Faculdade de Engenharia da Universidade do Porto

Corrida de Reis

Programação em Lógica

2017/2018 -- Mestrado Integrado em Engenharia Informática e Computação:

Turma 3:

Afonso Bernardino da Silva Pinto

(up201503316@fe.up.pt)

Tomás Sousa Oliveira

(up201504746@fe.up.pt)

Docentes:

Henrique Cardoso hlc@fe.up.pt Rui Camacho rcamacho@fe.up.pt

Resumo

Criado no âmbito da disciplina Programação em Lógica leccionada no 3.º ano do Mestrado Integrado em Engenharia Informática e Computação, Corrida de Reis, uma variante popular de xadrez, é o tema deste projeto que tem como objetivo a sua implementação em linguagem Prolog.

Utilizando o SICStus Prolog como sistema de desenvolvimento, criamos um jogo de tabuleiro segundo um paradigma de programação em lógica capaz de simular fielmente a variante de xadrez, corrida de reis.

Foram implementadas as vertentes jogador vs jogador, jogador vs computador e computador vs computador, com a inclusão de 2 tipos de jogadas pelo computador.

O utilizador terá todas estas faculdades através de um interface sugestivo, em modo de texto.

Palavras-Chave

Corrida de Reis; Prolog; Programação em Lógica; FEUP, SICStus.

Agradecimentos

Este projeto foi resultado de diversas contribuições e colaborações, dada de forma direta e indireta, mas todas elas essenciais à sua realização. Gostaríamos assim de expressar os nossos sinceros agradecimentos a todos os que tornaram possível este trabalho.

Ao professor Henrique Cardoso pela orientação dada e valioso acompanhamento constante durante o desenvolvimento do projeto.

Índice

1. Introdução

Este projeto foi realizado no âmbito da unidade curricular Programação em Lógica leccionada no 3º ano do Mestrado Integrado em Engenharia Informática e Computação da Universidade do Porto.

O objetivo deste projeto é implementar a popular variante de xadrez, corrida de reis, em linguagem Prolog com as vertentes jogador vs jogador, jogador vs computador e computador vs computador.

O sistema de desenvolvimento utilizado será o SICStus Prolog de forma a facilitar a posterior integração de um visualizador 3D desenvolvida na unidade curricular de LAIG.

O presente relatório servirá para explorar as funcionalidades do jogo e fornecer detalhes sobre a sua implementação.

2. O Jogo

2.1 História

Inventado por Vernan R. Parton, em 1961, Corrida de Reis é uma popular variante do xadrez.

2.2 Detalhes do Jogo

O jogo é realizado num tabuleiro 8x8, com células alternadas de duas cores claras e escuras.

Normalmente, as linhas têm como notação números de 1 a 8 (de baixo para cima) e as colunas letras de A a H (da esquerda para a direita).

No estado inicial, as peças são posicionadas nas duas primeiras linhas do tabuleiro - as pretas no lado esquerdo e as brancas no lado direito.

Cada jogador começa com 8 peças da mesma cor, que consistem num rei, uma rainha, dois bispos, dois cavalos e duas torres.

Começa o jogo quem tem as peças brancas.

2.3 Objetivo

O objetivo do jogo é levar o rei até à última linha (linha 8) primeiro que o adversário.

2.4 Jogada

Em cada rodada, um jogador pode mover a sua peça para uma célula vazia ou capturar uma peça do adversário, movendo a sua peça para a célula em que essa peça se encontra.

2.5 Regras da Variante

Nesta variante, fazer check é totalmente proibido: não só é proibido colocar o próprio rei em check, como também é proibido colocar o rei oponente

Para além do supracitado, as peças movem-se e capturam-se precisamente como no xadrez normal.

2.6 Finalização

O jogo termina quando o jogador move o rei para a última linha. No entanto, se o rei branco chegar primeiro a essa linha, e o rei preto chega também na rodada seguinte, o jogo é considerado um empate.[1]

[1] Esta última regra foi criada para compensar joga com as peças pretas, por causa da vantagem de quem começa com as peças brancas.

3. Lógica do Jogo

3.1 Representação do Estado do Jogo

Optamos por representar o estado do jogo através de uma lista de 3 elementos:

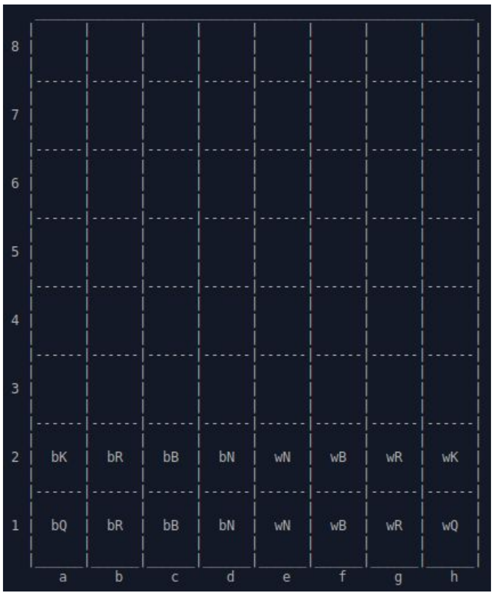
```
%%% Game[Board, gameState, gameMode];
Game = [Board, whiteToMove, pvp]
```

Representação do estado do jogo

O primeiro elemento da lista corresponde ao estado atual do tabuleiro. Este por si só é também uma lista, desta feita de peças[2]. Seguem-se alguns exemplos de possíveis tabuleiros:

- Tabuleiro no Estado Inicial:

```
[[NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece],
 [NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece],
 [NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece],
 [NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece],
 [NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece],
 [NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece],
 [BlackKing, BlackRook1, BlackBishop1, BlackKnigth1, WhiteKnigth1, WhiteBishop1, WhiteRook1, WhiteKing],
 [BlackQueen, BlackRook2, BlackBishop2, BlackKnigth2, WhiteKnigth2, WhiteBishop2, WhiteRook2, WhiteQueen]]
```



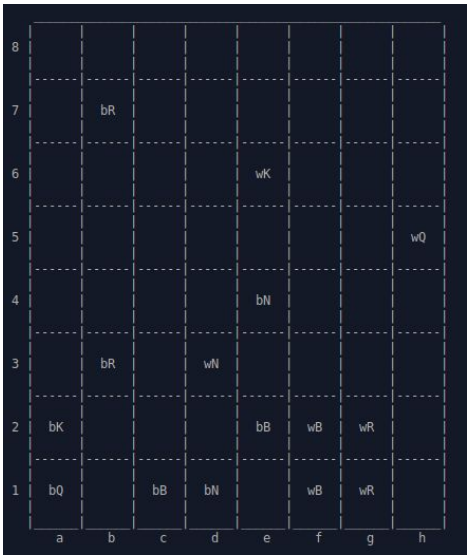
Estado Inicial do tabuleiro

---

[2] Cada peça corresponde a uma lista no formato: [Peça, Cor]

- Exemplo do Tabuleiro num Estado Intermédio:

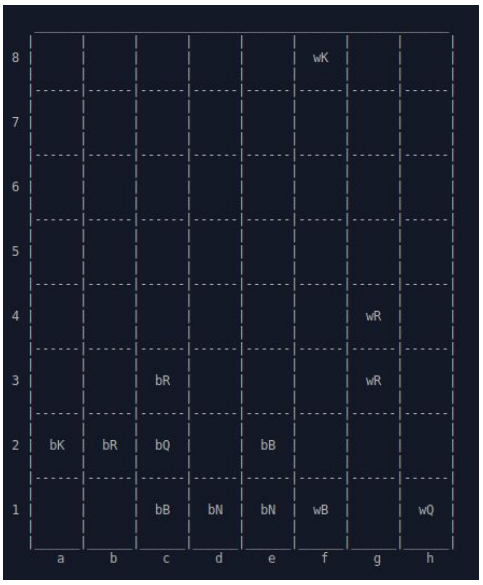
```
[NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece],
[NonePiece, BlackRook1, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece],
[NonePiece, NonePiece, NonePiece, NonePiece, WhiteKing, NonePiece, NonePiece, NonePiece],
[NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, WhiteQueen],
[NonePiece, NonePiece, NonePiece, NonePiece, BlackKnigth1, NonePiece, NonePiece, NonePiece],
[NonePiece, BlackRook2, NonePiece, WhiteKnigth1, NonePiece, NonePiece, NonePiece, NonePiece],
[BlackKing, NonePiece, NonePiece, NonePiece, BlackBishop2, WhiteBishop1, WhiteRook1, NonePiece],
[BlackQueen, NonePiece, BlackBishop1, BlackKnigth2, NonePiece, WhiteBishop2, WhiteRook2, NonePiece]]
```



Estado Intermédio do tabuleiro

- Exemplo do Tabuleiro num Estado Final:

```
[NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, WhiteKing, NonePiece, NonePiece],
[NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece],
[NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece],
[NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece],
[NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, NonePiece, WhiteRook1, NonePiece],
[NonePiece, NonePiece, BlackRook2, NonePiece, NonePiece, NonePiece, WhiteRook2, NonePiece],
[BlackKing, BlackRook1, BlackQueen, NonePiece, BlackBishop1, NonePiece, NonePiece, NonePiece],
[NonePiece, NonePiece, BlackBishop2, BlackKnigth1, BlackKnigth2, WhiteBishop1, NonePiece, WhiteQueen]]
```



Estado Final do tabuleiro

O segundo elemento da lista corresponde ao estado atual do jogo. Este campo é essencial para validar que peças podem ser movimentadas, ou para para concluir o jogo quando um dos estados finais é atingido.

O terceiro e último elemento da lista contém o modo de jogo: O modo **pvp** corresponde a humano vs humano, **pvc** a humano vs computador e **cvc** a computador vs computador.

3.2 Visualização do Tabuleiro

Os predicados responsáveis pela visualização do tabuleiro são apresentados de seguida. Os predicados foram implementados de forma recursiva. Para imprimir o tabuleiro basta chamar **printBoard(+Board)**.

```
printBoard(Board):-
  rowIDsList(RowIDs),
  write('    _____________________________________________________'), nl,
  write('   |     |     |     |     |     |     |     |     |'), nl,
  printBoard(Board, RowIDs).

printBoard([],[]):-
    printColumnIDs, nl, nl.

printBoard([Line|[]],[RowsIDTail|[]]):-
  write(RowsIDTail), write('|'),
  printLine(Line), nl,
  write('   |     |     |     |     |     |     |     |     |'), nl,
  write('   |_____|_____|_____|_____|_____|_____|_____|_____|'), nl,
  printBoard([],[]).

printBoard([Line|BoardTail], [RowsIDHead|RowsIDTail]) :-
  BoardTail \= [],
  write(RowsIDHead), write('|'),
  printLine(Line), nl,
  write('   |     |     |     |     |     |     |     |     |'), nl,
  write('   |-----|-----|-----|-----|-----|-----|-----|-----|'), nl,
  write('   |     |     |     |     |     |     |     |     |'), nl,
  printBoard(BoardTail,RowsIDTail).

printLine([]).
printLine([Piece|LineTail]):-
  getPieceSymbol(Piece, Symbol),
  write('  '), write(Symbol), write('  |'),
  printLine(LineTail).

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

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

Código referente à visualização do tabuleiro

3.3 Lista de Jogadas Válidas

Aqui podemos ver a lista de jogadas válidas implementadas para cada peça, em linguagem Prolog:

- Rei: Só pode movimentar-se 1 casa para qualquer lado:

```
validBasicMove('King', SrcCol, SrcRow, DestCol, DestRow, _):-
  DiffCols is abs(DestCol-SrcCol),
  DiffRows is abs(DestRow-SrcRow),
  DiffCols < 2,
  DiffRows < 2.
```

Código referente às jogadas válidas do Rei

- Rainha: Pode movimentar-se quantas casas quiser para qualquer lado:

```
validBasicMove('Queen', SrcCol, SrcRow, DestCol, DestRow, _):-
  (SrcCol == DestCol).
validBasicMove('Queen', SrcCol, SrcRow, DestCol, DestRow, _):-
  (SrcRow == DestRow).
validBasicMove('Queen', SrcCol, SrcRow, DestCol, DestRow, _):-
  DiffCols is abs(DestCol-SrcCol),
  DiffRows is abs(DestRow-SrcRow),
  DiffCols == DiffRows.
```

Código referente às jogadas válidas da Rainha

- Bispo: Pode movimentar-se quantas casas quiser apenas nas diagonais:

```
validBasicMove('Bishop', SrcCol, SrcRow, DestCol, DestRow, _):-
  DiffCols is abs(DestCol-SrcCol),
  DiffRows is abs(DestRow-SrcRow),
  DiffCols == DiffRows.
```

Código referente às jogadas válidas do Bispo

- Torre: Pode movimentar-se quantas casas quiser apenas na vertical ou na horizontal:

```
validBasicMove('Rook', SrcCol, SrcRow, DestCol, DestRow, _):-
  (SrcCol == DestCol).
validBasicMove('Rook', SrcCol, SrcRow, DestCol, DestRow, _):-
  (SrcRow == DestRow).
```

Código referente às jogadas válidas da Torre

- Cavalo: Pode movimentar-se apenas em “L”, ou seja, duas casas para a frente e uma para a esquerda ou direita. Esta é a única peça que pode pular outras peças:

```
validBasicMove('Knight', SrcCol, SrcRow, DestCol, DestRow, _):-
  DiffCols is abs(DestCol-SrcCol),
  DiffRows is abs(DestRow-SrcRow),
  DiffCols == 2,
  DiffRows == 1.

validBasicMove('Knight', SrcCol, SrcRow, DestCol, DestRow, _):-
  DiffCols is abs(DestCol-SrcCol),
  DiffRows is abs(DestRow-SrcRow),
  DiffCols == 1,
  DiffRows == 2.
```

Código referente às jogadas válidas do Cavalo

Para além dos movimentos básicos de cada peça vale a pena realçar que nenhuma peça além do cavalo, pode saltar outras peças que estejam no caminho, e ainda que qualquer movimento que coloque um dos reis em *check* é proibido.

Cada jogador apenas pode mover peças da sua cor e não é possível mover a peça para o mesmo local onde esta já se encontrava.

3.4 Execução de Jogadas

Após determinadas as coordenadas originais e de destino da peça a mover (por solicitação no caso do modo com utilizador humano ou calculadas segundo determinado algoritmo no caso do modo com computador) a jogada tem de ser validada pelo predicado ***validateMove(+SrcCol, +SrcRow, +DestCol, +DestRow, +Board, +Flag)*:**

- **differentPositions(+SrcCol, +SrcRow, +DestCol, +DestRow, +Flag);**
- **differentColors(+SrcCol, +SrcRow, +DestCol, +DestRow, +Board, +Flag);**
- **validBasicMove(+PieceName, +SrcCol, +SrcRow, +DestCol, +DestRow, +Flag);**
- **checkForJumping(+PieceName, +SrcCol, +SrcRow, +DestCol, +DestRow, +Board, +Flag);**
- **checkForCheck(+TempBoard, +Flag);**

Cada um destes predicados contidos em *validateMove* servem para validar se o movimento está em conformidade com as regras do jogo.

Se a jogada for válida, o movimento é então efetivamente realizado através do predicado ***makeMove(+Board, +SrcCol, +SrcRow, +DestCol, +DestRow, -NextBoard)*** e de seguida o estado do jogo é atualizado pelo predicado ***updateGameState(+Game, +NextBoard, -ContinueGame)*.**

Estes predicados encontram-se dentro de um predicado responsável por gerir e mostrar o jogo em conformidade com o estado e modo atual.
***playGame(+Game)* .**

3.5 Avaliação do Tabuleiro

A avaliação do tabuleiro é feita através do predicado ***evaluatePosition(+Color, +Board, -Value)***.

Esta avaliação é feita tendo não só em conta as peças existentes em jogo (da cor do jogador em causa), como também a posição do rei no tabuleiro.

O valor relativo de cada peça é obtido através de:

```
getPieceValue('King', Value):-
  Value = 900.
getPieceValue('Queen', Value):-
  Value = 90.
getPieceValue('Rook', Value):-
  Value = 50.
getPieceValue('Bishop', Value):-
  Value = 30.
getPieceValue('Knight', Value):-
  Value = 25.
```

Valor relativo de cada peça de acordo com a comunidade de Xadrez

A fórmula final aplicada é:

$$\sum_{i=0}^{i=\text{última peça da cor}} (valor\ da\ peça) + (100 * Linha\ do\ Rei) + \begin{cases} 10, se\ for\ a\ sua\ vez\ de\ jogar \\ 0, se\ não \end{cases}$$

Fórmula Final

Este predicado é utilizado no predicado ***printGameInfo(+Game)*** com o objetivo de mostrar ao utilizador a avaliação relativa do tabuleiro no momento da jogada.

$$Força\ Relativa\ =\ Força\ Brancos\ -\ Força\ Pretos$$

3.6 Final do Jogo

Sempre que o estado do jogo é atualizado pelo predicado ***updateGameState(+Game, +NextBoard, -ContinueGame)*** verifica-se se o jogo termina.

Existem 3 possibilidades de um jogo terminar:

- as brancas vencem[3];
- as pretas vencem;
- ocorre um empate[4].

Todas as possibilidades estão implementadas pelo predicado ***gameOver(+Game, +NextBoard, -ContinueGame).***

A atualização destas informações é depois interpretada pelo predicado ***playGame(+Game)***, conjuntamente com o predicado **isItOver(+Game)**, que posteriormente disponibilizam a informação ao utilizador.

3.7 Jogada do Computador

Foram implementados dois tipos de jogadas de computador: uma ***random***, que executa jogadas aleatórias; e uma ***someHowSmart***, que tenta fazer o rei subir antes de partir para movimentos random.

As jogadas deste último são efetuadas a partir do predicado ***somehowSmartMove(+Move, +SrcCol, +SrcRow, +DestCol, +DestRow, +Board)*** - este predicado é utilizado para variar a escolha da coluna para qual o rei tenta subir - dentro do predicado ***somehowSmartBotTurn(Game, ContinueGame)***.

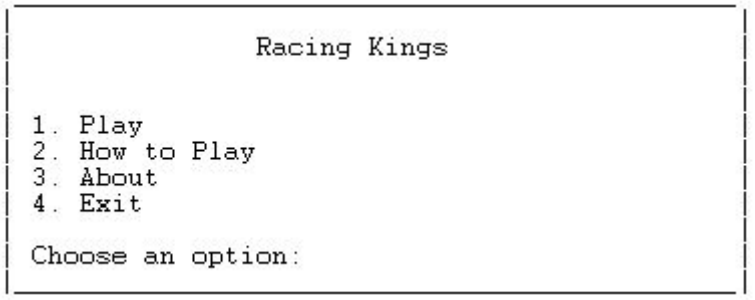
Caso não seja possível movimentar o rei de forma ascendente no tabuleiro o computador tenta um movimento aleatório conforme enunciado a seguir.

Os movimentos aleatórios são produzidos no predicado ***botTurn(Game, ContinueGame)***. Aqui é escolhido uma linha e coluna aleatoriamente até ser selecionado uma peça da cor de quem vai jogar, segue-se a escolha aleatória de uma coluna e linha de destino. Este processo é repetido até ser encontrado um movimento válido.

[3] As brancas podem ganhar de duas maneiras distintas quando chegam com o rei até à última linha: quando as pretas não têm a possibilidade de empatar e quando as pretas têm, mas não o fazem.
[4] Um empate pode ocorrer quando o rei branco chega à última linha sendo imediatamente seguido pelo rei preto ou por ocorrer um stalemate (um jogador ficar sem jogadas possíveis).

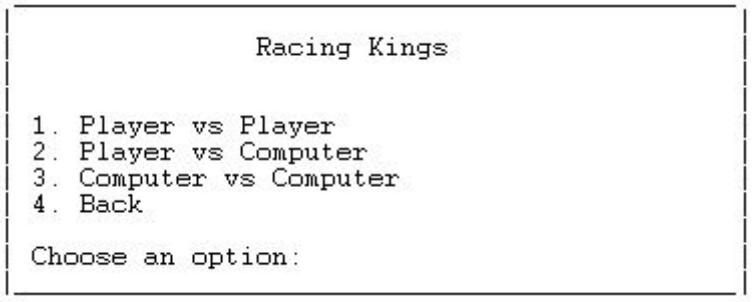
4. Interface com o Utilizador

Quando o utilizador inicia o jogo através do predicado *racingKings*, é apresentado o menu principal:



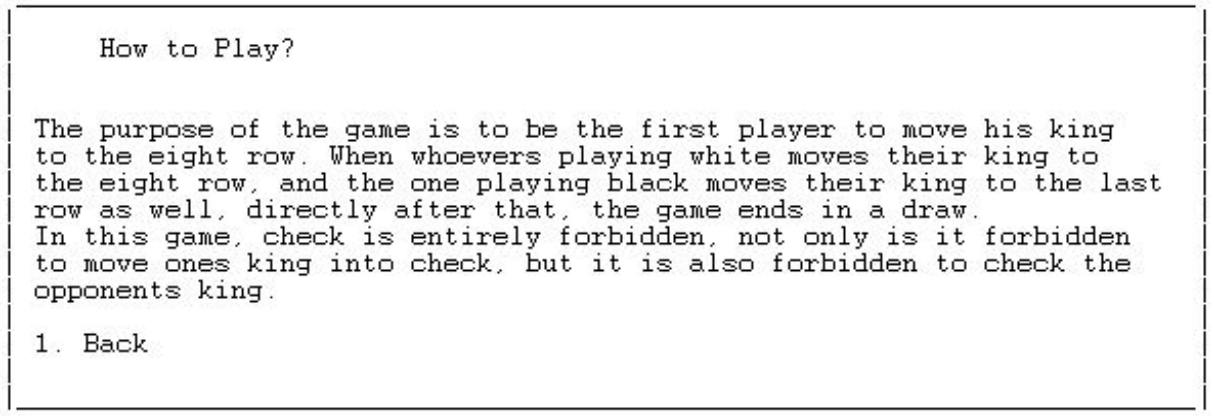
Menu Inicial

A primeira opção é para escolher o modo de jogo, que por sua vez tem as opções Player vs Player (jogar com dois jogadores), Player vs Computador (jogar contra o computador) e Computador vs Computador (ver um jogo entre dois computadores):



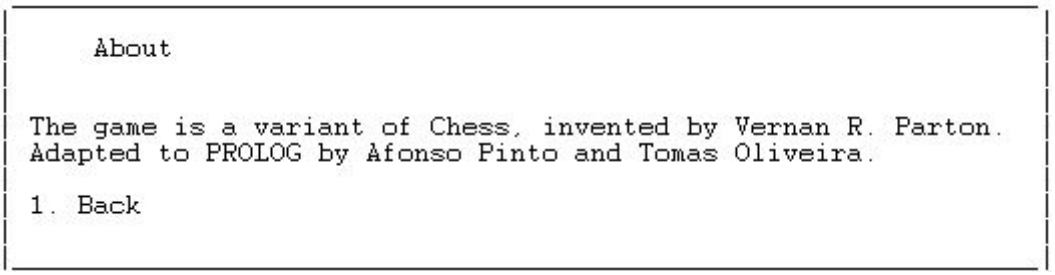
Menu de Jogo

A segunda opção mostra um menu onde é explicado ao utilizador como jogar esta variante do Xadrez:



Como Jogar

A terceira opção mostra o utilizador curiosidades sobre a adaptação do jogo para Prolog:



Sobre o Jogo

A última opção serve para o utilizador sair do jogo.

5. Conclusões

Com a realização deste projeto pudemos familiarizar-mos-nos com a linguagem de programação Prolog.

Os objetivos propostos foram cumpridos.

O projeto poderia ser melhorado com a inclusão de um algoritmo minimax, segundo uma heurística adequada ao jogo implementado, de forma a melhorar a inteligência artificial.

Bibliografia

- BrainKing - Game rules (Racing Kings). Accessed November 12, 2017. https://brainking.com/en/GameRules?tp=125
- "Lichess.org." Racing Kings • Race your King to the eighth rank to win. • lichess.org. Accessed November 12, 2017. https://lichess.org/variant/racingKings

Anexos

O código implementado está anexado, juntamente com este relatório.