Barragoon

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Barragoon_1:
Bruno Pinto - 201502960
João Mendes - 201505439

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

17 de Novembro de 2017

Resumo

O objetivo do trabalho foi o de implementar, em linguagem Prolog, o jogo de tabuleiro Barragoon, permitindo três modos de utilização: Humano/Humano, Humano/Computador e Computador/Computador e incluindo dois níveis de jogo para o computador e uma interface com o utilizador, em modo de texto.

O jogo foi desenvolvido por partes, utilizando uma abordagem topdown. Primeiro foi realizada a instanciação, manipulação e visualização do tabuleiro de jogo. Esta implementação foi bastante intuitiva, passamos depois ao desenvolvimento da interface com utilizador por forma a correr os primeiros testes, implementando para o efeito as regras mais básicas do jogo como não comer as próprias peça ou verificar se o jogo terminou (verificar a ausência de peças de uma cor).

A parte mais morosa foi o estabelecimento de regras para os três tipos de peças diferentes de cada jogador, e as duas jogadas possíveis para cada uma delas, um total de 6 possibilades combinadas com as 6 peças barragoon diferentes que podem afetar a jogada.

O paradigma da linguagem Prolog era algo ao qual não estávamos habituados, mas graças ao trabalho, consultas dos materias e professores, podemos dizer que os objetivos foram alcançados. Como resultado disso, podemos apresentar um jogo, que não só divertido, requer um nível alto de prática e estratégia.

Como conclusão final, este trabalho contribui imenso para a nossa melhor compreensão da linguagem e sobretudo aplicá-la da forma mais eficiente; entendemos que o objetivo do trabalho seria a consolidação das matérias da Unidade Curricular até à data e pensamos que essa premissa foi concluída com sucesso.

Índice

1 Introdução

Este trabalho foi realizado no âmbito da unidade curricular de Programação em Lógica, do primeiro semestre do terceiro ano do Mestrado Integrado em Engenharia Informática e Computação, da Faculdade de Engenharia da Universidade do Porto.

A escolha deste tema deveu-se ao facto de ser um jogo apelativo, com um conceito diferente dos jogos de tabuleiros a que estamos familiarizados, com muitas mais regras e diversidade nas peças. A ideia do jogo é interessante no sentido em que existem cada vez mais peças Barragoon no tabuleiro que dificultam a estratégia aos jogadores.

O objetivo do trabalho foi o de implementar, em linguagem Prolog, o jogo de tabuleiro Barragoon, que permitindo três modos de utilização: Humano/Humano, Humano/Computador e Computador/Computador e incluindo dois níveis de jogo para o computador e uma interface com o utilizador, em modo de texto.

O seguinte relatório descreve o jogo de tabuleiro Barragoon e implementação da sua lógica em Prolog, assim como o modo de funcionamento do módulo de interface com o utilizador em modo de texto.

2 O Barragoon

O Barragoon é um jogo de estratégia, em tabuleiro, para dois jogadores.

Cada jogador começa com as suas sete peças de jogo, cada uma representando o número de casas que podem avançar.

Para além das peças dos jogadores, existem ainda as chamadas peças Barragoon, que não podem ser movidas. São oito e representam a forma como cada peça dos jogadores pode saltar sobre elas. A estas somam-se outras vinte e quatro que começam de fora do jogo, entrando depois no decorrer do jogo sob condições adiante explicadas.

O objetivo do jogo passa por capturar as peças do jogador adversário ou impedi-lo de avançar as suas peças.

2.1 Peças Barragoon

As peças Barragoon são o elemento central do jogo e permitem bloquear ou permitir o avanço das peças do jogo, em direções específicas, dependendo da direção indicada pelo seu símbolo.

Os seus símbolos são:

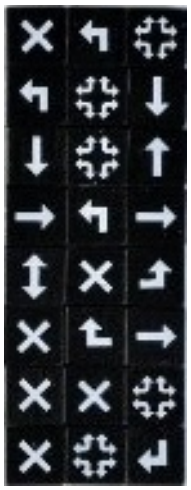

Figura 1: Peças Barragoon

- **'No Entry'** - A peça Barragoon não pode ser atravessada.
- **'One Way'** - A peça Barragoon apenas pode ser atravessada por qualquer das peças na direção indicada pela seta.
- **'Two Ways'** - A peça Barragoon apenas pode ser atravessada por qualquer das peças numa das duas direções indicadas pela seta.
- **'Right Turn'** - A peça Barragoon apenas pode ser atravessada por qualquer das peças com uma curva à direita na direção indicada pela seta.
- **'Left Turn'** - A peça Barragoon apenas pode ser atravessada por qualquer das peças com uma curva à esquerda na direção indicada pela seta.
- **'All Turns'** - A peça Barragoon apenas pode ser atravessada por qualquer das peças com qualquer uma das curvas indicadas pelas oito setas.

A colocação das peças Barragoon, à exceção da colocação inicial, é feita de forma a que fique a apontar na direção escolhida, quando aplicável.

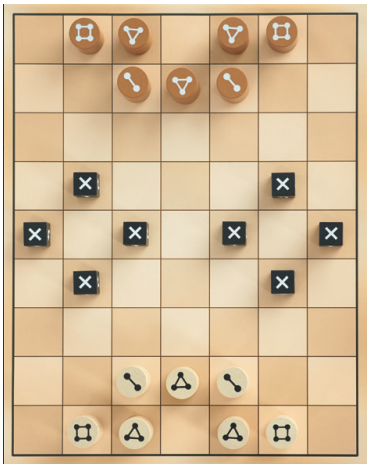

Figura 2: Tabuleiro

2.2 Peças dos Jogadores

Há três tipos de peças:

- **'Peça de Dois Espaços'** - Como o valor indica, esta peça pode ser movida duas ou, alternativamente, uma casa em *short move*.
- **'Peça de Três Espaços'** - Como o valor indica, esta peça pode ser movida três ou, alternativamente, duas casas em *short move*.
- **'Peça de Quatro Espaços'** - Como o valor indica, esta peça pode ser movida quatro ou, alternativamente, três casas em *short move*.

Na figura 1 podemos ver os 3 tipos de peças diferentes de jogadores (em branco e castanho) cujo número de pontos da figura que nela consta refere o máximo de movimentos numa jogada.

A contagem de casas avançadas começa na primeira casa após a inicial e inclui a final. As peças não podem ser movidas na diagonal e só podem mudar de direção num ângulo de 90^{o} e apenas uma vez durante uma jogada. Uma peça Barragoon pode ser atravessada durante a jogada se a direção indicada por essa peça permitir, sendo que a casa ocupada pela peça Barragoon também é contada no número de casas avançadas. A curva indicada pela peça Barragoon conta para o limite de curvas possíveis numa única jogada. Não é possível saltar sobre as próprias peças ou as peças do adversário, nem é possível capturar as próprias peças.

Uma peça pode se mover e capturar outras peças em *full move*, sendo que as *short moves* servem apenas para se mover. Os movimentos podem ser feitos para uma posição desocupada, para uma posição ocupada por uma peça do adversário, sendo esta capturada, ou para uma posição ocupada por uma peça Barragoon, capturando-a, à exceção das Peças de Dois Espaços, que não podem capturar uma peça Barragoon Todas as Curvas. Quando uma peça do jogador é capturada, são recolhidas duas peças Barragoon daquelas que estão de fora, sendo colocadas em posições livres à escolha, primeiro pelo jogador que viu a sua peça ser capturada e depois pelo outro. Quando uma peça Barragoon é capturada, é colocada pelo jogador que a capturou numa posição livre à escolha.

3 Lógica do Jogo

3.1 Representação do Estado do Jogo

As peças de jogador são representadas por números (correspondentes ao número máximo de movimentos dessa peça) e uma letra a representar a sua cor, e as células vazias por ' '.

As peças barragoon são representadas por letras (ver 1.1):

- **no**: No Entry;
- **ox**: One Way - com o x a poder ser R, L, T, B, dependendo de se a peça aponta para a direita, a esquerda, o topo ou o inferior do tabuleiro;
- **tx**: Two Ways - com o x a poder ser H, V, dependendo de se a peça aponta na vertical ou na horizontal;
- **rx**: Right Turn - com o x a poder ser R, L, T, B, dependendo de se a peça aponta para a direita, a esquerda, o topo ou o inferior do tabuleiro;;
- **lx**: Left Turn - com o x a poder ser R, L, T, B, dependendo de se a peça aponta para a direita, a esquerda, o topo ou o inferior do tabuleiro;;
- **at**: All Turns.

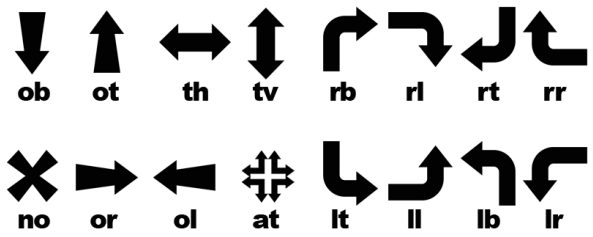

Figura 3: Representação dos barragoons em modo de texto

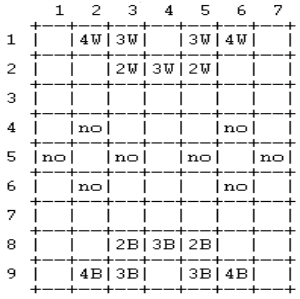

Figura 4: Estado inicial do tabuleiro com output da consola

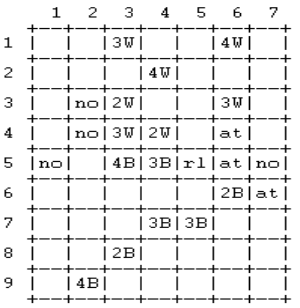

Figura 5: Estado intermédio do tabuleiro com output da consola

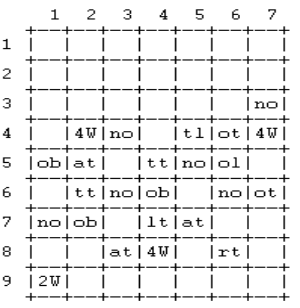

Figura 6: Estado final do tabuleiro com output da consola

3.2 Visualização do Tabuleiro

Para a visualização do tabuleiro foram construídos os seguintes predicados Prolog:

- **'gamePrint(+Board)'** - Inicia a recursão que imprime o tabuleiro de jogo.

- **'printBlackLine'** - Faz a impressão da linha divisória das linhas da tabuleiro.

- **'printRowByRow(+NoLine, +Matrix)** - Faz a impressão de cada uma das linhas da matriz.

- **'printSingleRow(+Row)'** - Faz a impressão de uma das linhas da matriz.

3.3 Lista de Jogadas Válidas

A lista de jogadas válidas é obtida através da função **'findall/3'**, usada com o predicado **'movePiece(+CurrPlayer, +BoardIn, +PieceLine, +PieceColumn, +MoveLine, +MoveColumn, -OutBoard)'** para obter a lista de tabuleiros possíveis e posição do movimento que representam.

3.4 Execução de Jogadas

A execução de jogadas é feita com recurso a uma função que faz a leitura da jogada do jogador humano **'readMove(-PieceLine, -PieceColumn, -MoveLine, -MoveColumn)'**, seguida de uma validação dessa mesma jogada, com **'validateMove(+CurrPlayer, +BoardIn, +PieceLine, +PieceColumn, +MoveLine, +MoveColumn)'**.

3.5 Avaliação do Tabuleiro

Cada tabuleiro, representando cada jogada possível, é avaliado em função de número de peças dos jogadores e peças Barragoon, sendo considerados mais valiosos aqueles em que o número de peças dos jogadores é menor ou, em caso de empate, aquele em que o número de barragoons é menor.

Para isso são usados os predicados de contagem **'countBarragoons(+Board, -NoBarragoons)'** e **'countPieces(+Board, -NoPieces)'**.

3.6 Final do Jogo

A verificação do fim do jogo é feita com recurso à função **'gameOver(+Board, -Loser)'**, que faz uso de uma outra função **'noPiece(+Board, +Color)'**, que verifica a existência de peças de jogo de uma dada cor no tabuleiro de jogo.

3.7 Jogada do Computador

As jogadas do CPU são feitas a partir dos predicados **'playCPUvsCPU-random(+CurrPlayer, +BoardIn, -BoardOut)'**, no caso de uma jogada aleatória, ou **'pcMove(+CurrPlayer, +BoardIn, -BoardOut)**, no caso de uma jogada artificialmente inteligente.

4 Interface com o Utilizador

Antes do início do jogo é apresentado um menu com os três diferentes modos de utilização, para escolha do desejado.

A representação do estado de jogo é feita através da impressão do tabuleiro e durante o ciclo de jogo são feitos pedidos de introdução da jogada a realizar pelo jogador corrente.

A introdução da jogada consiste na escolha da peça a mover, através da introdução da sua linha e coluna, e do local para onde se pretende mover a peça, também através da introdução da linha e coluna desejada.

```
*** BARRAGOON ***
1. Human vs Human
2. Human vs CPU
3. CPU vs CPU
Select an option (0 to exit): 1.

    1  2  3  4  5  6  7
  +--+--+--+--+--+--+--+
1 |  |3W|3W|  |3W|4W|  |
  +--+--+--+--+--+--+--+
2 |  |  |2W|3W|2W|  |  |
  +--+--+--+--+--+--+--+
3 |  |  |  |  |  |  |  |
  +--+--+--+--+--+--+--+
4 |  |no|  |  |  |no|  |
  +--+--+--+--+--+--+--+
5 |no|  |no|  |no|  |no|
  +--+--+--+--+--+--+--+
6 |  |no|  |  |  |no|  |
  +--+--+--+--+--+--+--+
7 |  |  |  |  |  |  |  |
  +--+--+--+--+--+--+--+
8 |  |  |2B|3B|2B|  |  |
  +--+--+--+--+--+--+--+
9 |  |4B|3B|  |3B|4B|  |
  +--+--+--+--+--+--+--+

W turn

*** SELECT PIECE ***
Line: 2.

Column: 3.


*** SELECT MOVE ***
Line: 3.

Column: 3.
```

Figura 7: Interface em modo de texto

5 Conclusões

O resultado final que obtivemos e os conhecimentos adquiridos durante o desenvolvimento do projeto foram positivos. O jogo decorre de forma fluída em todos os modos de jogo, sendo que o modo de Jogador vs Jogador é o mais usador por nós pela competição mas também porque apesar de ter dado trabalho, Barragoon é um jogo divertido e estratégico ao mesmo tempo.

O modo Computador vs Computador foi o mais complicado, mas foi concluído. A inteligência artificial poderia ter sido melhor explorado com um pouco mais de tempo e com menos carga de trabalho noutras unidades curriculares.

Gostamos da experiência de desenvolvimento de um jogo na linguagem PROLOG. Ao contrário do que estamos habituados, este tipo de linguagem requer um pensamento lógico em cada predicado desenvolvido. Achamos que saímos deste projeto com um raciocínio mais apurado e outra forma de ver a programação em si

Anexos

CPUvsCPU.pl

```
/*
* CPU vs CPU random playing
*/
playCPUvsCPUrandom(CurrPlayer, BoardIn, BoardOut):-
  gamePrint(BoardIn),
  nl, write(CurrPlayer), write(' turn'), nl,
  findall(XBoard-C-D, A^B^C^D^movePiece(CurrPlayer, BoardIn, A,
  ↪  B, C, D, XBoard), PossiblePlays),
  (
     PossiblePlays \= []
  ;
     changePlayer(CurrPlayer, NewPlayer),
     gamePrint(BoardIn), nl,
     write(NewPlayer), write(' won!'),
     abort
  ),
  random_permutation(PossiblePlays,
  ↪  [NewBoard-MoveLine-MoveColumn|_]),
  getPiece(BoardIn, MoveLine, MoveColumn, Piece),
  (
    gameOver(NewBoard, Loser),
    showResult(Loser),
    abort
  ;
    barragoon(Piece),
    putBarragoonRandom(NewBoard, Board1),
    copy_term(Board1, BoardOut)
  ;
    getColor(Piece, Color),
    Color \= ' ',
    putBarragoonRandom(NewBoard, Board1),
    putBarragoonRandom(Board1, Board2),
    copy_term(Board2, BoardOut)
  ;
    copy_term(NewBoard, BoardOut)
  ),
  get_code(_X).

/*
* Random CPU vs CPU game mode
*/
playRandomCPUvsCPU:-
  initialBoard(BoardIn),
  initialPlayer(PlayerIn),
  assert(board(BoardIn)),
  assert(player(PlayerIn)),
  repeat,
    retract(board(BoardCurr)),
```

```
47      retract(player(PlayerCurr)),
48      once(playCPUvsCPUrandom(PlayerCurr,BoardCurr, BoardOut)),
49      once(changePlayer(PlayerCurr, NewPlayer)),
50      assert(player(NewPlayer)),
51      assert(board(BoardOut)),
52      gameOver(BoardOut, Loser),
53      showResult(Loser),
54      !.
55
56  /*
57  * AI CPU vs CPU game mode
58  */
59  playAICPUvsCPU:-
60    initialBoard(BoardIn),
61    initialPlayer(PlayerIn),
62    assert(board(BoardIn)),
63    assert(player(PlayerIn)),
64    repeat,
65      retract(board(BoardCurr)),
66      retract(player(PlayerCurr)),
67      once(pcMove(PlayerCurr,BoardCurr, BoardOut)),
68      once(changePlayer(PlayerCurr, NewPlayer)),
69      assert(player(NewPlayer)),
70      assert(board(BoardOut)),
71      gameOver(BoardOut, Loser),
72      showResult(Loser),
73      !.
```

HvsCPU.pl

```
1   /*
2   * H vs CPU random playing
3   */
4   playCPUvsHrandom(CurrPlayer, BoardIn, BoardOut):-
5           findall(XBoard, A^B^C^D^movePiece(CurrPlayer, BoardIn, A,
            ↪  B, C, D, XBoard), PossiblePlays),
6           (
7                   PossiblePlays \= []
8           ;
9                   changePlayer(CurrPlayer, NewPlayer),
10                  gamePrint(BoardIn), nl,
11                  write(NewPlayer), write(' won!'),
12                  abort
13          ),
14          nl, write(CurrPlayer), write(' turn'), nl, nl,
15    findall(XBoard-C-D, A^B^C^D^movePiece(CurrPlayer, BoardIn, A,
      ↪  B, C, D, XBoard), PossiblePlays),
16          random_permutation(PossiblePlays,
            ↪  [NewBoard-MoveLine-MoveColumn|_]),
17    getPiece(BoardIn, MoveLine, MoveColumn, Piece),
18    (
19      gameOver(NewBoard, Loser),
```

```
20      showResult(Loser),
21      abort
22    ;
23      barragoon(Piece),
24      putBarragoonRandom(NewBoard, Board1),
25      copy_term(Board1, BoardOut)
26    ;
27      getColor(Piece, Color),
28      Color \= ' ',
29      changePlayer(CurrPlayer, NewPlayer),
30      (
31      CurrPlayer = 'W',
32      putBarragoonRandom(NewBoard, Board1),
33      write(CurrPlayer), write(' puts barragoon:'), nl,
34      putBarragoon(Board1, Board2),
35      copy_term(Board2, BoardOut)
36      ;
37      CurrPlayer = 'B',
38      write(NewPlayer), write(' puts barragoon:'), nl,
39      putBarragoon(NewBoard, Board1),
40      putBarragoonRandom(Board1, Board2),
41      copy_term(Board2, BoardOut)
42      )
43    ;
44      copy_term(NewBoard, BoardOut)
45    ).
46
47  /*
48  * Random CPU vs H game mode
49  */
50  playRandomCPUvsH:-
51    initialBoard(BoardIn),
52    initialPlayer(PlayerIn),
53    assert(board(BoardIn)),
54    assert(player(PlayerIn)),
55    repeat,
56          retract(board(BoardCurr)),
57          retract(player(PlayerCurr)),
58          once(askPlay(PlayerCurr,BoardCurr, NewBoard)),
59          once(changePlayer(PlayerCurr, NewPlayer)),
60          once(playCPUvsHrandom(NewPlayer, NewBoard, BoardOut)),
61          once(changePlayer(NewPlayer, PlayerCurr)),
62          assert(player(PlayerCurr)),
63          assert(board(BoardOut)),
64          gameOver(BoardIn, Loser),
65          showResult(Loser),
66     !.
67
68  /*
69  * AI CPU vs H game mode
70  */
71  playAICPUvsH:-
```

```
72    initialBoard(BoardIn),
73    initialPlayer(PlayerIn),
74    assert(board(BoardIn)),
75    assert(player(PlayerIn)),
76    repeat,
77          retract(board(BoardCurr)),
78          retract(player(PlayerCurr)),
79          once(askPlay(PlayerCurr,BoardCurr, NewBoard)),
80          once(changePlayer(PlayerCurr, NewPlayer)),
81          once(pcMove(NewPlayer, NewBoard, BoardOut)),
82          once(changePlayer(NewPlayer, PlayerCurr)),
83          assert(player(PlayerCurr)),
84          assert(board(BoardOut)),
85          gameOver(BoardIn, Loser),
86          showResult(Loser),
87     !.
```

HvsH.pl

```
1   /*
2   * H vs H game mode
3   */
4   playHvsH:-
5     initialBoard(BoardIn),
6     initialPlayer(PlayerIn),
7     assert(board(BoardIn)),
8     assert(player(PlayerIn)),
9     repeat,
10      retract(board(BoardCurr)),
11      retract(player(PlayerCurr)),
12      once(askPlay(PlayerCurr,BoardCurr, BoardOut)),
13      once(changePlayer(PlayerCurr, NewPlayer)),
14      assert(player(NewPlayer)),
15      assert(board(BoardOut)),
16      gameOver(BoardOut, Loser),
17      showResult(Loser),
18    !.
```

barragoon.pl

```
1   /**
2   * Place the Barragoon with the desired symbol in the desired
    ↪  position, if it is free
3   */
4   putBarragoon(InBoard, OutBoard):-
5           repeat,
6           nl, write('*** SELECT BARRAOON ***'), nl,
7           prompt(X, 'Barragoon: '),
8           read(Barragoon),
9           prompt('Barragoon: ', X),
10          barragoon(Barragoon),
11          nl,
```

```
12          write('*** SELECT MOVE ***'),        nl,
13          readInteger('Line: ', Nline), nl,
14          readInteger('Column: ', Ncolumn), nl,
15          getPiece(InBoard, Nline, Ncolumn, '  '),
16          setPiece(InBoard, Nline, Ncolumn, Barragoon, OutBoard).
17
18  /*
19  *        Auxiliar for findall of putBarragoonRandom
20  *        Finds free random places for barragoon
21  */
22  setBarragoon(BoardIn, Line, Column, BoardOut):-
23                  line(Line),
24                  col(Column),
25
                    ↪  random_permutation([no,or,ol,ot,ob,th,tv,rr,rl,rt,rb,lr,ll,lt,lb,at],
                    ↪  [Barragoon|_]),
26                  getPiece(BoardIn, Line, Column, '  '),
27                  setPiece(BoardIn, Line, Column, Barragoon,
                    ↪  BoardOut).
28
29  /*
30  *        Puts barragoon in a random place
31  */
32  putBarragoonRandom(BoardIn, BoardOut):-
33                  now(X),
34                  setrand(X),
35                  findall(XBoard, A^B^setBarragoon(BoardIn, A, B,
                    ↪  XBoard), Boards),
36                  random_permutation(Boards, [BoardOut|_]).
```

defs.pl

```
1   /**
2   * Barragoon list
3   */
4   barragoon(no).
5   barragoon(or).
6   barragoon(ol).
7   barragoon(ot).
8   barragoon(ob).
9   barragoon(th).
10  barragoon(tv).
11  barragoon(rr).
12  barragoon(rl).
13  barragoon(rt).
14  barragoon(rb).
15  barragoon(lr).
16  barragoon(ll).
17  barragoon(lt).
18  barragoon(lb).
19  barragoon(at).
20
```

```
/**
* Game lines
*/
line(1).
line(2).
line(3).
line(4).
line(5).
line(6).
line(7).
line(8).
line(9).

/**
* Game columns
*/
col(1).
col(2).
col(3).
col(4).
col(5).
col(6).
col(7).

/**
* Game play pieces
*/
piece('2W').
piece('3W').
piece('4W').
piece('2B').
piece('3B').
piece('4B').
```

game.pl

```
:- include('gamePrinting.pl').
:- include('pieceHandling.pl').
:- include('barragoon.pl').
:- include('piece2.pl').
:- include('piece3.pl').
:- include('piece4.pl').
:- include('pieceChecking.pl').
:- include('defs.pl').
:- include('menu.pl').
:- include('HvsH.pl').
:- include('HvsCPU.pl').
:- include('CPUvsCPU.pl').
:-include('pcAI.pl').
:- use_module(library(lists)).
:- use_module(library(random)).
:- use_module(library(system)).
```

```
17  :- dynamic board/1.
18  :- dynamic player/1.
19
20  /**
21  * Initial board
22  */
23  initialBoard([        ['  ','4W','3W','  ','3W','4W','  '],
24                  ['  ','  ','2W','3W','2W','  ','  '],
25                  ['  ','  ','  ','  ','  ','  ','  '],
26                  ['  ','no','  ','  ','  ','no','  '],
27                  ['no','  ','no','  ','no','  ','no'],
28                  ['  ','no','  ','  ','  ','no','  '],
29                  ['  ','  ','  ','  ','  ','  ','  '],
30                  ['  ','  ','2B','3B','2B','  ','  '],
31                  ['  ','4B','3B','  ','3B','4B','  ']]).
32
33  /**
34  * Initial player
35  */
36  initialPlayer('W').
37
38  /**
39  * Changes between players
40  */
41  changePlayer('W', 'B').
42  changePlayer('B', 'W').
43
44  /**
45  * Checks if there are pieces of a given color on board
46  */
47  noPiece(Board, Color) :-
48                  member(Line, Board),
49                  member(Piece, Line),
50                  name(Piece,[_|[Color|_]]),
51                  !, fail.
52  noPiece(_,_).
53
54  /**
55  * Game over condition checking
56  */
57  gameOver(Board, Loser):-
58                  (
59                     noPiece(Board, 66), Loser=66
60                  ;
61                     noPiece(Board, 87), Loser=87
62                  ).
63
64  /**
65  * Reads an integer number, with a given prompt
66  */
67  readInteger(Prompt,Integer):-
68                    repeat,
```

```
69                      prompt(X, Prompt),
70                      read(Integer),
71                      number(Integer),
72                  prompt(Prompt, X).
73
74  /**
75  * Read player move
76  */
77  readMove(PieceLine, PieceColumn, MoveLine, MoveColumn):-
78                  write('*** SELECT PIECE ***'),        nl,
79                  readInteger('Line: ', PieceLine), nl,
80                  readInteger('Column: ', PieceColumn), nl,
81                  nl,
82                  write('*** SELECT MOVE ***'),        nl,
83                  readInteger('Line: ', MoveLine), nl,
84                  readInteger('Column: ', MoveColumn), nl.
85
86  /**
87  * Move a piece from given coordinates to other given coordinates
88  */
89  movePiece(CurrPlayer, BoardIn, PieceLine, PieceColumn, MoveLine,
    ↪  MoveColumn, OutBoard):-
90                  line(PieceLine),
91                  col(PieceColumn),
92                  line(MoveLine),
93                  col(MoveColumn),
94                  getPiece(BoardIn, PieceLine, PieceColumn, Piece),
95                  piece(Piece),
96                  name(Piece,[_|[Color|_]]),
97                  name(CurrPlayer, [Color|_]),
98                  getNumber(Piece, Number),
99                  (
100                    Number=2, validateTwo(CurrPlayer, BoardIn,
                       ↪  PieceLine, PieceColumn, MoveLine,
                       ↪  MoveColumn)
101                 ;
102                    Number=3, validateThree(CurrPlayer, BoardIn,
                       ↪  PieceLine, PieceColumn, MoveLine,
                       ↪  MoveColumn)
103                 ;
104                    Number=4, validateFour(CurrPlayer, BoardIn,
                       ↪  PieceLine, PieceColumn, MoveLine,
                       ↪  MoveColumn)
105                 ),
106                 getPiece(BoardIn, PieceLine, PieceColumn, Piece),
107                 setPiece(BoardIn, PieceLine, PieceColumn, '  ',
                    ↪  NewBoard),
108                 setPiece(NewBoard, MoveLine, MoveColumn, Piece,
                    ↪  OutBoard).
109
110 /**
111 * Ask and validate human player play
```

```
112 */
113 askPlay(CurrPlayer, BoardIn, BoardOut):-
114                     findall(XBoard, A^B^C^D^movePiece(CurrPlayer,
                        ↪  BoardIn, A, B, C, D, XBoard),
                        ↪  PossiblePlays),
115                     (
116                        PossiblePlays \= []
117                     ;
118                        changePlayer(CurrPlayer, NewPlayer),
119                        gamePrint(BoardIn), nl,
120                        write(NewPlayer), write(' won!'),
121                        abort
122                     ),
123                     gamePrint(BoardIn),
124                 repeat,
125                 nl, write(CurrPlayer), write(' turn'), nl, nl,
126                 once(readMove(PieceLine, PieceColumn, MoveLine,
                    ↪  MoveColumn)),
127                 movePiece(CurrPlayer, BoardIn, PieceLine,
                    ↪  PieceColumn, MoveLine, MoveColumn, NewBoard),
128                 getPiece(BoardIn, MoveLine, MoveColumn, Piece),
129                 (
130                    gameOver(NewBoard, Loser),
131                    showResult(Loser),
132                    abort
133                 ;
134                    barragoon(Piece),
135                    write(CurrPlayer), write(' puts barragoon:'),
                       ↪  nl,
136                    copy_term(NewBoard, NewBoard),
137                    putBarragoon(NewBoard, Board1),
138                    copy_term(Board1, BoardOut)
139                 ;
140                    getColor(Piece, Color),
141                    Color \= ' ',
142                    changePlayer(CurrPlayer, NewPlayer),
143                    write(NewPlayer), write(' puts barragoon:'),
                       ↪  nl,
144                    putBarragoon(NewBoard, Board1),
145                    write(CurrPlayer), write(' puts barragoon:'),
                       ↪  nl,
146                    putBarragoon(Board1, Board2),
147                    copy_term(Board2, BoardOut)
148                 ;
149                    copy_term(NewBoard, BoardOut)
150                 ).
```

gamePrinting.pl

```
1 /**
2 *         Board printing function
3 */
```

```
gamePrint(Board) :-
        write('    1  2  3  4  5  6  7  '),        nl,
        printBlackLine,
        printRowByRow(1, Board).

/**
*        Prints divisory black line
*/
printBlackLine :-
        write('  +--+--+--+--+--+--+--+'),
        nl.

/**
*        Prints row by row
*/
printRowByRow(_, []).
printRowByRow(NoLine, [Line|Rest]) :-
        write(NoLine),
        write(' |'),
        printSingleRow(Line),
        NewNoLine is NoLine + 1,
        printRowByRow(NewNoLine, Rest).

/**
*        Prints single row
*/
printSingleRow([Cell]):-
        write(Cell),
        write('|'),
        nl,
        printBlackLine.
printSingleRow([Cell|More]):-
        write(Cell),
        write('|'),
        printSingleRow(More).

/**
*        Shows game Result
*/
showResult(Loser):-
        (
                Loser=66, name(X,[87]), write(X)
                ;
                Loser=87, name(X,[66]), write(X)
        ),
        write(' won! Congrats!'),        nl.
```

menu.pl

```
/**
*        Game level selection menu
*/
```

```
4  levelSelection(Level):-
5          write('*** SELECT DIFFICULTY ***'), nl,
6          write('1. Easier'), nl,
7          write('2. Harder'), nl,
8          prompt(X,'Select an option (0 to exit): '),
9          read(Level), nl,
10         prompt('Select an option (0 to exit): ', X).
11
12 /**
13 * Game menu
14 */
15 menu:-
16         write('*** BARRAGOON ***'), nl,
17         write('1. Human vs Human'), nl,
18         write('2. Human vs CPU'), nl,
19         write('3. CPU vs CPU'), nl,
20         write('Select an option (0 to exit):
           ↪  '),        read(Option), nl,
21         menuExe(Option).
22
23 /**
24 *         Executes menu option
25 */
26 menuExe(Option):-
27         (
28           Option=1, playHvsH
29           ;
30           Option=2, levelSelection(Level),
             ↪  levelSelectionExe(Option, Level)
31           ;
32           Option=3, levelSelection(Level),
             ↪  levelSelectionExe(Option, Level)
33           ;
34           Option=0, abort
35         ).
36
37 /*
38 *         Executes level selection menu option
39 */
40 levelSelectionExe(Option, Level):-
41         (
42           Option=2, Level=1, playRandomCPUvsH
43         ;
44           Option=3, Level=1, playRandomCPUvsCPU
45         ;
46           Option=2, Level=2, playAICPUvsH
47         ;
48           Option=3, Level=2, playAICPUvsCPU
49         ).
```

pcAI.pl

```
/*
* Makes PC AI move
*/
pcMove(CurrPlayer, BoardIn, BoardOut):-
  findall(XBoard-C-D, A^B^C^D^movePiece(CurrPlayer, BoardIn, A, B, C, D, XBoard), Boards),
  findall(XBoard-C-D, A^B^C^D^movePiece(CurrPlayer, BoardIn, A, B, C, D, XBoard), PossiblePlays),
        (
                PossiblePlays \= []
        ;
                changePlayer(CurrPlayer, NewPlayer),
                gamePrint(BoardIn), nl,
                write(NewPlayer), write(' won!'),
                abort
        ),
  gamePrint(BoardIn),
  nl, write(CurrPlayer), write(' turn'), nl,
  countBarragoons(BoardIn, BarragoonsIn),
  countPieces(BoardIn, PiecesIn),
  choosePlay(PossiblePlays, _Line, _Column, CurrPlayer, PiecesIn, BarragoonsIn, _Board, NewBoard, MoveLine, MoveColumn),
  (
  var(NewBoard),
        random_permutation(Boards, [NewBoard-MoveLine-MoveColumn|_])
  ;
  true
  ),
  getPiece(BoardIn, MoveLine, MoveColumn, Piece),
  (
    gameOver(NewBoard, Loser),
    gamePrint(NewBoard), nl,
    showResult(Loser),
    abort
  ;
    barragoon(Piece),
    putBarragoonRandom(NewBoard, Board1),
    copy_term(Board1, BoardOut)
  ;
    getColor(Piece, Color),
    Color \= ' ',
    putBarragoonRandom(NewBoard, Board1),
    putBarragoonRandom(Board1, Board2),
    copy_term(Board2, BoardOut)
  ;
    copy_term(NewBoard, BoardOut)
  ),
  get_code(_X).

```

```
47    choosePlay([],MoveLine, MoveColumn, _CurrPlayer, _PiecesOut,
      ↪  _BarragoonsOut, BoardOut, BoardOut, MoveLine, MoveColumn).
48
49    choosePlay([BoardIn-PlayLine-PlayColumn|Rest], MoveLine,
      ↪  MoveColumn, CurrPlayer, PiecesOut, BarragoonsOut, BoardOut,
      ↪  FinalBoard, LineOut, ColumnOut):-
50      countPieces(BoardIn, Pieces),
51      (
52      Pieces @< PiecesOut,
53      NewPiecesOut is Pieces,
54      NewMoveLine is PlayLine,
55      NewMoveColumn is PlayColumn,
56      choosePlay(Rest, NewMoveLine, NewMoveColumn, CurrPlayer,
        ↪  NewPiecesOut, BarragoonsOut, BoardIn, FinalBoard,
        ↪  LineOut, ColumnOut)
57      ;
58      Pieces = PiecesOut,
59      countBarragoons(BoardIn, Barragoons),
60      Barragoons @< BarragoonsOut,
61      NewBarragoonsOut is Barragoons,
62      NewMoveLine is PlayLine,
63      NewMoveColumn is PlayColumn,
64      choosePlay(Rest, NewMoveLine, NewMoveColumn, CurrPlayer,
        ↪  PiecesOut, NewBarragoonsOut, BoardIn, FinalBoard,
        ↪  LineOut, ColumnOut)
65      ;
66      choosePlay(Rest, MoveLine, MoveColumn, CurrPlayer, PiecesOut,
        ↪  BarragoonsOut, BoardOut, FinalBoard, LineOut, ColumnOut)
67      ).
68
69  /*
70  *  Counts barragoons in a single line
71  */
72  countBarragoonLine([], 0).
73  countBarragoonLine([Head | Tail], N) :-
74          (
75                  barragoon(Head) -> countBarragoonLine(Tail,
                    ↪  NextN), N is NextN + 1
76                  ;
77                  countBarragoonLine(Tail, N)
78          ).
79
80  /*
81  * Counts barragoons in board
82  */
83  countBarragoons([] , 0).
84  countBarragoons([Head | Tail], NTotal):-
85          countBarragoons(Tail, NextNTotal),
86          countBarragoonLine(Head, N),
87          NTotal is NextNTotal + N.
88
89  /*
```

```
90  * Counts number of pieces in a single line
91  */
92  countPiecesLine([], 0).
93  countPiecesLine([Head | Tail], N) :-
94  (
95          Head \= '   ' -> countPiecesLine(Tail, NextN), N is NextN
            ↪  + 1
96          ;
97          countPiecesLine(Tail, N)
98  ).
99
100 /*
101 * Counts number of pieces in board
102 */
103 countPieces([] , 0).
104 countPieces([Head | Tail], NTotal):-
105   countPieces(Tail, NextNTotal),
106   countPiecesLine(Head, N),
107   NTotal is NextNTotal + N.
```

piece2.pl

```
1   /**
2   * Validates all possible short moves for piece 2
3   */
4   shortMoveTwo(BoardIn, PieceLine, PieceColumn, MoveLine,
    ↪  MoveColumn):-
5     (
6       MoveLine =:= PieceLine + 1,
7       MoveColumn = PieceColumn,
8       checkEmpty(BoardIn, MoveLine, MoveColumn)
9     ;
10      MoveLine =:= PieceLine - 1,
11      MoveColumn = PieceColumn,
12      checkEmpty(BoardIn, MoveLine, MoveColumn)
13    ;
14      MoveColumn =:= PieceColumn - 1,
15      MoveLine = PieceLine,
16      checkEmpty(BoardIn, MoveLine, MoveColumn)
17    ;
18      MoveColumn =:= PieceColumn + 1,
19      MoveLine = PieceLine,
20      checkEmpty(BoardIn, MoveLine, MoveColumn)
21    ).
22
23  /**
24  * Validates all possible long moves for piece 2, starting by down
25  */
26  moveDownTwo(CurrPlayer, BoardIn, PieceLine, PieceColumn,
    ↪  MoveLine, MoveColumn):-
27    (
28      MoveLine =:= PieceLine + 2,
```

```
29        MoveColumn = PieceColumn,
30        checkOB(BoardIn, PieceLine + 1, PieceColumn),
31        checkCaptureAT(CurrPlayer, BoardIn, MoveLine, MoveColumn)
32    ;
33        MoveLine =:= PieceLine + 1,
34        MoveColumn =:= PieceColumn + 1,
35        checkLT(BoardIn, PieceLine + 1, PieceColumn),
36        checkCaptureAT(CurrPlayer, BoardIn, MoveLine, MoveColumn)
37    ;
38        MoveLine =:= PieceLine + 1,
39        MoveColumn =:= PieceColumn - 1,
40        checkRT(BoardIn, PieceLine + 1, PieceColumn),
41        checkCaptureAT(CurrPlayer, BoardIn, MoveLine, MoveColumn)
42    ).
43
44  /**
45  * Validates all possible long moves for piece 2, starting by up
46  */
47  moveUpTwo(CurrPlayer, BoardIn, PieceLine, PieceColumn, MoveLine,
     ↪  MoveColumn):-
48    (
49        MoveLine =:= PieceLine - 2,
50        MoveColumn = PieceColumn,
51        checkOT(BoardIn, PieceLine - 1, PieceColumn),
52        checkCaptureAT(CurrPlayer, BoardIn, MoveLine, MoveColumn)
53    ;
54        MoveLine =:= PieceLine - 1,
55        MoveColumn =:= PieceColumn - 1,
56        checkLB(BoardIn, PieceLine - 1, PieceColumn),
57        checkCaptureAT(CurrPlayer, BoardIn, MoveLine, MoveColumn)
58    ;
59        MoveLine =:= PieceLine - 1,
60        MoveColumn =:= PieceColumn + 1,
61        checkRB(BoardIn, PieceLine - 1, PieceColumn),
62        checkCaptureAT(CurrPlayer, BoardIn, MoveLine, MoveColumn)
63    ).
64
65  /**
66  * Validates all possible long moves for piece 2, starting by
     ↪  right
67  */
68  moveRightTwo(CurrPlayer, BoardIn, PieceLine, PieceColumn,
     ↪  MoveLine, MoveColumn):-
69    (
70        MoveColumn =:= PieceColumn + 2,
71        MoveLine = PieceLine,
72        checkOR(BoardIn, PieceLine, PieceColumn + 1),
73        checkCaptureAT(CurrPlayer, BoardIn, MoveLine, MoveColumn)
74    ;
75        MoveLine =:= PieceLine + 1,
76        MoveColumn =:= PieceColumn + 1,
77        checkRL(BoardIn, PieceLine, PieceColumn + 1),
```

```
78        checkCaptureAT(CurrPlayer, BoardIn, MoveLine, MoveColumn)
79    ;
80        MoveLine =:= PieceLine - 1,
81        MoveColumn =:= PieceColumn + 1,
82        checkLL(BoardIn, PieceLine, PieceColumn + 1),
83        checkCaptureAT(CurrPlayer, BoardIn, MoveLine, MoveColumn)
84    ).
85
86  /**
87  * Validates all possible long moves for piece 2, starting by left
88  */
89  moveLeftTwo(CurrPlayer, BoardIn, PieceLine, PieceColumn,
     ↪  MoveLine, MoveColumn):-
90    (
91        MoveColumn =:= PieceColumn - 2,
92        MoveLine = PieceLine,
93        checkOL(BoardIn, PieceLine, PieceColumn - 1),
94        checkCaptureAT(CurrPlayer, BoardIn, MoveLine, MoveColumn)
95    ;
96        MoveLine =:= PieceLine - 1,
97        MoveColumn =:= PieceColumn - 1,
98        checkRR(BoardIn, PieceLine, PieceColumn - 1),
99        checkCaptureAT(CurrPlayer, BoardIn, MoveLine, MoveColumn)
100   ;
101       MoveLine =:= PieceLine + 1,
102       MoveColumn =:= PieceColumn - 1,
103       checkLR(BoardIn, PieceLine, PieceColumn - 1),
104       checkCaptureAT(CurrPlayer, BoardIn, MoveLine, MoveColumn)
105   ).
106
107 /**
108 * Validates all possible moves for piece 2
109 */
110 validateTwo(CurrPlayer, BoardIn, PieceLine, PieceColumn,
     ↪  MoveLine, MoveColumn):-
111   (
112       moveDownTwo(CurrPlayer, BoardIn, PieceLine, PieceColumn,
          ↪  MoveLine, MoveColumn)
113   ;
114       moveUpTwo(CurrPlayer, BoardIn, PieceLine, PieceColumn,
          ↪  MoveLine, MoveColumn)
115   ;
116       moveRightTwo(CurrPlayer, BoardIn, PieceLine, PieceColumn,
          ↪  MoveLine, MoveColumn)
117   ;
118       moveLeftTwo(CurrPlayer, BoardIn, PieceLine, PieceColumn,
          ↪  MoveLine, MoveColumn)
119   ;
120       shortMoveTwo(BoardIn, PieceLine, PieceColumn, MoveLine,
          ↪  MoveColumn)
121   ).
```

piece3.pl

```
/**
* Validates all possible short moves for piece 3, starting by down
*/
shortMoveDownThree(BoardIn, PieceLine, PieceColumn, MoveLine, MoveColumn):-
    (
        MoveLine =:= PieceLine + 2,
        MoveColumn = PieceColumn,
        checkOB(BoardIn, PieceLine + 1, PieceColumn),
        checkEmpty(BoardIn, MoveLine, MoveColumn)
    ;
        MoveLine =:= PieceLine + 1,
        MoveColumn =:= PieceColumn + 1,
        checkLT(BoardIn, PieceLine + 1, PieceColumn),
        checkEmpty(BoardIn, MoveLine, MoveColumn)
    ;
        MoveLine =:= PieceLine + 1,
        MoveColumn =:= PieceColumn - 1,
        checkRT(BoardIn, PieceLine + 1, PieceColumn),
        checkEmpty(BoardIn, MoveLine, MoveColumn)
    ).

/**
* Validates all possible long moves for piece 3, starting by down
*/
moveDownThree(CurrPlayer, BoardIn, PieceLine, PieceColumn, MoveLine, MoveColumn):-
    (
        MoveLine =:= PieceLine + 3,
        MoveColumn = PieceColumn,
        checkOB(BoardIn, PieceLine + 1, PieceColumn),
        checkOB(BoardIn, PieceLine + 2, PieceColumn),
        checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
    ;
        MoveLine =:= PieceLine + 2,
        MoveColumn =:= PieceColumn + 1,
        checkOB(BoardIn, PieceLine + 1, PieceColumn),
        checkLT(BoardIn, PieceLine + 2, PieceColumn),
        checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
    ;
        MoveLine =:= PieceLine + 2,
        MoveColumn =:= PieceColumn - 1,
        checkOB(BoardIn, PieceLine + 1, PieceColumn),
        checkRT(BoardIn, PieceLine + 2, PieceColumn),
        checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
    ;
        MoveLine =:= PieceLine + 1,
        MoveColumn =:= PieceColumn - 2,
        checkRT(BoardIn, PieceLine + 1, PieceColumn),
```

```
48          checkOL(BoardIn, PieceLine + 1, PieceColumn - 1),
49          checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
50      ;
51          MoveLine =:= PieceLine + 1,
52          MoveColumn =:= PieceColumn + 2,
53          checkLT(BoardIn, PieceLine + 1, PieceColumn),
54          checkOR(BoardIn, PieceLine + 1, PieceColumn + 1),
55          checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
56      ).
57
58  /**
59  * Validates all possible short moves for piece 3, starting by up
60  */
61  shortMoveUpThree(BoardIn, PieceLine, PieceColumn, MoveLine,
    ↪  MoveColumn):-
62      (
63          MoveLine =:= PieceLine - 2,
64          MoveColumn = PieceColumn,
65          checkOT(BoardIn, PieceLine - 1, PieceColumn),
66          checkEmpty(BoardIn, MoveLine, MoveColumn)
67      ;
68          MoveLine =:= PieceLine - 1,
69          MoveColumn =:= PieceColumn + 1,
70          checkRB(BoardIn, PieceLine - 1, PieceColumn),
71          checkEmpty(BoardIn, MoveLine, MoveColumn)
72      ;
73          MoveLine =:= PieceLine - 1,
74          MoveColumn =:= PieceColumn - 1,
75          checkLB(BoardIn, PieceLine - 1, PieceColumn),
76          checkEmpty(BoardIn, MoveLine, MoveColumn)
77      ).
78
79  /**
80  * Validates all possible long moves for piece 3, starting by up
81  */
82  moveUpThree(CurrPlayer, BoardIn, PieceLine, PieceColumn,
    ↪  MoveLine, MoveColumn):-
83      (
84          MoveLine =:= PieceLine - 3,
85          MoveColumn = PieceColumn,
86          checkOT(BoardIn, PieceLine - 1, PieceColumn),
87          checkOT(BoardIn, PieceLine - 2, PieceColumn),
88          checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
89      ;
90          MoveLine =:= PieceLine - 2,
91          MoveColumn =:= PieceColumn + 1,
92          checkOT(BoardIn, PieceLine - 1, PieceColumn),
93          checkRB(BoardIn, PieceLine - 2, PieceColumn),
94          checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
95      ;
96          MoveLine =:= PieceLine - 2,
97          MoveColumn =:= PieceColumn - 1,
```

```
98          checkOT(BoardIn, PieceLine - 1, PieceColumn),
99          checkLB(BoardIn, PieceLine - 2, PieceColumn),
100         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
101     ;
102         MoveLine =:= PieceLine - 1,
103         MoveColumn =:= PieceColumn - 2,
104         checkLB(BoardIn, PieceLine - 1, PieceColumn),
105         checkOL(BoardIn, PieceLine - 1, PieceColumn - 1),
106         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
107     ;
108         MoveLine =:= PieceLine - 1,
109         MoveColumn =:= PieceColumn + 2,
110         checkRB(BoardIn, PieceLine - 1, PieceColumn),
111         checkOR(BoardIn, PieceLine - 1, PieceColumn + 1),
112         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
113     ).
114
115 /**
116 * Validates all possible short moves for piece 3, starting by
    ↪ right
117 */
118 shortMoveRightThree(BoardIn, PieceLine, PieceColumn, MoveLine,
    ↪ MoveColumn):-
119     (
120         MoveColumn =:= PieceColumn + 2,
121         MoveLine = PieceLine,
122         checkOR(BoardIn, PieceLine, PieceColumn + 1),
123         checkEmpty(BoardIn, MoveLine, MoveColumn)
124     ;
125         MoveLine =:= PieceLine + 1,
126         MoveColumn =:= PieceColumn + 1,
127         checkRL(BoardIn, PieceLine, PieceColumn + 1),
128         checkEmpty(BoardIn, MoveLine, MoveColumn)
129     ;
130         MoveLine =:= PieceLine - 1,
131         MoveColumn =:= PieceColumn + 1,
132         checkLL(BoardIn, PieceLine, PieceColumn + 1),
133         checkEmpty(BoardIn, MoveLine, MoveColumn)
134     ).
135
136 /**
137 * Validates all possible long moves for piece 3, starting by
    ↪ right
138 */
139 moveRightThree(CurrPlayer, BoardIn, PieceLine, PieceColumn,
    ↪ MoveLine, MoveColumn):-
140     (
141         MoveColumn =:= PieceColumn + 3,
142         MoveLine = PieceLine,
143         checkOR(BoardIn, PieceLine, PieceColumn + 1),
144         checkOR(BoardIn, PieceLine, PieceColumn + 2),
145         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
```

```
146         ;
147             MoveLine =:= PieceLine - 1,
148             MoveColumn =:= PieceColumn + 2,
149             checkOR(BoardIn, PieceLine, PieceColumn + 1),
150             checkLL(BoardIn, PieceLine, PieceColumn + 2),
151             checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
152         ;
153             MoveLine =:= PieceLine + 1,
154             MoveColumn =:= PieceColumn + 2,
155             checkOR(BoardIn, PieceLine, PieceColumn + 1),
156             checkRL(BoardIn, PieceLine, PieceColumn + 2),
157             checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
158         ;
159             MoveLine =:= PieceLine - 2,
160             MoveColumn =:= PieceColumn + 1,
161             checkLL(BoardIn, PieceLine, PieceColumn + 1),
162             checkOT(BoardIn, PieceLine - 1, PieceColumn + 1),
163             checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
164         ;
165             MoveLine =:= PieceLine + 2,
166             MoveColumn =:= PieceColumn + 1,
167             checkRL(BoardIn, PieceLine, PieceColumn + 1),
168             checkOB(BoardIn, PieceLine + 1, PieceColumn + 1),
169             checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
170         ).
171
172     /**
173     * Validates all possible short moves for piece 3, starting by
        ↪  left
174     */
175     shortMoveLeftThree(BoardIn, PieceLine, PieceColumn, MoveLine,
        ↪  MoveColumn):-
176         (
177             MoveColumn =:= PieceColumn - 2,
178             MoveLine = PieceLine,
179             checkOL(BoardIn, PieceLine, PieceColumn - 1),
180             checkEmpty(BoardIn, MoveLine, MoveColumn)
181         ;
182             MoveLine =:= PieceLine - 1,
183             MoveColumn =:= PieceColumn - 1,
184             checkRR(BoardIn, PieceLine, PieceColumn - 1),
185             checkEmpty(BoardIn, MoveLine, MoveColumn)
186         ;
187             MoveLine =:= PieceLine + 1,
188             MoveColumn =:= PieceColumn - 1,
189             checkLR(BoardIn, PieceLine, PieceColumn - 1),
190             checkEmpty(BoardIn, MoveLine, MoveColumn)
191         ).
192
193     /**
194     * Validates all possible long moves for piece 3, starting by left
195     */
```

```
196 moveLeftThree(CurrPlayer, BoardIn, PieceLine, PieceColumn,
    ↪ MoveLine, MoveColumn):-
197     (
198         MoveColumn =:= PieceColumn - 3,
199         MoveLine = PieceLine,
200         checkOL(BoardIn, PieceLine, PieceColumn - 1),
201         checkOL(BoardIn, PieceLine, PieceColumn - 2),
202         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
203     ;
204         MoveLine =:= PieceLine - 1,
205         MoveColumn =:= PieceColumn - 2,
206         checkOL(BoardIn, PieceLine, PieceColumn - 1),
207         checkRR(BoardIn, PieceLine, PieceColumn - 2),
208         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
209     ;
210         MoveLine =:= PieceLine + 1,
211         MoveColumn =:= PieceColumn - 2,
212         checkOL(BoardIn, PieceLine, PieceColumn - 1),
213         checkLR(BoardIn, PieceLine, PieceColumn - 2),
214         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
215     ;
216         MoveLine =:= PieceLine - 2,
217         MoveColumn =:= PieceColumn - 1,
218         checkRR(BoardIn, PieceLine, PieceColumn - 1),
219         checkOT(BoardIn, PieceLine - 1, PieceColumn - 1),
220         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
221     ;
222         MoveLine =:= PieceLine + 2,
223         MoveColumn =:= PieceColumn - 1,
224         checkLR(BoardIn, PieceLine, PieceColumn - 1),
225         checkOB(BoardIn, PieceLine + 1, PieceColumn - 1),
226         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
227     ).
228
229 /**
230 * Validates all possible moves for piece 3
231 */
232 validateThree(CurrPlayer, BoardIn, PieceLine, PieceColumn,
    ↪ MoveLine, MoveColumn):-
233     (
234         moveDownThree(CurrPlayer, BoardIn, PieceLine, PieceColumn,
            ↪ MoveLine, MoveColumn)
235     ;
236         moveUpThree(CurrPlayer, BoardIn, PieceLine, PieceColumn,
            ↪ MoveLine, MoveColumn)
237     ;
238         moveRightThree(CurrPlayer, BoardIn, PieceLine,
            ↪ PieceColumn, MoveLine, MoveColumn)
239     ;
240         moveLeftThree(CurrPlayer, BoardIn, PieceLine, PieceColumn,
            ↪ MoveLine, MoveColumn)
241     ;
```

```
242         shortMoveUpThree(BoardIn, PieceLine, PieceColumn,
            ↪ MoveLine, MoveColumn)
243     ;
244         shortMoveDownThree(BoardIn, PieceLine, PieceColumn,
            ↪ MoveLine, MoveColumn)
245     ;
246         shortMoveLeftThree(BoardIn, PieceLine, PieceColumn,
            ↪ MoveLine, MoveColumn)
247     ;
248         shortMoveRightThree(BoardIn, PieceLine, PieceColumn,
            ↪ MoveLine, MoveColumn)
249     ).
```

piece4.pl

```
1   /**
2   * Validates all possible short moves for piece 3, starting by
    ↪ down
3   */
4   shortMoveDownFour(BoardIn, PieceLine, PieceColumn, MoveLine,
    ↪ MoveColumn):-
5     (
6       MoveLine =:= PieceLine + 3,
7       MoveColumn = PieceColumn,
8       checkOB(BoardIn, PieceLine + 1, PieceColumn),
9       checkOB(BoardIn, PieceLine + 2, PieceColumn),
10      checkEmpty(BoardIn, MoveLine, MoveColumn)
11    ;
12      MoveLine =:= PieceLine + 2,
13      MoveColumn =:= PieceColumn + 1,
14      checkOB(BoardIn, PieceLine + 1, PieceColumn),
15      checkLT(BoardIn, PieceLine + 2, PieceColumn),
16      checkEmpty(BoardIn, MoveLine, MoveColumn)
17    ;
18      MoveLine =:= PieceLine + 2,
19      MoveColumn =:= PieceColumn - 1,
20      checkOB(BoardIn, PieceLine + 1, PieceColumn),
21      checkRT(BoardIn, PieceLine + 2, PieceColumn),
22      checkEmpty(BoardIn, MoveLine, MoveColumn)
23    ;
24      MoveLine =:= PieceLine + 1,
25      MoveColumn =:= PieceColumn - 2,
26      checkRT(BoardIn, PieceLine + 1, PieceColumn),
27      checkOL(BoardIn, PieceLine + 1, PieceColumn - 1),
28      checkEmpty(BoardIn, MoveLine, MoveColumn)
29    ;
30      MoveLine =:= PieceLine + 1,
31      MoveColumn =:= PieceColumn + 2,
32      checkLT(BoardIn, PieceLine + 1, PieceColumn),
33      checkOR(BoardIn, PieceLine + 1, PieceColumn + 1),
34      checkEmpty(BoardIn, MoveLine, MoveColumn)
35    ).
```

```
36
37  /**
38  * Validates all possible long moves for piece 4, starting by down
39  */
40  moveDownFour(CurrPlayer, BoardIn, PieceLine, PieceColumn,
    ↪  MoveLine, MoveColumn):-
41    (
42       MoveLine =:= PieceLine + 4,
43       MoveColumn = PieceColumn,
44       checkOB(BoardIn, PieceLine + 1, PieceColumn),
45       checkOB(BoardIn, PieceLine + 2, PieceColumn),
46       checkOB(BoardIn, PieceLine + 3, PieceColumn),
47       checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
48    ;
49       MoveLine =:= PieceLine + 3,
50       MoveColumn =:= PieceColumn + 1,
51       checkOB(BoardIn, PieceLine + 1, PieceColumn),
52       checkOB(BoardIn, PieceLine + 2, PieceColumn),
53       checkLT(BoardIn, PieceLine + 3, PieceColumn),
54       checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
55    ;
56       MoveLine =:= PieceLine + 3,
57       MoveColumn =:= PieceColumn - 1,
58       checkOB(BoardIn, PieceLine + 1, PieceColumn),
59       checkOB(BoardIn, PieceLine + 2, PieceColumn),
60       checkRT(BoardIn, PieceLine + 3, PieceColumn),
61       checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
62    ;
63       MoveLine =:= PieceLine + 2,
64       MoveColumn =:= PieceColumn - 2,
65       checkOB(BoardIn, PieceLine + 1, PieceColumn),
66       checkRT(BoardIn, PieceLine + 2, PieceColumn),
67       checkOL(BoardIn, PieceLine + 2, PieceColumn - 1),
68       checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
69    ;
70       MoveLine =:= PieceLine + 2,
71       MoveColumn =:= PieceColumn + 2,
72       checkOB(BoardIn, PieceLine + 1, PieceColumn),
73       checkLT(BoardIn, PieceLine + 2, PieceColumn),
74       checkOR(BoardIn, PieceLine + 2, PieceColumn + 1),
75       checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
76    ;
77       MoveLine =:= PieceLine + 1,
78       MoveColumn =:= PieceColumn + 3,
79       checkLT(BoardIn, PieceLine + 1, PieceColumn),
80       checkOR(BoardIn, PieceLine + 1, PieceColumn + 1),
81       checkOR(BoardIn, PieceLine + 1, PieceColumn + 2),
82       checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
83    ;
84       MoveLine =:= PieceLine + 1,
85       MoveColumn =:= PieceColumn - 3,
86       checkLT(BoardIn, PieceLine + 1, PieceColumn),
```

```
87        checkOL(BoardIn, PieceLine + 1, PieceColumn - 1),
88        checkOL(BoardIn, PieceLine + 1, PieceColumn - 2),
89        checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
90    ).
91
92  /**
93  * Validates all possible short moves for piece 4, starting by up
94  */
95  shortMoveUpFour(BoardIn, PieceLine, PieceColumn, MoveLine,
    ↪  MoveColumn):-
96    (
97        MoveLine =:= PieceLine - 3,
98        MoveColumn = PieceColumn,
99        checkOT(BoardIn, PieceLine - 1, PieceColumn),
100       checkOT(BoardIn, PieceLine - 2, PieceColumn),
101       checkEmpty(BoardIn, MoveLine, MoveColumn)
102   ;
103       MoveLine =:= PieceLine - 2,
104       MoveColumn =:= PieceColumn + 1,
105       checkOT(BoardIn, PieceLine - 1, PieceColumn),
106       checkRB(BoardIn, PieceLine - 2, PieceColumn),
107       checkEmpty(BoardIn, MoveLine, MoveColumn)
108   ;
109       MoveLine =:= PieceLine - 2,
110       MoveColumn =:= PieceColumn - 1,
111       checkOT(BoardIn, PieceLine - 1, PieceColumn),
112       checkLB(BoardIn, PieceLine - 2, PieceColumn),
113       checkEmpty(BoardIn, MoveLine, MoveColumn)
114   ;
115       MoveLine =:= PieceLine - 1,
116       MoveColumn =:= PieceColumn - 2,
117       checkLB(BoardIn, PieceLine - 1, PieceColumn),
118       checkOL(BoardIn, PieceLine - 1, PieceColumn - 1),
119       checkEmpty(BoardIn, MoveLine, MoveColumn)
120   ;
121       MoveLine =:= PieceLine - 1,
122       MoveColumn =:= PieceColumn + 2,
123       checkRB(BoardIn, PieceLine - 1, PieceColumn),
124       checkOR(BoardIn, PieceLine - 1, PieceColumn + 1),
125       checkEmpty(BoardIn, MoveLine, MoveColumn)
126   ).
127
128 /**
129 * Validates all possible long moves for piece 4, starting by up
130 */
131 moveUpFour(CurrPlayer, BoardIn, PieceLine, PieceColumn, MoveLine,
    ↪  MoveColumn):-
132   (
133       MoveLine =:= PieceLine - 4,
134       MoveColumn = PieceColumn,
135       checkOT(BoardIn, PieceLine - 1, PieceColumn),
136       checkOT(BoardIn, PieceLine - 2, PieceColumn),
```

```
137         checkOT(BoardIn, PieceLine - 3, PieceColumn),
138         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
139     ;
140         MoveLine =:= PieceLine - 3,
141         MoveColumn =:= PieceColumn + 1,
142         checkOT(BoardIn, PieceLine - 1, PieceColumn),
143         checkOT(BoardIn, PieceLine - 2, PieceColumn),
144         checkRB(BoardIn, PieceLine - 3, PieceColumn),
145         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
146     ;
147         MoveLine =:= PieceLine - 3,
148         MoveColumn =:= PieceColumn - 1,
149         checkOT(BoardIn, PieceLine - 1, PieceColumn),
150         checkOT(BoardIn, PieceLine - 2, PieceColumn),
151         checkLB(BoardIn, PieceLine - 3, PieceColumn),
152         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
153     ;
154         MoveLine =:= PieceLine - 2,
155         MoveColumn =:= PieceColumn - 2,
156         checkOT(BoardIn, PieceLine - 1, PieceColumn),
157         checkLB(BoardIn, PieceLine - 2, PieceColumn),
158         checkOL(BoardIn, PieceLine - 2, PieceColumn - 1),
159         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
160     ;
161         MoveLine =:= PieceLine - 2,
162         MoveColumn =:= PieceColumn + 2,
163         checkOT(BoardIn, PieceLine - 1, PieceColumn),
164         checkRB(BoardIn, PieceLine - 2, PieceColumn),
165         checkOR(BoardIn, PieceLine - 2, PieceColumn + 1),
166         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
167     ;
168         MoveLine =:= PieceLine - 1,
169         MoveColumn =:= PieceColumn + 3,
170         checkRB(BoardIn, PieceLine - 1, PieceColumn),
171         checkOR(BoardIn, PieceLine - 1, PieceColumn + 1),
172         checkOR(BoardIn, PieceLine - 1, PieceColumn + 2),
173         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
174     ;
175         MoveLine =:= PieceLine - 1,
176         MoveColumn =:= PieceColumn - 3,
177         checkLB(BoardIn, PieceLine - 1, PieceColumn),
178         checkOL(BoardIn, PieceLine - 1, PieceColumn - 1),
179         checkOL(BoardIn, PieceLine - 1, PieceColumn - 2),
180         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
181     ).
182
183 /**
184 * Validates all possible short moves for piece 4, starting by
    ↪  right
185 */
186 shortMoveRightFour(BoardIn, PieceLine, PieceColumn, MoveLine,
    ↪  MoveColumn):-
```

```
187     (
188         MoveColumn =:= PieceColumn + 3,
189         MoveLine = PieceLine,
190         checkOR(BoardIn, PieceLine, PieceColumn + 1),
191         checkOR(BoardIn, PieceLine, PieceColumn + 2),
192         checkEmpty(BoardIn, MoveLine, MoveColumn)
193     ;
194         MoveLine =:= PieceLine - 1,
195         MoveColumn =:= PieceColumn + 2,
196         checkOR(BoardIn, PieceLine, PieceColumn + 1),
197         checkLL(BoardIn, PieceLine, PieceColumn + 2),
198         checkEmpty(BoardIn, MoveLine, MoveColumn)
199     ;
200         MoveLine =:= PieceLine + 1,
201         MoveColumn =:= PieceColumn + 2,
202         checkOR(BoardIn, PieceLine, PieceColumn + 1),
203         checkRL(BoardIn, PieceLine, PieceColumn + 2),
204         checkEmpty(BoardIn, MoveLine, MoveColumn)
205     ;
206         MoveLine =:= PieceLine - 2,
207         MoveColumn =:= PieceColumn + 1,
208         checkLL(BoardIn, PieceLine, PieceColumn + 1),
209         checkOT(BoardIn, PieceLine - 1, PieceColumn + 1),
210         checkEmpty(BoardIn, MoveLine, MoveColumn)
211     ;
212         MoveLine =:= PieceLine + 2,
213         MoveColumn =:= PieceColumn + 1,
214         checkRL(BoardIn, PieceLine, PieceColumn + 1),
215         checkOB(BoardIn, PieceLine + 1, PieceColumn + 1),
216         checkEmpty(BoardIn, MoveLine, MoveColumn)
217     ).
218
219 /**
220 * Validates all possible long moves for piece 4, starting by
    ↪  right
221 */
222 moveRightFour(CurrPlayer, BoardIn, PieceLine, PieceColumn,
    ↪  MoveLine, MoveColumn):-
223     (
224         MoveColumn =:= PieceColumn + 4,
225         MoveLine = PieceLine,
226         checkOR(BoardIn, PieceLine, PieceColumn + 1),
227         checkOR(BoardIn, PieceLine, PieceColumn + 2),
228         checkOR(BoardIn, PieceLine, PieceColumn + 3),
229         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
230     ;
231         MoveLine =:= PieceLine - 1,
232         MoveColumn =:= PieceColumn + 3,
233         checkOR(BoardIn, PieceLine, PieceColumn + 1),
234         checkOR(BoardIn, PieceLine, PieceColumn + 2),
235         checkLL(BoardIn, PieceLine, PieceColumn + 3),
236         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
```

```
237     ;
238         MoveLine =:= PieceLine + 1,
239         MoveColumn =:= PieceColumn + 3,
240         checkOR(BoardIn, PieceLine, PieceColumn + 1),
241         checkOR(BoardIn, PieceLine, PieceColumn + 2),
242         checkRL(BoardIn, PieceLine, PieceColumn + 3),
243         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
244     ;
245         MoveLine =:= PieceLine - 2,
246         MoveColumn =:= PieceColumn + 2,
247         checkOR(BoardIn, PieceLine, PieceColumn + 1),
248         checkLL(BoardIn, PieceLine, PieceColumn + 2),
249         checkOT(BoardIn, PieceLine - 1, PieceColumn + 2),
250         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
251     ;
252         MoveLine =:= PieceLine + 2,
253         MoveColumn =:= PieceColumn + 2,
254         checkOR(BoardIn, PieceLine, PieceColumn + 1),
255         checkRL(BoardIn, PieceLine, PieceColumn + 2),
256         checkOB(BoardIn, PieceLine + 1, PieceColumn + 2),
257         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
258     ;
259         MoveLine =:= PieceLine + 3,
260         MoveColumn =:= PieceColumn + 1,
261         checkRL(BoardIn, PieceLine, PieceColumn + 1),
262         checkOB(BoardIn, PieceLine + 1, PieceColumn + 1),
263         checkOB(BoardIn, PieceLine + 2, PieceColumn + 1),
264         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
265     ;
266         MoveLine =:= PieceLine - 3,
267         MoveColumn =:= PieceColumn + 1,
268         checkLL(BoardIn, PieceLine, PieceColumn + 1),
269         checkOT(BoardIn, PieceLine - 1, PieceColumn + 1),
270         checkOT(BoardIn, PieceLine - 2, PieceColumn + 1),
271         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
272     ).
273 
274 /**
275 * Validates all possible short moves for piece 4, starting by
    ↪  left
276 */
277 shortMoveLeftFour(BoardIn, PieceLine, PieceColumn, MoveLine,
    ↪  MoveColumn):-
278     (
279         MoveColumn =:= PieceColumn - 3,
280         MoveLine = PieceLine,
281         checkOL(BoardIn, PieceLine, PieceColumn - 1),
282         checkOL(BoardIn, PieceLine, PieceColumn - 2),
283         checkEmpty(BoardIn, MoveLine, MoveColumn)
284     ;
285         MoveLine =:= PieceLine - 1,
286         MoveColumn =:= PieceColumn - 2,
```

```
287         checkOL(BoardIn, PieceLine, PieceColumn - 1),
288         checkRR(BoardIn, PieceLine, PieceColumn - 2),
289         checkEmpty(BoardIn, MoveLine, MoveColumn)
290     ;
291         MoveLine =:= PieceLine + 1,
292         MoveColumn =:= PieceColumn - 2,
293         checkOL(BoardIn, PieceLine, PieceColumn - 1),
294         checkLR(BoardIn, PieceLine, PieceColumn - 2),
295         checkEmpty(BoardIn, MoveLine, MoveColumn)
296     ;
297         MoveLine =:= PieceLine - 2,
298         MoveColumn =:= PieceColumn - 1,
299         checkRR(BoardIn, PieceLine, PieceColumn - 1),
300         checkOT(BoardIn, PieceLine - 1, PieceColumn - 1),
301         checkEmpty(BoardIn, MoveLine, MoveColumn)
302     ;
303         MoveLine =:= PieceLine + 2,
304         MoveColumn =:= PieceColumn - 1,
305         checkLR(BoardIn, PieceLine, PieceColumn - 1),
306         checkOB(BoardIn, PieceLine + 1, PieceColumn - 1),
307         checkEmpty(BoardIn, MoveLine, MoveColumn)
308     ).
309
310 /**
311 * Validates all possible long moves for piece 4, starting by left
312 */
313 moveLeftFour(CurrPlayer, BoardIn, PieceLine, PieceColumn,
    ↪  MoveLine, MoveColumn):-
314     (
315         MoveColumn =:= PieceColumn - 4,
316         MoveLine = PieceLine,
317         checkOL(BoardIn, PieceLine, PieceColumn - 1),
318         checkOL(BoardIn, PieceLine, PieceColumn - 2),
319         checkOL(BoardIn, PieceLine, PieceColumn - 3),
320         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
321     ;
322         MoveLine =:= PieceLine - 1,
323         MoveColumn =:= PieceColumn + 3,
324         checkOL(BoardIn, PieceLine, PieceColumn - 1),
325         checkOL(BoardIn, PieceLine, PieceColumn - 2),
326         checkRR(BoardIn, PieceLine, PieceColumn - 3),
327         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
328     ;
329         MoveLine =:= PieceLine + 1,
330         MoveColumn =:= PieceColumn - 3,
331         checkOL(BoardIn, PieceLine, PieceColumn - 1),
332         checkOL(BoardIn, PieceLine, PieceColumn - 2),
333         checkLR(BoardIn, PieceLine, PieceColumn - 3),
334         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
335     ;
336         MoveLine =:= PieceLine - 2,
337         MoveColumn =:= PieceColumn - 2,
```

```
338         checkOL(BoardIn, PieceLine, PieceColumn - 1),
339         checkRR(BoardIn, PieceLine, PieceColumn - 2),
340         checkOT(BoardIn, PieceLine - 1, PieceColumn - 2),
341         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
342     ;
343         MoveLine =:= PieceLine + 2,
344         MoveColumn =:= PieceColumn - 2,
345         checkOL(BoardIn, PieceLine, PieceColumn - 1),
346         checkLR(BoardIn, PieceLine, PieceColumn - 2),
347         checkOB(BoardIn, PieceLine + 1, PieceColumn - 2),
348         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
349     ;
350         MoveLine =:= PieceLine + 3,
351         MoveColumn =:= PieceColumn - 1,
352         checkLR(BoardIn, PieceLine, PieceColumn - 1),
353         checkOB(BoardIn, PieceLine + 1, PieceColumn - 1),
354         checkOT(BoardIn, PieceLine + 2, PieceColumn - 1),
355         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
356     ;
357         MoveLine =:= PieceLine - 3,
358         MoveColumn =:= PieceColumn - 1,
359         checkRR(BoardIn, PieceLine, PieceColumn - 1),
360         checkOT(BoardIn, PieceLine - 1, PieceColumn - 1),
361         checkOT(BoardIn, PieceLine - 2, PieceColumn - 1),
362         checkCapture(CurrPlayer, BoardIn, MoveLine, MoveColumn)
363     ).
364
365 /**
366 * Validates all possible moves for piece 4
367 */
368 validateFour(CurrPlayer, BoardIn, PieceLine, PieceColumn,
    ↪ MoveLine, MoveColumn):-
369     (
370         moveDownFour(CurrPlayer, BoardIn, PieceLine, PieceColumn,
            ↪ MoveLine, MoveColumn)
371     ;
372         moveUpFour(CurrPlayer, BoardIn, PieceLine, PieceColumn,
            ↪ MoveLine, MoveColumn)
373     ;
374         moveRightFour(CurrPlayer, BoardIn, PieceLine, PieceColumn,
            ↪ MoveLine, MoveColumn)
375     ;
376         moveLeftFour(CurrPlayer, BoardIn, PieceLine, PieceColumn,
            ↪ MoveLine, MoveColumn)
377     ;
378         shortMoveUpFour(BoardIn, PieceLine, PieceColumn, MoveLine,
            ↪ MoveColumn)
379     ;
380         shortMoveDownFour(BoardIn, PieceLine, PieceColumn, MoveLine,
            ↪ MoveColumn)
381     ;
```

```
382       shortMoveLeftFour(BoardIn, PieceLine, PieceColumn, MoveLine,
          ↪ MoveColumn)
383     ;
384       shortMoveRightFour(BoardIn, PieceLine, PieceColumn,
          ↪ MoveLine, MoveColumn)
385     ).
```

pieceChecking.pl

```
1   /**
2   * Check if a place is empty
3   */
4   checkEmpty(BoardIn, PieceLine, PieceColumn):-
5     getPiece(BoardIn, PieceLine, PieceColumn, '  ').
6
7   /**
8   * Check if it is possible to make a OB move in a given place
9   */
10  checkOB(BoardIn, PieceLine, PieceColumn):-
11    (
12      getPiece(BoardIn, PieceLine, PieceColumn, '  ')
13      ;
14      getPiece(BoardIn, PieceLine, PieceColumn, 'ob')
15      ;
16      getPiece(BoardIn, PieceLine, PieceColumn, 'tv')
17    ).
18
19  /**
20  * Check if it is possible to make a LT move in a given place
21  */
22  checkLT(BoardIn, PieceLine, PieceColumn):-
23    (
24      getPiece(BoardIn, PieceLine, PieceColumn, '  ')
25      ;
26      getPiece(BoardIn, PieceLine, PieceColumn, 'lt')
27      ;
28      getPiece(BoardIn, PieceLine, PieceColumn, 'at')
29    ).
30
31  /**
32  * Check if it is possible to make a RT move in a given place
33  */
34  checkRT(BoardIn, PieceLine, PieceColumn):-
35    (
36      getPiece(BoardIn, PieceLine, PieceColumn, '  ')
37      ;
38      getPiece(BoardIn, PieceLine, PieceColumn, 'rt')
39      ;
40      getPiece(BoardIn, PieceLine, PieceColumn, 'at')
41    ).
42
43  /**
```

```
44  * Check if it is possible to make a OT move in a given place
45  */
46  checkOT(BoardIn, PieceLine, PieceColumn):-
47    (
48      getPiece(BoardIn, PieceLine, PieceColumn, '  ')
49      ;
50      getPiece(BoardIn, PieceLine, PieceColumn, 'ot')
51      ;
52      getPiece(BoardIn, PieceLine, PieceColumn, 'tv')
53    ).
54
55  /**
56  * Check if it is possible to make a LB move in a given place
57  */
58  checkLB(BoardIn, PieceLine, PieceColumn):-
59    (
60      getPiece(BoardIn, PieceLine, PieceColumn, '  ')
61      ;
62      getPiece(BoardIn, PieceLine, PieceColumn, 'lb')
63      ;
64      getPiece(BoardIn, PieceLine, PieceColumn, 'at')
65    ).
66
67  /**
68  * Check if it is possible to make a RB move in a given place
69  */
70  checkRB(BoardIn, PieceLine, PieceColumn):-
71    (
72      getPiece(BoardIn, PieceLine, PieceColumn, '  ')
73      ;
74      getPiece(BoardIn, PieceLine, PieceColumn, 'rb')
75      ;
76      getPiece(BoardIn, PieceLine, PieceColumn, 'at')
77    ).
78
79  /**
80  * Check if it is possible to make a OR move in a given place
81  */
82  checkOR(BoardIn, PieceLine, PieceColumn):-
83    (
84      getPiece(BoardIn, PieceLine, PieceColumn, '  ')
85      ;
86      getPiece(BoardIn, PieceLine, PieceColumn, 'or')
87      ;
88      getPiece(BoardIn, PieceLine, PieceColumn, 'th')
89    ).
90
91  /**
92  * Check if it is possible to make a RL move in a given place
93  */
94  checkRL(BoardIn, PieceLine, PieceColumn):-
95    (
```

```
96      getPiece(BoardIn, PieceLine, PieceColumn, '  ')
97      ;
98      getPiece(BoardIn, PieceLine, PieceColumn, 'rl')
99      ;
100     getPiece(BoardIn, PieceLine, PieceColumn, 'at')
101   ).
102
103 /**
104 * Check if it is possible to make a LL move in a given place
105 */
106 checkLL(BoardIn, PieceLine, PieceColumn):-
107   (
108     getPiece(BoardIn, PieceLine, PieceColumn, '  ')
109     ;
110     getPiece(BoardIn, PieceLine, PieceColumn, 'll')
111     ;
112     getPiece(BoardIn, PieceLine, PieceColumn, 'at')
113   ).
114
115 /**
116 * Check if it is possible to make a OL move in a given place
117 */
118 checkOL(BoardIn, PieceLine, PieceColumn):-
119   (
120     getPiece(BoardIn, PieceLine, PieceColumn, '  ')
121     ;
122     getPiece(BoardIn, PieceLine, PieceColumn, 'ol')
123     ;
124     getPiece(BoardIn, PieceLine, PieceColumn, 'th')
125   ).
126
127 /**
128 * Check if it is possible to make a RR move in a given place
129 */
130 checkRR(BoardIn, PieceLine, PieceColumn):-
131   (
132     getPiece(BoardIn, PieceLine, PieceColumn, '  ')
133     ;
134     getPiece(BoardIn, PieceLine, PieceColumn, 'rr')
135     ;
136     getPiece(BoardIn, PieceLine, PieceColumn, 'at')
137   ).
138
139 /**
140 * Check if it is possible to make a LR move in a given place
141 */
142 checkLR(BoardIn, PieceLine, PieceColumn):-
143   (
144     getPiece(BoardIn, PieceLine, PieceColumn, '  ')
145     ;
146     getPiece(BoardIn, PieceLine, PieceColumn, 'lr')
147     ;
```

```
148       getPiece(BoardIn, PieceLine, PieceColumn, 'at')
149     ).
150 
151 /**
152 * Check if a piece 2 can capture in a given place
153 */
154 checkCaptureAT(CurrPlayer, BoardIn, PieceLine, PieceColumn):-
155     (
156       checkEmpty(BoardIn, PieceLine, PieceColumn)
157       ;
158       getPiece(BoardIn, PieceLine, PieceColumn, Piece),
159       Piece \= 'at',
160       name(Piece,[_|[Color|_]]),
161       name(CurrPlayer, [Ascii|_]),
162       Ascii \= Color
163     ).
164 
165 /**
166 * Check if a piece 3 or 4 can capture in a given place
167 */
168 checkCapture(CurrPlayer, BoardIn, PieceLine, PieceColumn):-
169     (
170     checkEmpty(BoardIn, PieceLine, PieceColumn)
171     ;
172     getPiece(BoardIn, PieceLine, PieceColumn, Piece),
173     name(Piece,[_|[Color|_]]),
174     name(CurrPlayer, [Ascii|_]),
175     Ascii \= Color
176     ).
```

pieceHandling.pl

```
1  /**
2  *        Gets game piece at a given position
3  */
4  getPiece(Board, Nline, Ncolumn, Piece) :-
5          getElePos(Nline, Board, Line),
6          getElePos(Ncolumn, Line, Piece).
7 
8  /**
9  *        Get piece in a given line position
10 */
11 getElePos(1, [Element|_], Element).
12 getElePos(Pos, [_|Rest], Element) :-
13         Pos > 1,
14         Next is Pos-1,
15         getElePos(Next, Rest, Element).
16 
17 /**
18 *        Gets a piece color
19 */
20 getColor(Piece, Color):-
```

```
21          name(Piece,[_|[Head|_]]),
22          name(Color, [Head]).
23
24  /**
25  *         Gets a piece number
26  */
27  getNumber(Piece, Number):-
28          name(Piece,[Head|_]),
29          name(Number,[Head]).
30
31  /**
32  *         Set piece at a given positon
33  */
34  setPiece(InBoard, Nline, Ncolumn, Piece, OutBoard) :-
35          setInLine(Nline, InBoard, Ncolumn, Piece, OutBoard).
36
37  /**
38  *         Set piece at a given line
39  */
40  setInLine(1, [Line|Rest], Ncolumn, Piece, [NewLine|Rest]):-
41          setInColumn(Ncolumn, Line, Piece, NewLine).
42  setInLine(Pos, [Line|Rest], Ncolumn, Piece, [Line|NewLine]):-
43          Pos > 1,
44          Next is Pos-1,
45          setInLine(Next, Rest, Ncolumn, Piece, NewLine).
46
47  /**
48  *         Set piece at a given column
49  */
50  setInColumn(1, [_|Rest], Piece, [Piece|Rest]).
51  setInColumn(Pos, [X|Rest], Piece, [X|NewRest]):-
52          Pos > 1,
53          Next is Pos-1,
54          setInColumn(Next, Rest, Piece, NewRest).
```