.

validarLinhaVertical(PL,PC,CL,_,FL,FC):-
        NUMERADOR is CL-PL,
        DIVISOR is FL-PL,
        DIFFH is FC-PC,
        DIVISOR\=0,
```

```
        DIFFH=0,
        ALPHA is NUMERADOR/DIVISOR,
        ALPHA>0,
        ALPHA<1.

validarLinhaHorizontal(PL,PC,_,CC,FL,FC):-
        NUMERADOR is CC-PC,
        DIVISOR is FC-PC,
        DIFFV is FL-PL,
        DIVISOR\=0,
        DIFFV=0,
        ALPHA is NUMERADOR/DIVISOR,
        ALPHA>0,
        ALPHA<1.
```

É utilizada matemática para verificar se um ponto está na reta definida entre a posição inicial e a posição final.

3.4.3 Mexer ou consumir peça

Esta fase do jogo corresponde a efetuar o movimento para a posição final, em que é possível ser consumida uma peça adversária ou não. Para estas duas possibilidades foram definidos dois predicados:

```
mexerPecaBase({PL,PC,TIPO,JOGADOR},{FL,FC},TAB,T):-
        consumirPecaBase({PL,PC,TIPO,JOGADOR},{FL,FC},TAB,T,CONSUMIU),
        CONSUMIU>0.

mexerPecaBase({PL,PC,TIPO,JOGADOR},{FL,FC},TAB,T):-
        obterPecasTabuleiro(TAB,PECAS),
        \+member({FL,FC,_,_},PECAS),
        mexerPeca({PL,PC,TIPO,JOGADOR},{FL,FC},TAB,L2),
        apagarPeca({PL,PC,TIPO,JOGADOR},L2,T,1).
```

O primeiro predicado é responsável por mover uma peça para a posição final e consumir a peça adversária que lá se encontra, devolvendo o novo tabuleiro `T`. Se o argumento `CONSUMIU` for igual a 0, ou seja, não foi consumida nenhuma peça, é executado o segundo predicado, o qual é responsável por movimentar a peça para a posição final sem consumir nenhuma peça adversária.

3.4.4 Verificar xeque

Depois de um movimento ser executado e ser criado um novo estado do tabuleiro é preciso validar se ambos os Reis estão em xeque, pois o jogador não pode pôr o seu Rei em xeque nem deixar o Rei adversário em xeque. Se alguma das situações ocorrer a posição final introduzida pelo jogador não é válida. O predicado abaixo apresentado é responsável pela referida validação e devolve *true* se nenhum Rei estiver em xeque.

```
verificarXeque(TAB):-
        getPosicoesRei(TAB,POSICAOB,POSICAOP),
        verificarXeque(TAB,POSICAOB,preto,TAB),
        verificarXeque(TAB,POSICAOP,branco,TAB).
```

Inicialmente é executado o predicado que devolve as posições dos dois Reis, posteriormente é verificada a condição de xeque para ambos, sendo que para o Rei Branco são tidas em conta as peças pretas e para o Rei Preto as peças brancas. No predicado `verificarXeque` é utilizado o predicado `validarMovimento`, consoante o tipo de peça que se está a analisar, bem como o predicado `validarIntersecaoEntrePecasBase`. É apresentado o exemplo da Torre:

```
verificarXequeTorre({PL,PC,t,JOGADOR},{PLR,PCR},JOGADOR,_):-
        \+validarMovimentoTorre(PL,PC,PLR,PCR).

verificarXequeTorre({PL,PC,t,JOGADOR},{PLR,PCR},JOGADOR,TAB):-
        \+validarIntersecaoEntrePecasBase({PL,PC,t,JOGADOR},
                                          {PLR,PCR},TAB).
```

Para uma peça não pôr o Rei adversário em xeque é necessário que o movimento da peça para a posição do Rei não seja válido, ou, caso o movimento seja válido, é necessário que haja interseção de outras peças no movimento.

O exemplo corresponde ao caso da Torre, mas é aplicado de igual forma à Rainha e ao Bispo. Quanto ao Cavalo é apenas verificado se o movimento não é válido.

3.4.5 Verificar fim do jogo

No que refere ao fim do jogo é verificado se algum Rei já chegou à linha 1 do tabuleiro ou se ocorre empate. O predicado seguinte é o predicado base:

```
validarFimJogo(TAB,JOGADOR):-
        validarFim(TAB,1, JOGADOR).
```

No que refere ao empate, inicialmente é verificado se o Rei Preto se encontra na linha 2, representada pelo argumento H1, e se o Rei Branco se encontra na linha 1, argumento H. Caso esta situação se verifique é validado se o Rei Preto consegue efetuar uma jogada para a última linha, e se conseguir ocorre empate.

```
validarEmpate(H,[H1|_]):-
        !,validarReiLinha(H1, preto),
        validarReiLinha(H, branco),
        validarReiPodeMexer(H1,H).
```

Caso o empate não se verifique, é testado se o jogador que está a jogar tem o seu Rei na última linha, sendo assim considerado o vencedor. Para esta validação são executados os seguintes predicados:

```
validarReiLinha([], 0).

validarReiLinha([{_,_,k,JOGADOR}|_], JOGADOR):-!.

validarReiLinha([_|T],JOGADOR):-
        validarReiLinha(T,JOGADOR).
```

3.5 Jogadas do computador

No que refere às jogadas do computador foram definidos dois níveis:

- Nível 1: não pode atacar o adversário;
- Nível 2: pode atacar o adversário.

Para a elaboração dos níveis foi utilizado o predicado nativo *random*. O predicado `getPosicaoInicialRandom` gera uma posição que esteja contida no argumento `PECAS`, e que a peça que está nessa posição pertença ao jogador `JOGADOR`, e é utilizado para ambos os níveis.

```
getPosicaoInicialRandom(JOGADOR,PECAS,{LINHA,COLUNA,TIPO,JOGADOR}):-
```

```
        random(0,9,LINHA),
        random(0,9,COLUNA),
        member({LINHA,COLUNA,TIPO,JOGADOR},PECAS),!.

getPosicaoInicialRandom(JOGADOR,PECAS,{LINHA,COLUNA,TIPO,JOGADOR}):-
         getPosicaoInicialRandom(JOGADOR,PECAS,
         {LINHA,COLUNA,TIPO,JOGADOR}).
```

No que refere à escolha da posição final é executado o predicado getPosicaoFinalRandom1, caso o nível seja o 1, ou o predicado getPosicaoFinalRandom2, caso o nível seja o 2. No nível 1 a posição gerada não pode pertencer à lista de peças `PECAS`, de modo a não puder atacar o adversário, enquanto que no nível 2 é aceite qualquer posição gerada desde que seja válida.

```
getPosicaoFinalRandom1(PECAS,{LINHA,COLUNA}):-
        random(0,9,LINHA),
        random(0,9,COLUNA),
        \+member({LINHA,COLUNA,_,_},PECAS),!.

getPosicaoFinalRandom1(PECAS,{LINHA,COLUNA}):-
        getPosicaoFinalRandom(PECAS,{LINHA,COLUNA}).

getPosicaoFinalRandom2({LINHA,COLUNA}):-
        random(0,9,LINHA),
        random(0,9,COLUNA).
```

4 Interface com o utilizador

A interface foi desenhada de modo a 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" 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 (*3. 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.

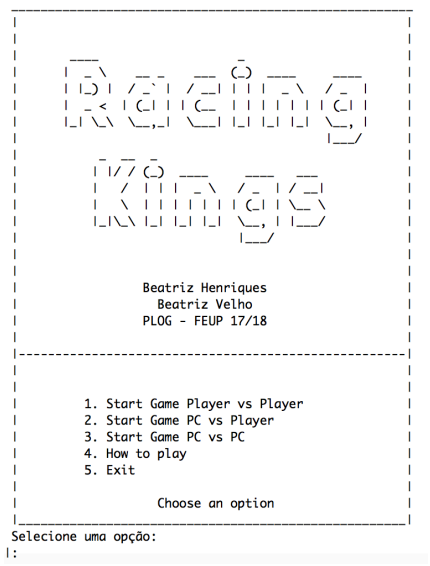


Figura 13: Menu de início

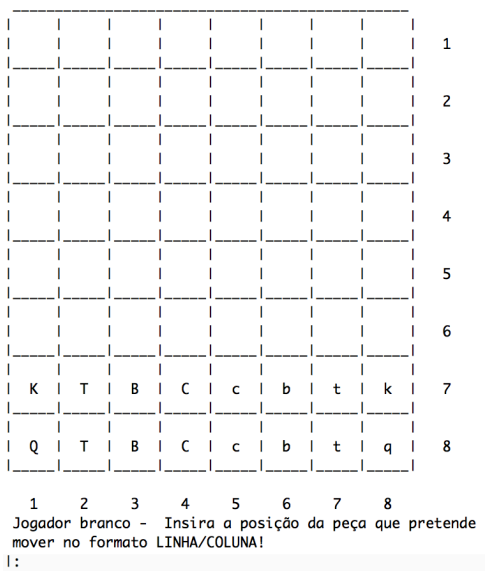


Figura 14: Tabuleiro inicial de modo de jogo Humano vs Humano

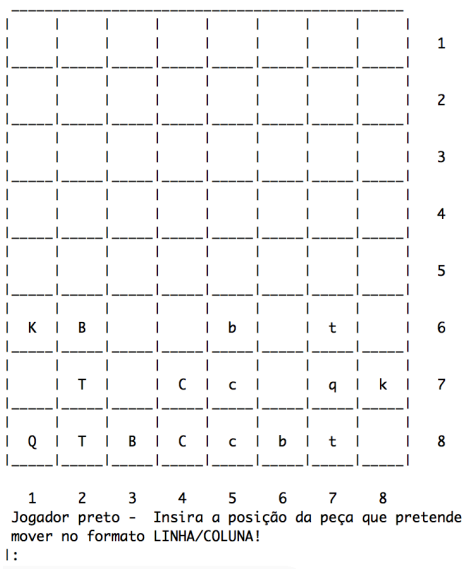


Figura 15: Exemplo de modo de jogo Humano vs Humano

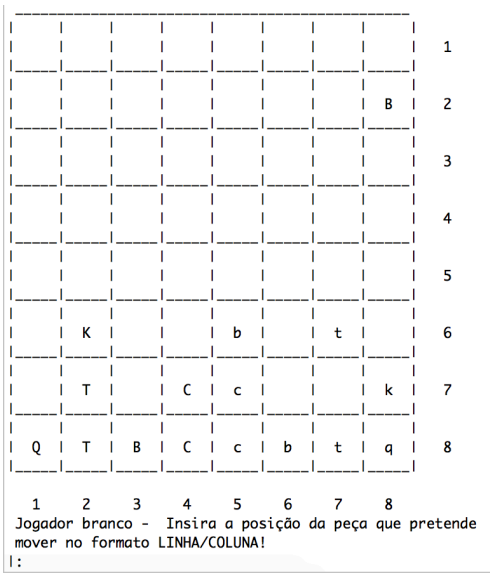


Figura 16: Exemplo de modo de jogo Computador vs Humano

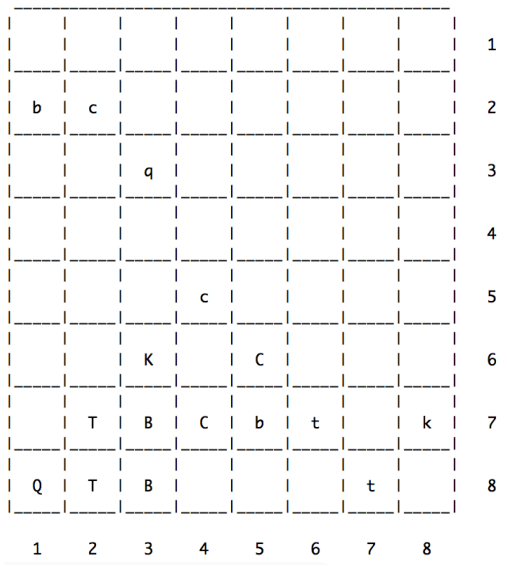


Figura 17: Exemplo de modo de jogo Computador vs Computador

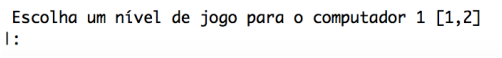


Figura 18: Exemplo de escolha de nível

Figura 19: Menu de fim de aplicação

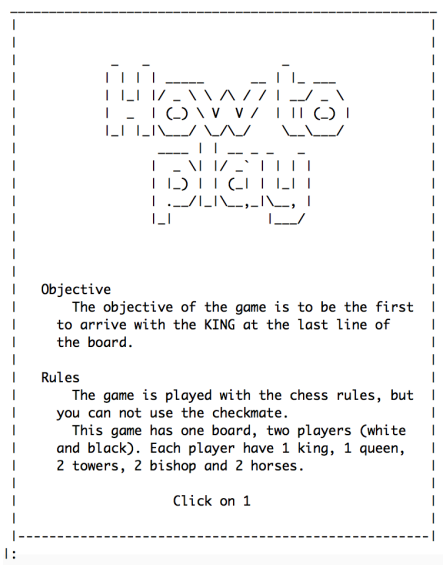


Figura 20: Menu de apoio ao jogo

5 Conclusão

A realização deste trabalho foi bastante útil para o aprofundamento da linguagem de programação em lógica, *Prolog*.

Em geral, o grupo está bastante satisfeito com o resultado final. Todos os objetivos propostos no trabalho foram cumpridos, desde um modo de jogo humano *vs* humano, humano *vs* computador e computador *vs* computador, vários níveis de dificuldade e uma interface gráfica bastante intuitiva. Um dos objetivos prioritários era a implementação de predicados simples e objetivos para a melhor compreensão do código, o que a nosso ver foi cumprido com bastante êxito.

Ao longo do trabalho deparamo-nos com algumas dificuldades, como por exemplo, criar níveis de jogo com alguma interatividade com o utilizador e não apenas dois níveis estáticos. Como o grupo estava pouca familiarizado com o Xadrez e também devido à falta de tempo não foi possível o melhoramento destes.

6 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

Anexo 1 Código Prolog

```
:- use_module(library(random)).
:- use_module(library(system)).

clearScreen:-
        write('\33\[2J'),nl.

% JOGADOR VS JOGADOR
startPvP(TAB,JOGADOR):-
        imprimirTabuleiro(TAB),
        getPosicaoInicial(TAB,JOGADOR,CASA),
        verificarAcabar(TAB,0,0,_,_),
        getPosicaoFinal(TAB,JOGADOR,CASA,GANHOU,NEWTAB),
        verificarAcabar(NEWTAB,GANHOU,1,JOGADOR,pvp).

% JOGADOR VS COMPUTADOR
startPCvP(TAB,JOGADOR,_,humano):-
        imprimirTabuleiro(TAB),
        getPosicaoInicial(TAB,JOGADOR,CASA),
        verificarAcabar(TAB,0,0,_,_),
        getPosicaoFinal(TAB,JOGADOR,CASA,GANHOU,NEWTAB),
        verificarAcabar(NEWTAB,GANHOU,1,JOGADOR,pcvp1).

startPCvP(TAB,JOGADOR,NIVEL,computador):-
        imprimirTabuleiro(TAB),
        getPosicaoInicial(TAB,NIVEL,JOGADOR,CASAI),
        verificarAcabar(TAB,0,0,_,_),
        getPosicaoFinal(TAB,NIVEL,CASAI,GANHOU,NEWTAB,computador),
        verificarAcabar(NEWTAB,GANHOU,1,JOGADOR,pcvp1).

% COMPUTADOR VS COMPUTADOR
```

```
startPCvPC(TAB,JOGADOR,NIVEL,NIVEL1):-
        sleep(3),
        imprimirTabuleiro(TAB),
        getPosicaoInicial(TAB,NIVEL,JOGADOR,CASAI),
        verificarAcabar(TAB,0,0,_,_),
        getPosicaoFinal(TAB,NIVEL,CASAI,GANHOU,NEWTAB,computador),
        verificarAcabar(NEWTAB,GANHOU,1,JOGADOR,NIVEL,NIVEL1,pcvpc).

% P vs P
getPosicaoInicial(TAB,JOGADOR,CASA):-
        write(' Jogador '), write(JOGADOR),write(' '),write('-'),write(' '),
        write(' Insira a posicao da peca que pretende '), nl,
        write(' mover no formato LINHA/COLUNA!'),nl,
        read(PL/PC),
        validarPosicaoInicialIntroduzida(TAB,PL,PC,JOGADOR,CASA).

getPosicaoInicial(TAB,JOGADOR,CASA):-
        clearScreen, imprimirTabuleiro(TAB),
        write('Posicao invalida. Insira novamente.'),nl,
        getPosicaoInicial(TAB,JOGADOR,CASA).

% PC vs P && PC vs PC
getPosicaoInicial(TAB,1,JOGADOR,CASAI):-
        jogarNivel1Inicial(TAB,JOGADOR,{PL,PC,_,_}),
        validarPosicaoInicialIntroduzida(TAB,PL,PC,JOGADOR,CASAI).

getPosicaoInicial(TAB,1,JOGADOR,CASAI):-
        getPosicaoInicial(TAB,1,JOGADOR,CASAI).

getPosicaoInicial(TAB,2,JOGADOR,CASAI):-
        jogarNivel2Inicial(TAB,JOGADOR,{PL,PC,_,_}),
        validarPosicaoInicialIntroduzida(TAB,PL,PC,JOGADOR,CASAI).

getPosicaoInicial(TAB,2,JOGADOR,CASAI):-
        getPosicaoInicial(TAB,2,JOGADOR,CASAI).

% P vs P
getPosicaoFinal(TAB,JOGADOR,CASA,GANHOU,NEWTAB):-
        write('Jogador '), write(JOGADOR),write(' '),write('-'),write(' '),
        write('Insira a posicao para onde pretende mover a peca no formato
LINHA/COLUNA!'),nl,
        read(FL/FC),
        validarPosicaoFinalIntroduzida(TAB,FL,FC,CASA,GANHOU,NEWTAB).

getPosicaoFinal(TAB,JOGADOR,CASA,GANHOU,NEWTAB):-
        clearScreen, imprimirTabuleiro(TAB),
        write('Posicao invalida. Insira novamente.'),nl,
        getPosicaoFinal(TAB,JOGADOR,CASA,GANHOU,NEWTAB).

% PC vs P && PC vs PC
getPosicaoFinal(TAB,1,CASA,GANHOU,NEWTAB,computador):-
         jogarNivel1Final(TAB,{PL,PC}),
         validarPosicaoFinalIntroduzida(TAB,PL,PC,CASA,GANHOU,NEWTAB),!,
         clearScreen.

getPosicaoFinal(TAB,1,CASA,GANHOU,NEWTAB,computador):-
        getPosicaoFinal(TAB,1,CASA,GANHOU,NEWTAB,computador).

getPosicaoFinal(TAB,2,CASA,GANHOU,NEWTAB,computador):-
         jogarNivel2Final({PL,PC}),
         validarPosicaoFinalIntroduzida(TAB,PL,PC,CASA,GANHOU,NEWTAB),!,
         clearScreen.

getPosicaoFinal(TAB,2,CASA,GANHOU,NEWTAB,computador):-
        getPosicaoFinal(TAB,2,CASA,GANHOU,NEWTAB,computador).

% TABULEIRO,GANHOU,DEVE_CONTINUAR,JOGADOR,TIPOJOGO
% PvP
verificarAcabar(TAB,0,1,branco,pvp):-startPvP(TAB,preto).
verificarAcabar(TAB,0,1,preto,pvp):-startPvP(TAB,branco).

%PCvP - Nivel 1
verificarAcabar(TAB,0,1,branco,pcvp1):-startPCvP(TAB,preto,1,computador).
```

```
verificarAcabar(TAB,0,1,preto,pcvp1):-startPCvP(TAB,branco,1,humano).

%PCvP - Nivel 2
verificarAcabar(TAB,0,1,branco,pcvp2):-startPCvP(TAB,preto,2,computador).
verificarAcabar(TAB,0,1,preto,pcvp2):-startPCvP(TAB,branco,2,humano).

%PCvP - Nivel 2
verificarAcabar(TAB,0,1,branco,NIVEL,NIVEL1,pcvpc):-startPCvPC(TAB,preto,NIVEL1,NIVEL).
verificarAcabar(TAB,0,1,preto,NIVEL,NIVEL1,pcvpc):-startPCvPC(TAB,branco,NIVEL1,NIVEL).

verificarAcabar(_,preto,_,_,pcvp1):-write('O COMPUTADOR (PRETO) GANHOU!').
verificarAcabar(_,preto,_,_,pcvp2):-write('O COMPUTADOR (PRETO) GANHOU!').
verificarAcabar(_,preto,_,_,_,_,pcvpc):-write('O COMPUTADOR 2 (PRETO) GANHOU!').
verificarAcabar(_,preto,_,_,_):-write('O JOGADOR PRETO GANHOU!').

verificarAcabar(_,branco,_,_,_,_,pcvpc):-write('O COMPUTADOR 1 (BRANCO) GANHOU!').
verificarAcabar(_,branco,_,_,_):-write('O JOGADOR BRANCO GANHOU!').

verificarAcabar(_,empate,_,_,_,_,pcvpc):-write('O JOGO EMPATOU!').
verificarAcabar(_,empate,_,_,_):-write('O JOGO EMPATOU!').

verificarAcabar(_,0,0,_,_).

validarPosicaoInicialIntroduzida(TAB,PL,PC,JOGADOR,CASA):-
        validarDentroTabuleiro(PL,PC),
        validarPecaIntroduzidaJogador(TAB,PL,PC,JOGADOR,CASA).

validarPosicaoFinalIntroduzida(TAB,FL,FC,CASA,GANHOU,L):-
        validarDentroTabuleiro(FL,FC),
        validarMovimento(CASA,{FL,FC}),
        validarIntersecaoEntrePecasBase(CASA,{FL,FC},TAB),
        mexerPecaBase(CASA,{FL,FC},TAB,L),
        verificarXeque(L),
        validarFimJogo(L,GANHOU).

jogarNivel1Inicial(TAB,JOGADOR,CASAI):-
        obterPecasTabuleiro(TAB,PECAS),
        getPosicaoInicialRandom(JOGADOR,PECAS,CASAI).

jogarNivel1Final(TAB,CASAF):-
        obterPecasTabuleiro(TAB,PECAS),
        getPosicaoFinalRandom(PECAS,CASAF).

jogarNivel2Inicial(TAB,JOGADOR,CASAI):-
        obterPecasTabuleiro(TAB,PECAS),
        getPosicaoInicialRandom(JOGADOR,PECAS,CASAI).

jogarNivel2Final(CASAF):-
        getPosicaoFinalRandom(CASAF).

getPosicaoInicialRandom(JOGADOR,PECAS,{LINHA,COLUNA,TIPO,JOGADOR}):-
        random(0,9,LINHA),
        random(0,9,COLUNA),
        member({LINHA,COLUNA,TIPO,JOGADOR},PECAS),!.

getPosicaoInicialRandom(JOGADOR,PECAS,{LINHA,COLUNA,TIPO,JOGADOR}):-
        getPosicaoInicialRandom(JOGADOR,PECAS,{LINHA,COLUNA,TIPO,JOGADOR}).

getPosicaoFinalRandom(PECAS,{LINHA,COLUNA}):-
        random(0,9,LINHA),
        random(0,9,COLUNA),
        \+member({LINHA,COLUNA,_,_},PECAS),!.

getPosicaoFinalRandom(PECAS,{LINHA,COLUNA}):-
        getPosicaoFinalRandom(PECAS,{LINHA,COLUNA}).

getPosicaoFinalRandom({LINHA,COLUNA}):-
        random(0,9,LINHA),
        random(0,9,COLUNA).

%%%%%%%%%%%%%%%%%%%%%%%

validarFimJogo(Tabuleiro,JOGADOR):-
```

```
        validarFim(Tabuleiro,1, JOGADOR).

validarFim([H|T], 1, empate):-
        validarEmpate(H,T), !.

validarFim([H|_], 1, JOGADOR):-
        !,
        validarReiLinha(H, JOGADOR).

validarFim([_|T], Count, J):-
        C1 is Count + 1,
        validarFim(T, C1, J).

validarEmpate(H,[H1|_]):-
        !,
        validarReiLinha(H1, preto),
        validarReiLinha(H, branco),
        validarReiPodeMexer(H1,H).

validarReiPodeMexer([{_,RX,k,preto}|_],LINHA):-
        RX1 is RX-1,
        RX1 > 0,
        verificarNaoHaPesaEm(RX1,LINHA).

validarReiPodeMexer([{_,RX,k,preto}|_],LINHA):-
        RX2 is RX,
        verificarNaoHaPesaEm(RX2,LINHA).

validarReiPodeMexer([{_,RX,k,preto}|_],LINHA):-
        RX3 is RX+1,
        RX3 <9,
        verificarNaoHaPesaEm(RX3,LINHA).

validarReiPodeMexer([_|T], LINHA):-
        validarReiPodeMexer(T, LINHA).

verificarNaoHaPesaEm(_,[]).

verificarNaoHaPesaEm(PX,[{_,X,_,_}|T]):-
        PX\=X,
        verificarNaoHaPesaEm(PX,T).

verificarNaoHaPesaEm(PX,[{}|T]):-
        verificarNaoHaPesaEm(PX,T).

validarReiLinha([], 0).

validarReiLinha([{_,_,k,JOGADOR}|_], JOGADOR):-!.

validarReiLinha([_|T],JOGADOR):-
        validarReiLinha(T,JOGADOR).

% true se nao existir xeque
verificarXeque(TAB):-
        getPosicoesRei(TAB,POSICAOB,POSICAOP),
        verificarXeque(TAB,POSICAOB,preto,TAB),
        verificarXeque(TAB,POSICAOP,branco,TAB).

verificarXeque([],_,_,_).

verificarXeque([H|T],POSICAOREI,JOGADOR,TAB):-
        verificarXequeRei(H,POSICAOREI,JOGADOR,TAB),
        verificarXeque(T,POSICAOREI,JOGADOR,TAB).

verificarXequeRei([],_,_,_).

verificarXequeRei([H|T],POSICAOR,JOGADOR,TAB):-
        verificarXequePecas(H,POSICAOR,JOGADOR,TAB),
        verificarXequeRei(T,POSICAOR,JOGADOR,TAB).

verificarXequePecas({PL,PC,q,JOGADOR},{PLR,PCR},JOGADOR,TAB):-
        verificarXequeQueen({PL,PC,q,JOGADOR},{PLR,PCR},JOGADOR,TAB).
```

```
verificarXequePecas({PL,PC,c,JOGADOR},{PLR,PCR},JOGADOR,TAB):-
        verificarXequeCavalo({PL,PC,c,JOGADOR},{PLR,PCR},JOGADOR,TAB).

verificarXequePecas({PL,PC,b,JOGADOR},{PLR,PCR},JOGADOR,TAB):-
        verificarXequeBispo({PL,PC,b,JOGADOR},{PLR,PCR},JOGADOR,TAB).

verificarXequePecas({PL,PC,t,JOGADOR},{PLR,PCR},JOGADOR,TAB):-
        verificarXequeTorre({PL,PC,t,JOGADOR},{PLR,PCR},JOGADOR,TAB).

verificarXequePecas({_,_,k,_},_,_,_).

verificarXequePecas({_,_,_,JOGADOR},_,JOGADOR2,_):-JOGADOR\=JOGADOR2.

verificarXequePecas({},_,_,_).

verificarXequeQueen({PL,PC,q,JOGADOR},{PLR,PCR},JOGADOR,_):-
        \+validarMovimentoRainha(PL,PC,PLR,PCR).

verificarXequeQueen({PL,PC,q,JOGADOR},{PLR,PCR},JOGADOR,TAB):-
        \+validarIntersecaoEntrePecasBase({PL,PC,q,JOGADOR},{PLR,PCR},TAB).

verificarXequeCavalo({PL,PC,c,JOGADOR},{PLR,PCR},JOGADOR,_):-
        \+validarMovimentoCavalo(PL,PC,PLR,PCR).

verificarXequeBispo({PL,PC,b,JOGADOR},{PLR,PCR},JOGADOR,_):-
        \+validarMovimentoBispo(PL,PC,PLR,PCR).

verificarXequeBispo({PL,PC,b,JOGADOR},{PLR,PCR},JOGADOR,TAB):-
        \+validarIntersecaoEntrePecasBase({PL,PC,b,JOGADOR},{PLR,PCR},TAB).

verificarXequeTorre({PL,PC,t,JOGADOR},{PLR,PCR},JOGADOR,_):-
        \+validarMovimentoTorre(PL,PC,PLR,PCR).

verificarXequeTorre({PL,PC,t,JOGADOR},{PLR,PCR},JOGADOR,TAB):-
        \+validarIntersecaoEntrePecasBase({PL,PC,t,JOGADOR},{PLR,PCR},TAB).

getPosicoesRei(TAB,POSICAOB,POSICAOP):-
        getPosicaoRei(branco,POSICAOB,TAB),
        getPosicaoRei(preto,POSICAOP,TAB).

getPosicaoRei(CORREI,{PL,PC},[LINHA|_]) :-
    member({PL,PC,k,CORREI},LINHA).

getPosicaoRei(CORREI,{PL,PC},[_|T]) :-
  getPosicaoRei(CORREI,{PL,PC},T).

mexerPecaBase({PL,PC,TIPO,JOGADOR},{FL,FC},TAB,L):-
        consumirPecaBase({PL,PC,TIPO,JOGADOR},{FL,FC},TAB,L,CONSUMIU),
        CONSUMIU>0.

mexerPecaBase({PL,PC,TIPO,JOGADOR},{FL,FC},TAB,L):-
        obterPecasTabuleiro(TAB,PECAS),
        \+member({FL,FC,_,_},PECAS),
        mexerPeca({PL,PC,TIPO,JOGADOR},{FL,FC},TAB,L2),
        apagarPeca({PL,PC,TIPO,JOGADOR},L2,L,1).

mexerPeca(PECA,TARGETPOSITION,TAB,L2):-
        mexerPecaRecurse(PECA,TARGETPOSITION,TAB,L2,_).

mexerPecaRecurse(_,_,[],[],8).

mexerPecaRecurse(PECA,TARGETPOSITION,[H|T],L2,LINHA):-
        mexerPecaRecurse(PECA,TARGETPOSITION,T,L3,LINHA2),
        LINHA is LINHA2-1,
        mexerPecaRecurseRecurse(PECA,TARGETPOSITION,H,L4,LINHA2,_),
        append([L4],L3,L2).

mexerPecaRecurseRecurse(_,_,[],[],_,8).

mexerPecaRecurseRecurse({PL,PC,TIPO,JOGADOR},{FL,FC},[H|T],L,LINHA,COLUNA):-
        mexerPecaRecurseRecurse({PL,PC,TIPO,JOGADOR},{FL,FC},T,L2,LINHA,COLUNA2),
        COLUNA is COLUNA2-1,
```

```
mexerPecaRecurseRecurseSet({PL,PC,TIPO,JOGADOR},{FL,FC},LINHA,COLUNA2,H,NEWPIECE),
        append([NEWPIECE],L2,L).

mexerPecaRecurseRecurseSet({_,_,TIPO,JOGADOR},{FL,FC},LINHA,COLUNA,{},{FL,FC,TIPO,JOGADO
R}):-
        LINHA=FL,
        COLUNA=FC.

mexerPecaRecurseRecurseSet(_,_,_,_,H,H).

consumirPecaBase({PL,PC,TIPO,JOGADOR},{FL,FC},TAB,L,CONSUMIU):-
        consumirPeca({PL,PC,TIPO,JOGADOR},{FL,FC},TAB,L2,CONSUMIU),
        apagarPeca({PL,PC,TIPO,JOGADOR},L2,L,CONSUMIU).

apagarPeca(_,L,L,0).

apagarPeca(_,[],[],_).

apagarPeca({PL,PC,TIPO,JOGADOR},[H|T],L,CONSUMIU):-
        CONSUMIU>0,
        apagarPeca({PL,PC,TIPO,JOGADOR},T,L2,CONSUMIU),
        apagarPecaRecurse({PL,PC,TIPO,JOGADOR},H,H2),
        append([H2],L2,L).

apagarPecaRecurse(_,[],[]).

%Goes through columns
apagarPecaRecurse({PL,PC,TIPO,JOGADOR},[H|T],L):-
        apagarPecaRecurse({PL,PC,TIPO,JOGADOR},T,L2),
        apagarPecaRecurseSubstitute({PL,PC,TIPO,JOGADOR},H,NEWPIECE),
        append([NEWPIECE],L2,L).

apagarPecaRecurseSubstitute({PL,PC,_,_},{CL,CC,_,_},{}):-
        CL=PL,
        PC=CC.

apagarPecaRecurseSubstitute(_,H,H).

consumirPeca(_,_,[],[],0).

%Goes through lines
consumirPeca({PL,PC,TIPO,JOGADOR},{FL,FC},[H|T],L,CONSUMIU):-
        consumirPeca({PL,PC,TIPO,JOGADOR},{FL,FC},T,L2,CONSUMIU1),
        consumirPecaRecurse({PL,PC,TIPO,JOGADOR},{FL,FC},H,H2,CONSUMIU2),
        CONSUMIU is CONSUMIU1+CONSUMIU2,
        append([H2],L2,L).

consumirPecaRecurse(_,_,[],[],0).

%Goes through columns
consumirPecaRecurse({PL,PC,TIPO,JOGADOR},{FL,FC},[H|T],L,CONSUMIU):-
        consumirPecaRecurse({PL,PC,TIPO,JOGADOR},{FL,FC},T,L2,CONSUMIU2),
        consumirPecaRecurseSubstitute({PL,PC,TIPO,JOGADOR},{FL,FC},H,NEWPIECE,CONSVAL),
        CONSUMIU is CONSUMIU2+CONSVAL,
        append([NEWPIECE],L2,L).

consumirPecaRecurseSubstitute({_,_,TIPO,JOGADOR},{FL,FC},{CL,CC,TIPO2,JOGADOR2},{FL,FC,T
IPO,JOGADOR},1):-
        CL=FL,
        FC=CC,
        JOGADOR\=JOGADOR2,
        TIPO2\=k,
        TIPO\=k.

consumirPecaRecurseSubstitute(_,_,H,H,0).

%Resultado: true se nao existir intersecao
validarIntersecaoEntrePecasBase(PECA,PF,TAB):-
        obterPecasTabuleiro(TAB,PECAS),
        validarIntersecaoEntrePecas(PECA,PF,PECAS).

obterPecasTabuleiro([],[]).
```

```
obterPecasTabuleiro([H|T],PECAS):-
        obterPecasTabuleiro(T,PECAS1),
        obterPecasTabuleiroRecurse(H,PECAS2),
        append(PECAS1,PECAS2,PECAS).

obterPecasTabuleiroRecurse([],[]).

obterPecasTabuleiroRecurse([H|T],PECAS):-
        obterPecasTabuleiroRecurse(T,PECAS1),
        setPecaEmLista(H,PECAS2),
        append(PECAS1,PECAS2,PECAS).

setPecaEmLista({A,B,C,D},[{A,B,C,D}]).

setPecaEmLista({},[]).

%true se nao existir intersecao
validarIntersecaoEntrePecas({PL,PC,TIPO,_},{FL,FC},PECAS):-
       validarConsumoEntrePecasRecurse({PL,PC,TIPO,_},{FL,FC},PECAS).

validarIntersecaoEntrePecas({_,_,c,_},{_,_},_).

validarConsumoEntrePecasRecurse(_,_,[]).

validarConsumoEntrePecasRecurse({PL,PC,TIPO,_},{FL,FC},[{CL,CC,_,_}|T]):-
        \+validarPontoNaLinha(PL,PC,CL,CC,FL,FC),
        validarConsumoEntrePecasRecurse({PL,PC,TIPO,_},{FL,FC},T).

validarPontoNaLinha(PL,PC,CL,CC,FL,FC):-
        validarLinha(PL,PC,CL,CC,FL,FC).

validarPontoNaLinha(PL,PC,CL,CC,FL,FC):-
        validarLinhaHorizontal(PL,PC,CL,CC,FL,FC).

validarPontoNaLinha(PL,PC,CL,CC,FL,FC):-
        validarLinhaVertical(PL,PC,CL,CC,FL,FC).

%Tanto faz utilizar xcomo y, desde que não sejam zero!
validarLinha(PL,PC,CL,CC,FL,FC):-
        NUMERADOR1 is CC-PC, % Linha Horizontal
        DIVISOR1 is FC-PC,   % Linha Horizontal
        NUMERADOR is CL-PL,
        DIVISOR is FL-PL,
        number(DIVISOR),
        DIVISOR\=0,
        DIVISOR1\=0,
        ALPHA1 is NUMERADOR1/DIVISOR1,
        ALPHA is NUMERADOR/DIVISOR,
        ALPHA1=ALPHA,
        ALPHA>0,
        ALPHA<1.

% X = P+ALPHA*(Q-P)
validarLinhaVertical(PL,PC,CL,_,FL,FC):-
        NUMERADOR is CL-PL,
        DIVISOR is FL-PL,
        DIFFH is FC-PC,
        DIVISOR\=0,
        DIFFH=0,
        ALPHA is NUMERADOR/DIVISOR,
        ALPHA>0,
        ALPHA<1.

% X = P+ALPHA*(Q-P)
validarLinhaHorizontal(PL,PC,_,CC,FL,FC):-
        NUMERADOR is CC-PC,
        DIVISOR is FC-PC,
        DIFFV is FL-PL,
        DIVISOR\=0,
        DIFFV=0,
        ALPHA is NUMERADOR/DIVISOR,
        ALPHA>0,
```

```
        ALPHA<1.

validarDentroTabuleiro(PL,PC):-
        PL>=1,
        PL=<8,
        PC>=1,
        PC=<8.

validarPecaIntroduzidaJogador(TAB,PL,PC,JOGADOR,{PL,PC,TIPO,JOGADOR}):-
        obterPecasTabuleiro(TAB,PECAS),
        member({PL,PC,TIPO,JOGADOR},PECAS).

validarMovimento({PL,PC,TIPO,_},{FL,FC}):-
        PL\=FL,
        validarMovimentoSub({PL,PC,TIPO},{FL,FC}).

validarMovimento({PL,PC,TIPO,_},{FL,FC}):-
        PC\=FC,
        validarMovimentoSub({PL,PC,TIPO},{FL,FC}).

validarMovimentoSub({PL,PC,t},{FL,FC}):-
        validarMovimentoTorre(PL,PC,FL,FC).

validarMovimentoSub({PL,PC,q},{FL,FC}):-
        validarMovimentoRainha(PL,PC,FL,FC).

validarMovimentoSub({PL,PC,k},{FL,FC}):-
        validarMovimentoRei(PL,PC,FL,FC).

validarMovimentoSub({PL,PC,b},{FL,FC}):-
        validarMovimentoBispo(PL,PC,FL,FC).

validarMovimentoSub({PL,PC,c},{FL,FC}):-
        validarMovimentoCavalo(PL,PC,FL,FC).

validarMovimentoCavalo(PL,PC,FL,FC):-
        AX is abs(PC-FC),
        AY is abs(PL-FL),
        validarMovimentoCavaloSub(AX,AY).

validarMovimentoCavaloSub(AX,AY):-AX=1,AY=2.

validarMovimentoCavaloSub(AX,AY):-AX=2,AY=1.

validarMovimentoRei(PL,PC,FL,FC):-
        validarMovimentoVerticalHorizontal(PL,PC,FL,FC),
        AX is abs(FC-PC),
        AY is abs(FL-PL),
        validarVHRei(AX,AY).

validarMovimentoRei(PL,PC,FL,FC):-
        validarMovimentoDiagonal(PL,PC,FL,FC),
        AX is abs(FC-PC),
        AY is abs(FL-PL),
        validarDGRei(AX,AY).

validarDGRei(AX,AX):-AX=1.

validarVHRei(AX,AY):-AX=0,AY=1.

validarVHRei(AX,AY):-AX=1,AY=0.

validarMovimentoTorre(PL,PC,FL,FC):-validarMovimentoVerticalHorizontal(PL,PC,FL,FC).

validarMovimentoRainha(PL,PC,FL,FC):-validarMovimentoVerticalHorizontal(PL,PC,FL,FC).

validarMovimentoRainha(PL,PC,FL,FC):-validarMovimentoDiagonal(PL,PC,FL,FC).

validarMovimentoBispo(PL,PC,FL,FC):-validarMovimentoDiagonal(PL,PC,FL,FC).

validarMovimentoVerticalHorizontal(PL,PC,FL,FC):-
        X is FC-PC,
        Y is FL-PL,
```

```
        validarMovimentoVH(X,Y).

validarMovimentoVH(X,Y):-
        X=0,Y\=0.
validarMovimentoVH(X,Y):-
        X\=0,Y=0.

validarMovimentoDiagonal(PL,PC,FL,FC):-
        X is FC-PC,
        Y is FL-PL,
        AX is abs(X),
        AY is abs(Y),
        AX=AY.

% Imprime as letras que permitem identificar uma coluna do tabuleiro
impimirIdentificadoresColunas:-
        write('   1     2     3     4     5     6     7     8').

% Lista com os numeros que permitem identificar uma linha do tabuleiro
numeroLinhas(['1','2','3','4','5','6','7','8']).

% 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, 1):-
        write('  '), write(Peca), write('  ').
imprimirCasa(_, _):-
        write('     ').

% Imprime as pecas que estao 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(TAB):-
        converterTabuleiro(TAB,[H|T]),
        imprimirSeparadorInicial, nl,
        numeroLinhas(ListaLinhas),
        imprimirLinhas([H|T], ListaLinhas), nl,
        impimirIdentificadoresColunas, nl.

converterTabuleiro([],[]).

converterTabuleiro([H|T],[TAB2|TAB1]):-
        converterTabuleiro(T,TAB1),
```

```
        converterTabuleiroRecurse(H,TAB2).

converterTabuleiroRecurse([],[]).

converterTabuleiroRecurse([H|T],[TAB2|TAB1]):-
        converterTabuleiroRecurse(T,TAB1),
        converterCasa(H,TAB2).

converterCasa({},vazio).

converterCasa({_,_,TIPO,branco},TIPO).

converterCasa({_,_,TIPO,preto},CASA):-
        char_code(TIPO,CODE),
        CODE1 is CODE-32,
        char_code(CASA,CODE1).

% Imprime o menu inicial
imprimirMenuInicial :-
        nl, nl, nl,
        write(' _____________________________________________________ '), nl,
        write(' |                                                   | '), nl,
        write(' |                                                   | '), nl,
        write(' |       ____                  _                     | '), nl,
        write(' |      |  _ \\     __ _     ___   (_)   ____    ____    | '), nl,
        write(' |      | |_) |   / _` |   / __|  | |  |  _ \\   / _  |   | '), nl,
        write(' |      |  _ <   | (_| |  | (__   | |  | | | | | (_| |   | '), nl,
        write(' |      |_| \\_\\   \\__,_|   \\___|  |_|  |_| |_|  \\__, |   | '), nl,
        write(' |                                            |___/    | '), nl,
        write(' |                 _   __   _                        | '), nl,
        write(' |                | | / /  (_)  ____     ____   ___   | '), nl,
        write(' |                |   /   | | |  _ \\    / _  | / __|   | '), nl,
        write(' |                |   \\   | | | | | |  (_| | \\__ \\    | '), nl,
        write(' |                |_|\\_\\  |_| |_| |_|   \\__, | |___/   | '), nl,
        write(' |                                     |___/         | '), nl,
        write(' |                                                   | '), nl,
        write(' |                                                   | '), nl,
        write(' |                  Beatriz Henriques                | '), nl,
        write(' |                    Beatriz Velho                  | '), nl,
        write(' |                  PLOG - FEUP 17/18                | '), nl,
        write(' |                                                   | '), nl,
        write(' |---------------------------------------------------| '), nl,
        write(' |                                                   | '), nl,
        write(' |                                                   | '), nl,
        write(' |         1. Start Game Player vs Player            | '), nl,
        write(' |         2. Start Game PC vs Player                | '), nl,
        write(' |         3. Start Game PC vs PC                    | '), nl,
        write(' |         4. How to play                            | '), nl,
        write(' |         5. Exit                                   | '), nl,
        write(' |                                                   | '), nl,
        write(' |                   Choose an option                | '), nl,
        write(' |___________________________________________________| '), nl.

% Imprime o menu de ajuda
imprimirMenuAjuda :-
        write(' _____________________________________________________ '), nl,
        write(' |                                                   | '), nl,
        write(' |                                                   | '), nl,
        write(' |                                                   | '), nl,
        write(' |             _   _                  _              | '), nl,
        write(' |            | | | | _____        __ | |_ ___        | '), nl,
        write(' |            | |_| |/ _ \\ \\ /\\ / / | __/ _ \\       | '), nl,
        write(' |            |  _  | (_) \\ V  V /  | || (_) |      | '), nl,
        write(' |            |_| |_|\\___/ \\_/\\_/    \\__\\___/       | '), nl,
        write(' |                  _____ | | __ _ _   _             | '), nl,
        write(' |                 |  _ \\\| |/ _` | | | |            | '), nl,
        write(' |                 | |_) | | (_| | |_| |            | '), nl,
        write(' |                 | .__/|_|\\__,_|\\__, |            | '), nl,
        write(' |                 |_|            |___/             | '), nl,
        write(' |                                                   | '), nl,
        write(' |                                                   | '), nl,
        write(' |                                                   | '), nl,
        write(' |   Objective                                       | '), nl,
        write(' |       The objective of the game is to be the first  | '), nl,
```

```
        write(' |     to arrive with the KING at the last line of     | '), nl,
        write(' |     the board.                                       | '), nl,
        write(' |                                                      | '), nl,
        write(' |   Rules                                              | '), nl,
        write(' |       The game is played with the chess rules, but   | '), nl,
        write(' |     you can not use the checkmate.                   | '), nl,
        write(' |       This game has one board, two players (white    | '), nl,
        write(' |     and black). Each player have 1 king, 1 queen,    | '), nl,
        write(' |     2 towers, 2 bishop and 2 horses.                 | '), nl,
        write(' |                                                      | '), nl,
        write(' |                    Click on 1                        | '), nl,
        write(' |                                                      | '), nl,
        write(' |------------------------------------------------------| '), nl.

% Imprime menu de saida
imprimirMenuExit :-
        write(' ______________________________________________________ '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |      _____                  _                      | '), nl,
        write(' |     |  _ \\      __ _    ___  (_)  ____     ____     | '), nl,
        write(' |     | |_) |    / _` |   / __| | | |  _ \\   / _  |    | '), nl,
        write(' |     |  _ <    | (_| |  | (__  | | | | | | | (_| |    | '), nl,
        write(' |     |_| \\_\\    \\__,_|   \\___| |_| |_| |_|  \\__, |    | '), nl,
        write(' |                                             |___/    | '), nl,
        write(' |           _   __   _                              | '), nl,
        write(' |          | |/ /  (_)  ____     ____    ___         | '), nl,
        write(' |          |   /   | | |  _ \\   / _  |  / __|        | '), nl,
        write(' |          | . \\   | | | | | | | (_| | \\__ \\         | '), nl,
        write(' |          |_|\\_\\  |_| |_| |_|  \\__, | |___/         | '), nl,
        write(' |                               |___/                | '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |            Thanks for playing our game!            | '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |                                                    | '), nl,
        write(' |------------------------------------------------------| '), nl,
        nl,
        nl.

% Selecciona o modo de jogo
selecionarModoJogo :-
        repeat,
        read(Action),
        Action > 0,
        Action < 6,
        modoJogo(Action).

% humano vs humano
modoJogo(1) :-
        comecarCorridaReis(T),
        startPvP(T,branco).

% computador vs humanos
modoJogo(2) :-
        comecarCorridaReis(T),
        lerNivelJogo(NIVEL,1),
        startPCvP(T,branco,NIVEL,humano).

% computador vs computador
modoJogo(3) :-
        comecarCorridaReis(T),
        lerNivelJogo(NIVEL1,1), % computador 1 - jogador branco
```

```
        lerNivelJogo(NIVEL2,2), % computador 2 - jogador preto
        startPCvPC(T,branco,NIVEL1,NIVEL2).

% menu de ajuda
modoJogo(4) :-
        imprimirMenuAjuda,
        repeat,
        read(Action),
        Action=1,
        iniciarJogo.

% sair
modoJogo(5) :-
        imprimirMenuExit.

modoJogo(_) :-
        write('Introduza uma opcao valida!').

lerNivelJogo(NIVEL,C):-
        write('Escolha um nivel de jogo para o computador '), write(C), write(' '),
write('[1,2]'),nl,
        read(NIVEL),
        (NIVEL=1;NIVEL=2).

lerNivelJogo(NIVEL,C):-
        lerNivelJogo(NIVEL,C).

iniciarJogo:-
        imprimirMenuInicial,
        write(' Selecione uma opcao:'), nl,
        selecionarModoJogo.

% Inicia o jogo Corrida de Reis
comecarCorridaReis(T):-
        inicializarTabuleiro(T).

% Inicializa o tabuleiro com as pecas na casa correta
inicializarTabuleiro(
[
[{},{},{},{},{},{},{},{}],
[{},{},{},{},{},{},{},{}],
[{},{},{},{},{},{},{},{}],
[{},{},{},{},{},{},{},{}],
[{},{},{},{},{},{},{},{}],
[{},{},{},{},{},{},{},{}],
[{7,1,k,preto},{7,2,t,preto},{7,3,b,preto},{7,4,c,preto},{7,5,c,branco},{7,6,b,branco},{
7,7,t,branco},{7,8,k,branco}],
[{8,1,q,preto},{8,2,t,preto},{8,3,b,preto},{8,4,c,preto},{8,5,c,branco},{8,6,b,branco},{
8,7,t,branco},{8,8,q,branco}]
]).
```