Jogo Cage

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo 2:
José Peixoto - 200603103
Luís Cruz - 201303248

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

13 de Novembro de 2016

Resumo

No âmbito da unidade curricular de Programação em Lógica, foi-nos proposto o desenvolvimento de um jogo de tabuleiro em *Prolog*: o *Cage*. O principal objetivo na realização deste projeto foi a aquisição de novas competências na expressão de conceitos lógicos em *Prolog*. Abordámos o problema de forma iterativa, tendo feito numa primeira fase, um levantamento das regras do jogo e posteriormente uma reflexão sobre a representação do estado do jogo. Uma vez implementados os procedimentos para uma manipulação básica de listas, iniciámos o desenvolvimento da interface de linha de comandos básica e posteriormente, um modo de jogo humano contra humano básico, para o qual se tentou adicionar e testar as jogadas previstas para o jogo. Numa fase final implementou-se um modo de jogo automático no qual é feita uma seleção aleatória de jogadas válidas.

Findo o projeto, realçamos a vantagem na expressividade do *Prolog* em conceitos lógicos, e a nossa carência de experiência com este paradigma de programação.

Conteúdo

1 Introdução

O principal objetivo na realização deste projeto foi a aquisição de novas competências na expressão de conceitos lógicos em *Prolog* ao implementar uma versão do jogo de tabuleiro *Cage*. Este relatório tenta explicar o estado final do projeto, salientando as suas funcionalidades e incluindo imagens ilustrativas do resultado final.

2 O Jogo Cage

O Cage é um jogo de estratégia em tabuleiro semelhante às damas que foi inventado por Mark Steere em maio de 2010. O autor descreve-o como um jogo para dois jogadores sem qualquer informação oculta. É um jogo abstrato sem fator de sorte nem empates. É jogado num tabuleiro de damas 10x10 ou 8x8 e, ao contrário do jogo original das damas, todo tabuleiro está preenchido, no início, com peças já promovidas a "damas". "Jogo de aniquilação de alta energia" é a frase escolhida pelo autor para caricaturar o jogo, uma vez que o movimento para o centro do tabuleiro assegura a aniquilação, de pelo menos, uma das cores.

2.1 Regras

O Cage é jogado por dois jogadores num tabuleiro de damas com 50 damas vermelhas e 50 damas azuis na versão de tabuleiro 10x10 ou com 32 damas vermelhas e 32 damas azuis na versão de 8x8 tabuleiro. O tabuleiro é iniciado preenchendo todas as casas com damas de cor alternada.

2.1.1 Objetivo

Para vencer é necessário capturar todas as damas inimigas. No final, pode ganhar-se mesmo que se perca a última peça que se está a movimentar (saltar) para capturar todas as damas inimigas ainda em jogo.

2.1.2 Movimentos

Existem quatro tipos de movimentos:

1. Restrito
2. Centralizador
3. Adjacente
4. Salto

Durante um turno, um jogador apenas pode utilizar um tipo de movimento.

Restrição 1 Nunca se pode colocar uma dama ortogonalmente (horizontal ou verticalmente) adjacente a uma dama de cor idêntica. Nem de forma transitória durante um turno de vários movimentos.

Restrição 2 Nunca se pode movimentar uma dama que tenha adjacências ortogonais com damas inimigas para uma casa onde tal não aconteça.

Centralizador Este movimento de uma casa, permite à dama deslocar-se na horizontal, vertical ou diagonal para uma casa vazia e que permite que a dama se aproxime do centro do tabuleiro.

Adjacente Uma dama que não tenha adjacências ortogonais com damas inimigas pode mover-se apenas uma casa em qualquer direção que contenha adjacências ortogonais com uma ou mais damas inimigas.

Salto O movimento de salto permite capturar uma dama inimiga, movimentando a dama do jogador de uma casa ortogonalmente adjacente de um lado da dama inimiga para a casa vazia adjacente do lado oposto. É possível capturar uma dama inimiga nas casas periféricas do tabuleiro de uma casa adjacente e do lado oposto da dama inimiga na borda do tabuleiro. O resultado é que quer a dama capturada quer a dama que captura são removidas do tabuleiro.

3 Lógica do Jogo

3.1 Representação do Estado do Jogo

Na representação do tabuleiro de jogo usam-se listas de listas que apenas incluem átomos para os diferentes tipos de peças (*red* e *blue*) e a casa vazia (*empty*). Para simplificação do desenvolvimento do jogo, escolheu-se a versão mais pequena do tabuleiro 8x8 com um total de 64 damas no início do jogo. O tabuleiro por sua vez é um elemento de uma lista que contém, além do tabuleiro, informação relativa ao estado do jogo, como o número de peças vermelhas e azuis, o modo de jogo, a situação de obrigação de salto e as coordenadas de uma posição da qual um salto é obrigatório.

3.1.1 Representação do estado inicial do tabuleiro:

```
[[blue,red,blue,red,blue,red,blue,red],
 [red,blue,red,blue,red,blue,red,blue],
 [blue,red,blue,red,blue,red,blue,red],
 [red,blue,red,blue,red,blue,red,blue],
 [blue,red,blue,red,blue,red,blue,red],
 [red,blue,red,blue,red,blue,red,blue],
 [blue,red,blue,red,blue,red,blue,red],
 [red,blue,red,blue,red,blue,red,blue]
]).
```

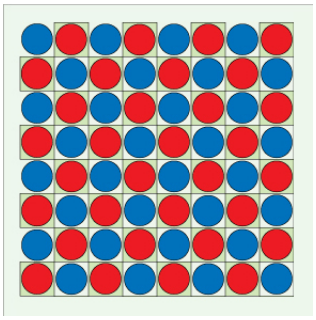

Figura 1: Estado inicial do jogo

3.2 Visualização do Tabuleiro

O tabuleiro pode ser visualizado pela linha de comandos a cada nova jogada efetuada quer pelo utilizador quer pelo computador. É disponibilizada informação acerca das peças ainda em jogo, e auxiliares na seleção de jogadas como a letra representativa de uma coluna e um número para uma linha.

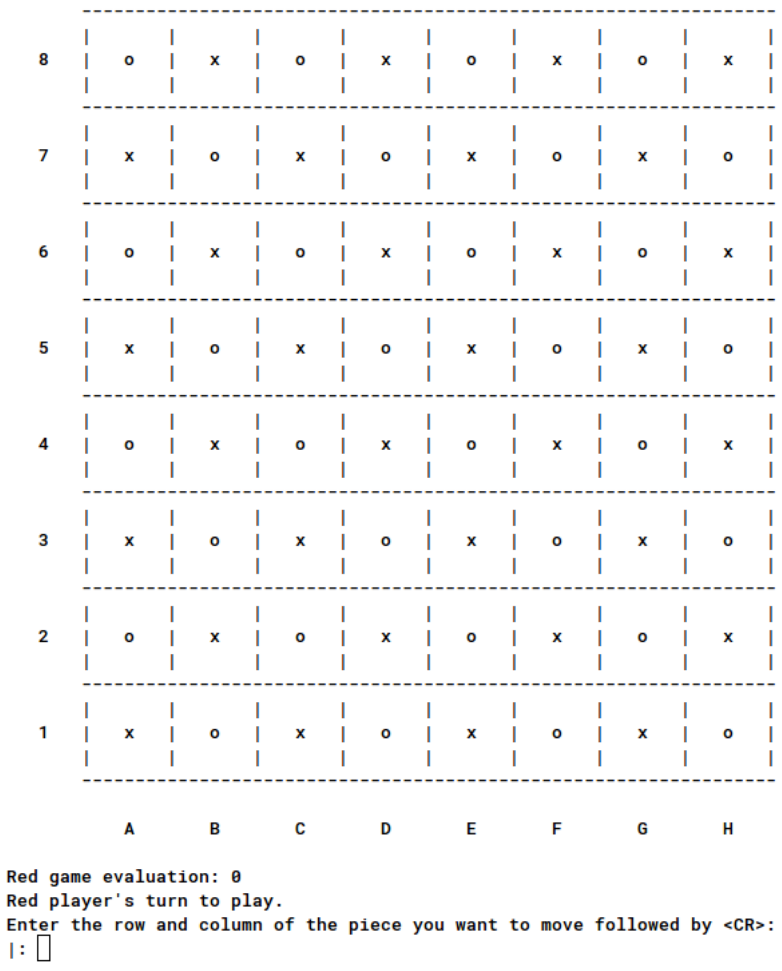

Figura 2: Estado inicial do jogo com pedido de seleção de jogada

3.3 Lista de Jogadas Válidas

Não está implementado nenhum método que angarie um conjunto de jogadas válidas, no entanto é possível determinar se existem movimentos válidos para uma dada peça do tabuleiro, sem movimentar nenhuma peça do tabuleiro, através de uma chamada à função `check_move_availability`.

```
check_move_availability(SrcRow, SrcCol, Player, Board):-
  % a move must be checked in all directions
  IncRow is SrcRow + 1,
  DecRow is SrcRow - 1,
  IncCol is SrcCol + 1,
  DecCol is SrcCol - 1,
  (
```

```
    IncRow =< 7, validate_move(SrcRow, SrcCol, IncRow, SrcCol,
        Player, Board, _, _);
    DecRow >= 0, validate_move(SrcRow, SrcCol, DecRow, SrcCol,
        Player, Board, _, _);
    IncCol =< 7, validate_move(SrcRow, SrcCol, SrcRow, IncCol,
        Player, Board, _, _);
    DecCol >= 0, validate_move(SrcRow, SrcCol, SrcRow, DecCol,
        Player, Board, _, _);
    DecRow >= 0, DecCol >= 0, validate_move(SrcRow, SrcCol,
        DecRow, DecCol, Player, Board, _, _);
    IncRow =< 7, IncCol =< 7, validate_move(SrcRow, SrcCol,
        IncRow, IncCol, Player, Board, _, _);
    DecRow >= 0, IncCol =< 7, validate_move(SrcRow, SrcCol,
        DecRow, IncCol, Player, Board, _, _);
    IncRow =< 7, DecCol >= 0, validate_move(SrcRow, SrcCol,
        IncRow, DecCol, Player, Board, _, _)
  ), !.
```

3.4 Execução de Jogadas

No caso da seleção de uma jogada manualmente através da consola, com a introdução de uma letra a representar a coluna e um número a representar uma linha, o programa tenta validar e mover uma peça do tabuleiro, tentando primeiro fazer uma jogada do tipo de salto. Quando o salto falha, são tentados outros dois tipos de movimentos: um adjacente e por fim, em último recurso, um movimento centralizador. Caso nenhum movimento seja válido, de acordo com as regras, assume-se que o jogador tem de dar a vez ao seu adversário. Em nenhum caso o jogo pode ficar numa situação na qual nenhum dos dois jogadores está sem jogadas válidas para executar.

```
make_move(SrcRow,SrcCol, DestRow, DestCol, Game, ModifiedGame):-
  (
     nl, write('Attempting to make a jump move...'), nl,
     make_jump(SrcRow,SrcCol, DestRow, DestCol, Game,
         TemporaryGame);

     write('Failed to make a jump move!'), nl, nl,
     write('Attempting to make an adjoining move...'), nl,
     make_adjoining_move(SrcRow,SrcCol, DestRow, DestCol, Game,
         TemporaryGame);

     write('Failed to make an adjoining move!'), nl, nl,
     write('Attempting to make a centering move...'), nl,
     make_centering_move(SrcRow,SrcCol, DestRow, DestCol, Game,
         TemporaryGame);

     write('Failed to make a centering move!'), nl, nl,
     get_board(Game, Board), get_player_turn(Game, Player),
     check_move_availability(SrcRow, SrcCol, Player, Board),
         ModifiedGame = Game;

     write('No valid moves were available -> Switching player
         turn!'), nl, nl,
     change_player_turn(Game,ModifiedGame), true
  ),
get_force_jump(TemporaryGame, ForceJumpMode),
(
   ForceJumpMode == noForceJump -> change_player_turn(
       TemporaryGame,ModifiedGame),! ;
   ModifiedGame = TemporaryGame
).
```

3.5 Avaliação do Tabuleiro

Apesar de ser feita uma avaliação simples do estado do jogo, não é feito nenhum aproveitamento para além da visualização deste valor no início de cada jogada. É feito um cálculo da avaliação do tabuleiro para um dado jogador, com a diferença do seu número de peças com o número de peças do adversário. Neste jogo em específico, é um cálculo relativamente acertado, ignorando os casos em que, no turno em vigor é possível fazer múltiplos saltos, sendo a avaliação um valor subestimado.

```
get_evaluation(Game,Player,Evaluation):-
    get_num_red_pieces(Game, NumRedPieces),
    get_num_blue_pieces(Game, NumBluePieces),
    (
       Player == redPlayer -> Evaluation is NumRedPieces -
           NumBluePieces;
       Player == bluePlayer -> Evaluation is NumBluePieces -
           NumRedPieces
    ),!.
```

3.6 Final do Jogo

No início de cada iteração do ciclo principal do jogo, é feita a verificação do número de peças no tabuleiro. Quando um ou mais jogadores tiver zero peças em cima do tabuleiro o jogo está terminado e o vencedor foi o último jogador que fez uma movimentação no tabuleiro. É de salientar a possibilidade de o tabuleiro final estar completamente vazio, sendo o vencedor aquele que fez o salto final e que eliminou uma peça inimiga para fora do tabuleiro.

```
validate_board_pieces(Game):-
    get_num_red_pieces(Game,NumRedPieces),
    get_num_blue_pieces(Game,NumBluePieces),
    NumRedPieces > 0,
    NumBluePieces > 0,!.
```

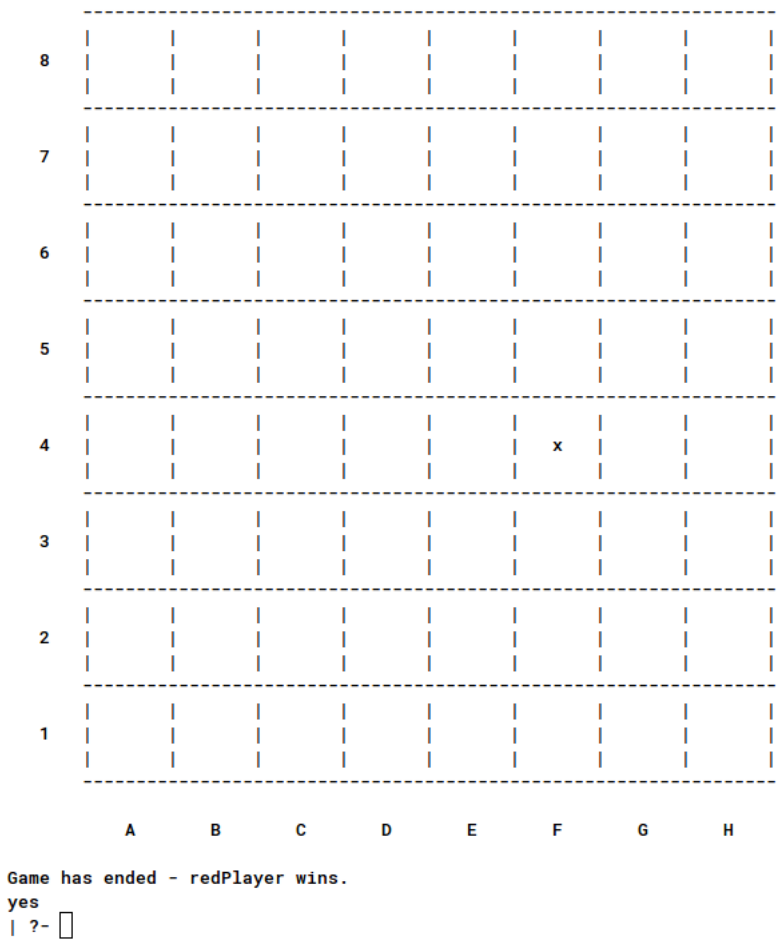

Figura 3: Estado final de um jogo com declaração de vencedor

3.7 Jogada do Computador

Não está implementada a possibilidade de seleção do modo de dificuldade de uma jogada do computador. O computador executa jogadas de forma aleatória.

4 Interface com o Utilizador

No primeiro contato que o utilizador tem com o programa é-lhe solicitado o modo de jogo de três à disposição: humano contra humano, humano contra computador e computador contra computador.

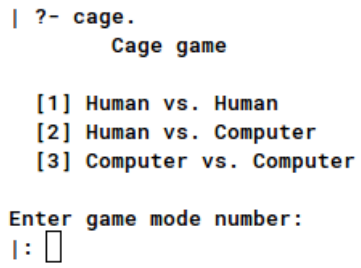

Figura 4: Menu inicial do jogo

A partir do momento em que o utilizador escolhe o modo de jogo computador contra computador, o programa inicia e finaliza rapidamente um jogo cujos movimentos são escolhidos pela simulação e execução aleatória de jogadas de forma automática. Nos outros dois modos de jogo, pode ser pedida a introdução de uma jogada a efectuar, contendo as coordenadas de partida e de destino de uma peça da cor do jogador do turno. Para tal usa-se a representação das coordenadas sob forma de uma letra de coluna seguido de um número da linha (Exemplo: $a2$ indica a coordenada de uma peça que está simultâneamente na coluna A e na linha 2).

5 Conclusões

Após a realização deste projeto, concluímos que ainda temos pouca experiência no desenvolvimento de procedimentos em programação em lógica e que muitos dos hábitos herdados de programação de outras linguagens nos trouxeram muitas situações ante problemas dos quais ainda não sabemos como contornar. É também de criticar a complexidade exagerada do trabalho, considerando o nosso conhecimento limitado e inexperiência na linguagem em questão, sendo que propomos este nível de complexidade apenas a partir de um segundo trabalho, ou um prazo de entrega mais alargado para este primeiro projeto.

Referências

[1] Sterling, Leon *The Art of Prolog*, The MIT Press 2nd edition, 2000.

[2] Abstract games, `http://www.marksteeregames.com/MSG_abstract_games.html`, 14 10 2016.

[3] Cage rules, `http://www.marksteeregames.com/Cage_rules.html`, 14 10 2016.

A Código fonte

A.1 board.pl

```
1  % cell contents
2  cell(red).
3  cell(blue).
4  cell(empty).
5
6  % symbols
7  symbol(red,'x').
8  symbol(blue,'o').
9  symbol(empty,' ').
10
11 get_cell_symbol(empty,' ').
12 get_cell_symbol(red,'x').
13 get_cell_symbol(blue,'o').
14
15 piece_owned_by(red,redPlayer).
16 piece_owned_by(blue,bluePlayer).
17
18 get_board_cell(0,Col,[HeadList|_],Symbol):-
19         get_list_element(Col,HeadList,Symbol).
20
21 get_board_cell(Row,Col,[_|TailList], Symbol):-
22         Row > 0,
23         Row1 is Row - 1,
24         get_board_cell(Row1,Col,TailList,Symbol).
25
26 % checks if both players have pieces on the board
27 validate_board_pieces(Game):-
28         get_num_red_pieces(Game,NumRedPieces),
29         get_num_blue_pieces(Game,NumBluePieces),
30         NumRedPieces > 0,
31         NumBluePieces > 0,!.
32
33 % container | 32 red | 32 blue |
34 initial_board([[blue,red,blue,red,blue,red,blue,red],
35                 [red,blue,red,blue,red,blue,red,blue],
36                 [blue,red,blue,red,blue,red,blue,red],
37                 [red,blue,red,blue,red,blue,red,blue],
38                 [blue,red,blue,red,blue,red,blue,red],
39                 [red,blue,red,blue,red,blue,red,blue],
40                 [blue,red,blue,red,blue,red,blue,red],
41                 [red,blue,red,blue,red,blue,red,blue]
42                ]).
43
44 % empty board | 0 red | 0 blue |
45 empty_board([[empty,empty,empty,empty,empty,empty,empty,empty],
46               [empty,empty,empty,empty,empty,empty,empty,empty],
47               [empty,empty,empty,empty,empty,empty,empty,empty],
48               [empty,empty,empty,empty,empty,empty,empty,empty],
49               [empty,empty,empty,empty,empty,empty,empty,empty],
50               [empty,empty,empty,empty,empty,empty,empty,empty],
51               [empty,empty,empty,empty,empty,empty,empty,empty],
52               [empty,empty,empty,empty,empty,empty,empty,empty]
53              ]).
54
55 % test board | 1 red | 4 blue |
56 jump_test_board([[empty,empty,empty,empty,empty,empty,empty,
       empty],
57                   [empty,empty,empty,empty,empty,empty,empty,
                         empty],
58                   [empty,empty,empty,empty,empty,empty,empty,
                         empty],
```

```
59                  [empty,empty,empty,blue,empty,blue,empty,empty
                        ],
60                  [empty,empty,blue,empty,blue,empty,empty,empty
                        ],
61                  [empty,empty,red,empty,empty,empty,empty,empty
                        ],
62                  [empty,empty,empty,empty,empty,empty,empty,
                        empty],
63                  [empty,empty,empty,empty,empty,empty,empty,
                        empty]
64                 ]).
65
66  bot_test_board([[blue,red,blue,red,blue,red,blue,red],
67                  [red,blue,red,blue,red,blue,red,blue],
68                  [blue,red,blue,red,blue,red,empty,empty],
69                  [red,blue,red,blue,red,blue,red,blue],
70                  [blue,red,blue,red,blue,red,blue,red],
71                  [red,blue,red,blue,red,blue,red,blue],
72                  [blue,red,blue,empty,blue,red,blue,red],
73                  [red,blue,red,empty,red,blue,red,blue]
74                 ]).
75
76  % bot winning test | 1 red | 3 blue |
77  bot_win_test_board([[empty,empty,empty,empty,empty,empty,empty,
        empty],
78                      [empty,empty,empty,empty,empty,empty,empty,
                            empty],
79                      [empty,empty,empty,empty,empty,empty,empty,
                            blue],
80                      [empty,empty,empty,red,blue,empty,blue,empty
                            ],
81                      [empty,empty,empty,empty,empty,empty,empty,
                            empty],
82                      [empty,empty,empty,empty,empty,empty,empty,
                            empty],
83                      [empty,empty,empty,empty,empty,empty,empty,
                            empty],
84                      [empty,empty,empty,empty,empty,empty,empty,
                            empty]
85                     ]).
86
87  % display board
88  display_board([H|T], R) :-
89          % how to display 1st line border?
90          write('    '), write('
                ------------------------------------------------------------------
                 '), nl,
91          write('    '), display_empty_line([]),
92          write('   '), write(R), display_line(H), nl,
93          write('    '), display_empty_line([]),
94          R1 is R - 1,
95          display_board(T,R1).
96
97  display_board([],_):-
98          write('     '), write('
                ------------------------------------------------------------------
                 '), nl, nl,
99          write('     '), write('      A       B       C       D
                      E       F       G       H       '), nl.
100
101 % display line
102 display_line([H|T]) :-
103         symbol(H,S),
104         write('   |   '), write(S),
105         display_line(T).
```

```
106 display_line([]) :-
107         write('   |   ').
108
109 %display empty line
110 display_empty_line([]):-
111         write('   '), write('|'),write('       '),
112         write('|'), write('       '), write('|'),
113         write('       '), write('|'), write('       '),
114         write('|'), write('       '), write('|'), write('
            '),
115         write('|'), write('       '), write('|'), write('
            '),
116         write('|'), nl.
```

A.2 cage.pl

```
1  % includes
2  :- use_module(library(random)).
3  :- use_module(library(system)).
4  :- include('utils.pl').
5  :- include('cli.pl').
6  :- include('game.pl').
7  :- include('computer.pl').
8  :- include('board.pl').
9  :- include('logic.pl').
10
11 % program starting pointc
12 cage:-
13         initialize_random_seed,
14         main_menu.
15
16 % players and pieces
17 player(redPlayer).
18 player(bluePlayer).
19
20 get_player_name(redPlayer, 'Red').
21 get_player_name(bluePlayer, 'Blue').
22
23 player_ownes_cell(Row,Col,Game):-
24         get_board(Game,Board),
25         get_player_turn(Game,Player),
26         get_board_cell(Row,Col,Board,Cell),
27         piece_owned_by(Cell,Player).
28
29 game_loop(Game):-
30         validate_board_pieces(Game),
31         get_board(Game, Board),
32         display_board(Board,8),
33         get_mode(Game, GameMode),
34         (
35            GameMode == cvc -> (computer_play(0,Game,ModifiedGame
               ));
36            GameMode == hvc -> (
37                                  (get_player_turn(Game, Player),
                                       Player \== redPlayer) ->
                                      computer_play(0, Game,
                                      ModifiedGame);
38                                  human_play(Game, ModifiedGame)
39                               );
40            human_play(Game, ModifiedGame)
41         ),!,
42           game_loop(ModifiedGame).
43
44 game_loop(Game):-
45         get_board(Game, Board),
```

```
46          display_board(Board,8),
47          determine_winner(Game, Winner),
48          nl, write('Game has ended - '), write(Winner), write('
                wins.'), nl.
49
50  determine_winner(Game, Winner):-
51          get_previous_turn(Game, PreviousPlayer),
52          get_num_red_pieces(Game,NumRedPieces),
53          get_num_blue_pieces(Game,NumBluePieces),
54
55          (
56             NumRedPieces > 0 -> Winner = redPlayer;
57             NumBluePieces > 0 -> Winner = bluePlayer;
58             Winner = PreviousPlayer
59          ).
60
61
62  human_play(Game, ModifiedGame):-
63          get_player_turn(Game, Player),
64          get_board(Game, Board),
65
66          repeat,
67          display_turn_info(Game), nl,
68          get_moving_piece_source_coordinates(SrcRow, SrcCol),
69          validate_piece_owner(SrcRow, SrcCol, Board, Player),
70
71          get_force_jump(Game,ForceMode),
72          get_force_starting_row(Game,ForceJumRow),
73          get_force_starting_col(Game,ForceJumCol),
74
75          (
76             ForceMode == forceJump -> write('A jumping move from
                    the same spot is mandatory'), nl,
77                                       SrcRow == ForceJumRow,
                                             SrcCol == ForceJumCol;
78             true
79          ),
80
81          get_piece_destiny_coordinates(DestRow, DestCol),
82          validate_source_to_destiny_delta(SrcRow, SrcCol, DestRow
                , DestCol),
83
84          (
85             ForceMode == forceJump -> write('A jumping move is
                    mandatory'), nl,
86                                       get_enemy_piece(Player,
                                             EnemyPiece),
87                                       validate_cell_contents(
                                             DestRow, DestCol, Board
                                             , EnemyPiece);
88             validate_destiny_cell_type(DestRow, DestCol, Board,
                    Player)
89          ),
90
91          make_move(SrcRow, SrcCol, DestRow, DestCol, Game,
                ModifiedGame), !.
```

A.3 cli.pl

```
1   get_moving_piece_source_coordinates(SrcRow, SrcCol):-
2           write('Enter the column and row (Ex: a1) of the piece to
                 move followed by <CR>: '), nl,
3           get_coordinates(SrcRow,SrcCol), nl.
4
5   get_piece_destiny_coordinates(DstRow, DstCol):-
```

```
6           write('Enter the destiny column and row (Ex: a1) of the
                piece to move followed by <CR>:'), nl,
7           get_coordinates(DstRow,DstCol), nl.
8
9   get_integer(Input):-
10          get_code(TempInput),
11          Input is TempInput - 48.
12
13  get_return_key:-
14          get_code(_).
15
16  get_coordinates(Row,Col):-
17          get_integer(C),
18          get_integer(R),
19          get_return_key,
20          Row is 8-R,
21          Col is C-49.
22
23  display_turn_info(Game):-
24          get_player_turn(Game, Player),
25          get_player_name(Player, PlayerName),
26          get_evaluation(Game, Player, Evaluation),
27          nl, write(PlayerName), write(' game evaluation: '),
                write(Evaluation), nl,
28          write(PlayerName), write(' player\'s turn to play.'), !.
29
30  main_menu:-
31          print_menu,
32          get_char(Input),
33          get_char(_),
34          (
35              Input == '1' -> human_vs_human(Game), game_loop(Game)
                    ;
36              Input == '2' -> human_vs_computer(Game), game_loop(
                    Game);
37              Input == '3' -> computer_vs_computer(Game), game_loop
                    (Game);
38              main_menu
39          ).
40
41  print_menu:-
42          write('        Cage game'), nl, nl,
43          write('  [1] Human vs. Human'), nl,
44          write('  [2] Human vs. Computer'), nl,
45          write('  [3] Computer vs. Computer'), nl, nl,
46          write('Enter game mode number:'), nl.
```

A.4 computer.pl

```
1   validate_force_jump(DestRow, DestCol, Game):-
2           get_player_turn(Game, Player),
3           get_board(Game, Board),
4           get_force_jump(Game,ForceMode),
5           (       ForceMode == forceJump -> write('A jumping move
                    is mandatory'), nl,
6                                             get_enemy_piece(Player
                                                  , EnemyPiece),
7                                             validate_cell_contents
                                                  (DestRow, DestCol,
                                                   Board, EnemyPiece
                                                  );
8                   ForceMode == noForceJump;
9                   fail
10          ),!.
11
```

```
12  computer_play(0, Game, ModifiedGame):-
13          get_player_turn(Game, Player),
14          get_board(Game, Board),
15          get_force_jump(Game,ForceMode),
16          repeat,
17          (
18             ForceMode == forceJump -> get_force_starting_row(Game
                   ,StartRandRow), get_force_starting_col(Game,
                   StartRandCol);
19
20             random(0,8,StartRandRow),
21             random(0,8,StartRandCol),
22             get_matrix_element(StartRandRow,StartRandCol,Board,
                   BoardPiece),
23             piece_owned_by(BotPiece,Player),
24             BoardPiece == BotPiece
25          ),
26
27          check_move_availability(StartRandRow, StartRandCol,
                Player, Board), !,
28
29          IncRow is StartRandRow + 1,
30          DecRow is StartRandRow - 1,
31          IncCol is StartRandCol + 1,
32          DecCol is StartRandCol - 1,
33          repeat,
34          random(0, 8, RandomMove),
35          (
36             (RandomMove =:= 0, validate_force_jump(IncRow,
                   StartRandCol,Game), validate_move(StartRandRow,
                   StartRandCol, IncRow, StartRandCol, Player, Board
                   , DestRow, DestCol));
37             (RandomMove =:= 1, validate_force_jump(DecRow,
                   StartRandCol,Game), validate_move(StartRandRow,
                   StartRandCol, DecRow, StartRandCol, Player, Board
                   , DestRow, DestCol));
38             (RandomMove =:= 2, validate_force_jump(StartRandRow,
                   IncCol,Game), validate_move(StartRandRow,
                   StartRandCol, StartRandRow, IncCol, Player, Board
                   , DestRow, DestCol));
39             (RandomMove =:= 3, validate_force_jump(StartRandRow,
                   DecCol,Game), validate_move(StartRandRow,
                   StartRandCol, StartRandRow, DecCol, Player, Board
                   , DestRow, DestCol));
40             (RandomMove =:= 4, validate_force_jump(IncRow,IncCol,
                   Game), validate_move(StartRandRow, StartRandCol,
                   IncRow, IncCol, Player, Board, DestRow, DestCol))
                   ;
41             (RandomMove =:= 5, validate_force_jump(DecRow,DecCol,
                   Game), validate_move(StartRandRow, StartRandCol,
                   DecRow, DecCol, Player, Board, DestRow, DestCol))
                   ;
42             (RandomMove =:= 6, validate_force_jump(DecRow,IncCol,
                   Game), validate_move(StartRandRow, StartRandCol,
                   DecRow, IncCol, Player, Board, DestRow, DestCol))
                   ;
43             (RandomMove =:= 7, validate_force_jump(IncRow,DecCol,
                   Game), validate_move(StartRandRow, StartRandCol,
                   IncRow, DecCol, Player, Board, DestRow, DestCol))
                   ;
44             fail
45          ), make_move(StartRandRow, StartRandCol, DestRow,
                DestCol, Game, ModifiedGame), !.
```

A.5 game.pl

```
% human vs human mode
human_vs_human(Game):-
        %               initial_board(Board),
        initial_board(Board),
        Game = [Board, [32, 32], redPlayer, hvh, noForceJump, 0,
             0], !.

% human vs computer mode
human_vs_computer(Game):-
        initial_board(Board),
        Game = [Board, [32, 32], redPlayer, hvc, noForceJump, 0,
             0], !.

% human vs computer mode
computer_vs_computer(Game):-
        %        bot_test_board(Board),
        %        bot_win_test_board(Board),
        initial_board(Board),
        Game = [Board, [32, 32], redPlayer, cvc, noForceJump, 0,
             0], !.

% board procedures
get_board([Board|_], Board).

set_board(Board, Game, ModifiedGame):-
        set_list_element(0, Board, Game, ModifiedGame).

% board evaluation
get_evaluation(Game,Player,Evaluation):-
        get_num_red_pieces(Game, NumRedPieces),
        get_num_blue_pieces(Game, NumBluePieces),
        (
           Player == redPlayer -> Evaluation is NumRedPieces -
               NumBluePieces;
           Player == bluePlayer -> Evaluation is NumBluePieces -
                NumRedPieces
        ),!.

% pieces procedures
get_num_board_pieces(Game,ListOfPieces):-
        get_list_element(1,Game,ListOfPieces).

set_num_board_pieces(NumPiecesList, Game, ModifiedGame):-
        set_list_element(1, NumPiecesList, Game, ModifiedGame).

get_num_red_pieces(Game,NumRedPieces):-
        get_num_board_pieces(Game,ListOfPieces),
        get_list_element(0,ListOfPieces,NumRedPieces).

set_num_red_pieces(NumRedPieces, Game, ModifiedGame):-
        get_num_board_pieces(Game, NumPiecesList),
        set_list_element(0, NumRedPieces, NumPiecesList,
            ResNumPiecesList),
        set_num_board_pieces(ResNumPiecesList, Game,
            ModifiedGame).

get_num_blue_pieces(Game,NumBluePieces):-
        get_num_board_pieces(Game,ListOfPieces),
        get_list_element(1,ListOfPieces,NumBluePieces).

set_num_blue_pieces(NumBluePieces, Game, ModifiedGame):-
        get_num_board_pieces(Game, NumPiecesList),
        set_list_element(1, NumBluePieces, NumPiecesList,
            ResNumPiecesList),
```

```
57          set_num_board_pieces(ResNumPiecesList, Game,
                ModifiedGame).
58
59  dec_piece(Piece,Game,ModifiedGame):-
60          (
61              Piece == red -> dec_num_red_pieces(Game, ModifiedGame
                    );
62              Piece == blue -> dec_num_blue_pieces(Game,
                    ModifiedGame)
63          ),!.
64
65  dec_num_red_pieces(Game, ModifiedGame):-
66          get_num_red_pieces(Game, NumRedPieces),
67          NumRedPieces1 is NumRedPieces - 1,
68          set_num_red_pieces(NumRedPieces1, Game, ModifiedGame).
69
70  dec_num_blue_pieces(Game, ModifiedGame):-
71          get_num_blue_pieces(Game, NumBluePieces),
72          NumBluePieces1 is NumBluePieces - 1,
73          set_num_blue_pieces(NumBluePieces1, Game, ModifiedGame).
74
75  % player
76  get_player_turn(Game,Player):-
77          get_list_element(2,Game,Player).
78
79  get_previous_turn(Game,PreviousPlayer):-
80          get_list_element(2,Game,Player),
81          Player == redPlayer, PreviousPlayer = bluePlayer;
82          PreviousPlayer = redPlayer.
83
84
85  set_player_turn(Player, Game, ModifiedGame):-
86          set_list_element(2, Player, Game, ModifiedGame).
87
88  change_player_turn(TemporaryGame,ModifiedGame):-
89          get_player_turn(TemporaryGame,OldTurn),
90          (
91              OldTurn == redPlayer -> NewTurn = bluePlayer;
92              OldTurn == bluePlayer -> NewTurn = redPlayer
93          ), set_player_turn(NewTurn,TemporaryGame,ModifiedGame)
                ,!.
94
95  get_enemy_piece(Player, EnemyPiece):-
96          piece_owned_by(PlayerPiece,Player),
97          (
98              PlayerPiece == red -> EnemyPiece = blue;
99              PlayerPiece == blue -> EnemyPiece = red
100         ).
101
102 % game mode
103 get_mode(Game,Mode):-
104         get_list_element(3,Game,Mode).
105
106 % jump forcing variable
107 get_force_jump(Game,ForceMode):-
108         get_list_element(4,Game,ForceMode).
109
110 get_force_starting_row(Game,Row):-
111         get_list_element(5,Game,Row).
112
113 get_force_starting_col(Game,Col):-
114         get_list_element(6,Game,Col).
115
116 set_force_jump(ForceMode, ForceStartingRow, ForceStartingCol,
        Game, ModifiedGame):-
```

```
117         set_list_element(4, ForceMode, Game, TemporaryGame),
118         set_force_jump_starting_row(ForceStartingRow,
                TemporaryGame, TemporaryGame2),
119         set_force_jump_starting_col(ForceStartingCol,
                TemporaryGame2, ModifiedGame).
120
121 set_force_jump_starting_row(Row, Game, ModifiedGame):-
122         set_list_element(5, Row, Game, ModifiedGame).
123
124 set_force_jump_starting_col(Col, Game, ModifiedGame):-
125         set_list_element(6, Col, Game, ModifiedGame).
```

A.6 logic.pl

```
1  % attempt to make a move
2  make_move(SrcRow,SrcCol, DestRow, DestCol, Game, ModifiedGame):-
3          (
4             nl, write('Attempting to make a jump move...'), nl,
5             make_jump(SrcRow,SrcCol, DestRow, DestCol, Game,
                  TemporaryGame);
6
7             write('Failed to make a jump move!'), nl, nl,
8             write('Attempting to make an adjoining move...'), nl,
9             make_adjoining_move(SrcRow,SrcCol, DestRow, DestCol,
                  Game, TemporaryGame);
10
11            write('Failed to make an adjoining move!'), nl, nl,
12            write('Attempting to make a centering move...'), nl,
13            make_centering_move(SrcRow,SrcCol, DestRow, DestCol,
                  Game, TemporaryGame);
14
15            write('Failed to make a centering move!'), nl, nl,
16            get_board(Game, Board), get_player_turn(Game, Player)
                  ,
17            check_move_availability(SrcRow, SrcCol, Player, Board
                  ), ModifiedGame = Game;
18
19            write('No valid moves were available -> Switching
                  player turn!'), nl, nl,
20            change_player_turn(Game,ModifiedGame), true
21         ),
22         get_force_jump(TemporaryGame, ForceJumpMode),
23         (
24            ForceJumpMode == noForceJump -> change_player_turn(
                  TemporaryGame,ModifiedGame),! ;
25            ModifiedGame = TemporaryGame
26         ).
27
28 move_piece(SrcRow, SrcCol, DestRow, DestCol, Board,
       ModifiedBoard):-
29         get_matrix_element(SrcRow,SrcCol,Board,SrcElem),
30         set_matrix_element(SrcRow,SrcCol,empty,Board,
               TemporaryBoard),
31         set_matrix_element(DestRow,DestCol,SrcElem,
               TemporaryBoard,ModifiedBoard).
32
33 make_centering_move(SrcRow,SrcCol, DestRow, DestCol, Game,
       ModifiedGame):-
34         get_board(Game,Board),
35         get_player_turn(Game, Player),
36         validate_centering_move(SrcRow, SrcCol, DestRow, DestCol
               , Player, Board),
37         move_piece(SrcRow, SrcCol, DestRow, DestCol, Board,
               ModifiedBoard),
38         set_board(ModifiedBoard, Game, ModifiedGame).
```

```
39
40 make_adjoining_move(SrcRow,SrcCol, DestRow, DestCol, Game,
       ModifiedGame):-
41         get_board(Game,Board),
42         get_player_turn(Game,Player),
43         validate_adjoining_move(SrcRow, SrcCol, DestRow, DestCol
               , Player, Board),
44         move_piece(SrcRow, SrcCol, DestRow, DestCol, Board,
               ModifiedBoard),
45         set_board(ModifiedBoard, Game, ModifiedGame).
46
47 make_jump(SrcRow, SrcCol, DestRow, DestCol, Game, ModifiedGame)
       :-
48         get_board(Game,Board),
49         get_player_turn(Game,Player),
50         validate_jump(SrcRow, SrcCol, DestRow, DestCol, Player,
               Board, JumpDestinyRow, JumpDestinyCol),
51         capture_piece(SrcRow, SrcCol, DestRow, DestCol,
               JumpDestinyRow, JumpDestinyCol, Game, TemporaryGame)
               ,
52         get_board(TemporaryGame,ModifiedBoard),
53         (
54            JumpDestinyRow >= 0,
55            JumpDestinyRow =< 7,
56            JumpDestinyCol >= 0,
57            JumpDestinyCol =< 7,
58            validate_force_jump(DestRow, DestCol, JumpDestinyRow,
                   JumpDestinyCol, Player, ModifiedBoard),
59            set_force_jump(forceJump, JumpDestinyRow,
                   JumpDestinyCol, TemporaryGame, ModifiedGame),
60            write('Jumping move will be forced next turn.'), nl,
                   nl;
61
62            set_force_jump(noForceJump, 0, 0, TemporaryGame,
                   ModifiedGame),
63            write('Jumping move is not forced in the next turn.')
                   , nl, nl
64         ), !.
65
66 capture_piece(SrcRow, SrcCol, DestRow, DestCol, JumpDestinyRow,
       JumpDestinyCol, Game, ModifiedGame):-
67         % get current board
68         get_board(Game, Board),
69
70         % empty the captured piece cell
71         get_matrix_element(DestRow, DestCol, Board,
               RemovedCapturedPiece),
72         set_matrix_element(DestRow, DestCol, empty, Board,
               TemporaryBoard),
73         dec_piece(RemovedCapturedPiece,Game,TemporaryGame),
74
75         (
76            % piece jumped out of the board -> clean two pieces
77            (JumpDestinyRow < 0 ; JumpDestinyRow > 7 ;
                   JumpDestinyCol < 0 ; JumpDestinyCol > 7) ->
                   get_matrix_element(SrcRow, SrcCol, TemporaryBoard
                   , RemovedPiece),
78
```

```
79
80
81
82              (JumpDestinyRow >= 0, JumpDestinyRow =< 7,
                    JumpDestinyCol =< 7, JumpDestinyCol >= 0) ->
                    move_piece(SrcRow, SrcCol, JumpDestinyRow,
                    JumpDestinyCol, TemporaryBoard, ModifiedBoard),
83
84          ).
85
86
87
88  get_jump_destiny_cell_coordinates(SrcRow, SrcCol, DestRow,
        DestCol, JumpType, EmptyCellRow, EmptyCellCol):-
89          DeltaRow is DestRow - SrcRow,
90          DeltaCol is DestCol - SrcCol,
91          (
92              JumpType == horizontal ->
93              (
94                  (DeltaCol > 0) -> EmptyCellCol is DestCol + 1,
                        EmptyCellRow is SrcRow;
95                  (DeltaCol < 0) -> EmptyCellCol is DestCol - 1,
                        EmptyCellRow is SrcRow
96              );
97              JumpType == vertical ->
```

```
98              (
99                 (DeltaRow > 0) -> EmptyCellRow is DestRow + 1,
                       EmptyCellCol is SrcCol;
100                (DeltaRow < 0) -> EmptyCellRow is DestRow - 1,
                       EmptyCellCol is SrcCol
101             )
102         ),!.
103
104 validate_adjoining_move(SrcRow, SrcCol, DestRow, DestCol, Player
        , Board):-
105         get_enemy_piece(Player,EnemyPiece),
106         validate_ortogonal_adjancencies(SrcRow, SrcCol, SrcRow,
                SrcCol, EnemyPiece, Board),!,
107         \+validate_ortogonal_adjancencies(SrcRow, SrcCol,
                DestRow, DestCol, EnemyPiece, Board), !.
108
109 validate_jump(SrcRow, SrcCol, DestRow, DestCol, Player, Board,
        JumpDestinyRow, JumpDestinyCol):-
110         % selected destiny cell must contain an opponent piece
111         (
112            Player == redPlayer -> validate_cell_contents(DestRow
                   , DestCol, Board, blue);
113            Player == bluePlayer -> validate_cell_contents(
                   DestRow, DestCol, Board, red)
114         ),
115
116         % we first need to know if the jump is horizontal or
              vertical
117         % get the destiny empty cell coordinates
118         % real destiny cell must be empty
119         get_jump_type(SrcRow, SrcCol, DestRow, DestCol, JumpType
                ),
120         get_jump_destiny_cell_coordinates(SrcRow, SrcCol,
                DestRow, DestCol, JumpType, JumpDestinyRow,
                JumpDestinyCol),
121
122         (
123            (JumpDestinyRow >= 0, JumpDestinyRow =< 7,
                   JumpDestinyCol >= 0, JumpDestinyCol =< 7) ->
                   validate_cell_contents(JumpDestinyRow,
                   JumpDestinyCol, Board, empty),
124
125
```

```
126             true
127         ), !.
128
129 validate_move(SrcRow, SrcCol, DestRow, DestCol, Player, Board,
        ValidDestRow, ValidDestCol):-
130         validate_jump(SrcRow, SrcCol, DestRow, DestCol, Player,
                Board, _, _) -> ValidDestRow = DestRow, ValidDestCol
                 = DestCol;
131         validate_force_jump_cell_contents(DestRow, DestCol,
                Board, empty), validate_adjoining_move(SrcRow,
                SrcCol, DestRow, DestCol, Player, Board) ->
                ValidDestRow = DestRow, ValidDestCol = DestCol;
132         validate_force_jump_cell_contents(DestRow, DestCol,
                Board, empty), validate_centering_move(SrcRow,
                SrcCol, DestRow, DestCol, Player, Board) ->
                ValidDestRow = DestRow, ValidDestCol = DestCol,!.
133
134 check_move_availability(SrcRow, SrcCol, Player, Board):-
135         % a move must be checked in all directions
136         IncRow is SrcRow + 1,
137         DecRow is SrcRow - 1,
138         IncCol is SrcCol + 1,
139         DecCol is SrcCol - 1,
140         (
141             IncRow =< 7, validate_move(SrcRow, SrcCol, IncRow,
                    SrcCol, Player, Board, _, _);
142             DecRow >= 0, validate_move(SrcRow, SrcCol, DecRow,
                    SrcCol, Player, Board, _, _);
143             IncCol =< 7, validate_move(SrcRow, SrcCol, SrcRow,
                    IncCol, Player, Board, _, _);
144             DecCol >= 0, validate_move(SrcRow, SrcCol, SrcRow,
                    DecCol, Player, Board, _, _);
145             DecRow >= 0, DecCol >= 0, validate_move(SrcRow,
                    SrcCol, DecRow, DecCol, Player, Board, _, _);
146             IncRow =< 7, IncCol =< 7, validate_move(SrcRow,
                    SrcCol, IncRow, IncCol, Player, Board, _, _);
147             DecRow >= 0, IncCol =< 7, validate_move(SrcRow,
                    SrcCol, DecRow, IncCol, Player, Board, _, _);
148             IncRow =< 7, DecCol >= 0, validate_move(SrcRow,
                    SrcCol, IncRow, DecCol, Player, Board, _, _)
149         ), !.
150
151 get_jump_type(SrcRow, SrcCol, DestRow, DestCol, JumpType):-
152         (
153             SrcRow == DestRow -> JumpType = horizontal;
154             SrcCol == DestCol -> JumpType = vertical
155         ).
156
157 validate_force_jump(CapturedRow, CapturedCol, JumpDestinyRow,
        JumpDestinyCol, Player, Board):-
158         piece_owned_by(PlayerPiece,Player),
159         get_enemy_piece(Player,EnemyPiece),
160         IncRow is JumpDestinyRow + 1,
161         DecRow is JumpDestinyRow - 1,
162         IncCol is JumpDestinyCol + 1,
163         DecCol is JumpDestinyCol - 1,!,
164         (
165             (IncRow =< 7, validate_force_jump_cell_contents(
                    IncRow, JumpDestinyCol, Board, EnemyPiece),
166              IncRow2 is IncRow + 1,
```

```
167             (
168                 IncRow2 =<7 ->(
169                                     validate_force_jump_cell_contents
                                        (IncRow2, JumpDestinyCol,
                                        Board, empty),
170                                     validate_ortogonal_adjancencies(
                                        CapturedRow, CapturedCol,
                                        IncRow2, JumpDestinyCol,
                                        PlayerPiece, Board)
171                                 );
172                 true
173             )
174             );
175
176             (DecRow >= 0, validate_force_jump_cell_contents(
                    DecRow, JumpDestinyCol, Board, EnemyPiece),
177              DecRow2 is DecRow - 1,
178              (
179                 DecRow2 >= 0 -> (
180                                     validate_force_jump_cell_contents
                                        (DecRow2, JumpDestinyCol,
                                        Board, empty),
181                                     validate_ortogonal_adjancencies
                                        (CapturedRow, CapturedCol,
                                         DecRow2, JumpDestinyCol,
                                        PlayerPiece, Board)
182                                 );
183                 true
184              )
185             );
186
187             (IncCol =< 7, validate_force_jump_cell_contents(
                    JumpDestinyRow, IncCol, Board, EnemyPiece),
188              IncCol2 is IncCol + 1,
189              (
190                 IncCol2 =< 7 -> (
191                                     validate_force_jump_cell_contents
                                        (JumpDestinyRow, IncCol2,
                                        Board, empty),
192                                     validate_ortogonal_adjancencies
                                        (CapturedRow, CapturedCol,
                                         JumpDestinyRow, IncCol2,
                                        PlayerPiece, Board)
193                                 );
194                 true
195              )
196             );
197
198             (DecCol >= 0, validate_force_jump_cell_contents(
                    JumpDestinyRow, DecCol, Board, EnemyPiece),
199              DecCol2 is DecCol - 1,
200              (
201                 DecCol2 >= 0 -> (
202                                     validate_force_jump_cell_contents
                                        (JumpDestinyRow, DecCol2,
                                        Board, empty),
203                                     validate_ortogonal_adjancencies
                                        (CapturedRow, CapturedCol,
                                         JumpDestinyRow, DecCol2,
                                        PlayerPiece, Board)
204                                 );
205                 true
206              )
207             )
208         ).
```

```
209
210 validate_centered(SrcRow,SrcCol):-
211         SrcRow == 3 -> (SrcCol == 4; SrcCol == 3);
212         SrcRow == 4 -> (SrcCol == 4; SrcCol == 3);
213         fail.
214
215 get_quadrant(SrcRow,SrcCol,Quadrant):-
216         SrcRow >=0, SrcRow =< 3, SrcCol >= 0, SrcCol =< 3 ->
                Quadrant = 1;
217         SrcRow >=0, SrcRow =< 3, SrcCol >= 4, SrcCol =< 7 ->
                Quadrant = 2;
218         SrcRow >=4, SrcRow =< 7, SrcCol >= 0, SrcCol =< 3 ->
                Quadrant = 3;
219         SrcRow >=4, SrcRow =< 7, SrcCol >= 4, SrcCol =< 7 ->
                Quadrant = 4.
220
221 validate_centering_move(SrcRow, SrcCol, DestRow, DestCol, Player
        , Board):-
222         \+ validate_centered(SrcRow,SrcCol), !,
223         piece_owned_by(PlayerPiece, Player), !,
224         validate_ortogonal_adjancencies(SrcRow, SrcCol, DestRow,
                 DestCol, PlayerPiece, Board),
225         get_quadrant(SrcRow,SrcCol,Quadrant),
226         DestRow >= 0, DestRow =< 7,
227         DestCol >= 0, DestCol =< 7,
228         DeltaRow is DestRow - SrcRow,
229         DeltaCol is DestCol - SrcCol,
230         (
231            Quadrant == 1 -> (
232                                DeltaRow > 0, DeltaCol >= 0;
233                                DeltaCol > 0, DeltaRow >= 0
234                             );
235            Quadrant == 2 -> (
236                                DeltaRow > 0, DeltaCol =< 0;
237                                DeltaCol < 0, DeltaRow >= 0
238                             );
239            Quadrant == 3 -> (
240                                DeltaRow < 0, DeltaCol >= 0;
241                                DeltaCol > 0, DeltaRow =< 0
242                             );
243            Quadrant == 4 -> (
244                                DeltaRow < 0, DeltaCol =< 0;
245                                DeltaCol < 0, DeltaRow =< 0
246                             )
247         ),!.
248
249 validate_ortogonal_adjancencies(SrcRow, SrcCol, DestRow, DestCol
        , AvoidPiece, Board):-
250         IncRow is DestRow + 1,
251         DecRow is DestRow - 1,
252         IncCol is DestCol + 1,
253         DecCol is DestCol - 1, !,
254         (
255            SrcRow \== IncRow -> (
                validate_ortogonal_cell_contents(IncRow, DestCol,
                 Board, AvoidPiece),!);
256            true
257         ),
258         (
259            SrcRow \== DecRow -> (
                validate_ortogonal_cell_contents(DecRow, DestCol,
                 Board, AvoidPiece),!);
260            true
261         ),
262         (
```

```
263             SrcCol \== IncCol -> (
                    validate_ortogonal_cell_contents(DestRow, IncCol,
                     Board, AvoidPiece),!);
264             true
265         ),
266         (
267             SrcCol \== DecCol -> (
                    validate_ortogonal_cell_contents(DestRow, DecCol,
                     Board, AvoidPiece),!);
268             true
269         ),!.
270
271 validate_ortogonal_adjancencies(_, _, _, _, _, _):-
272         write('Destiny cell with bad ortogonal adjacency!'), nl,
273         fail.
274
275 validate_source_to_destiny_delta(SrcRow, SrcCol, DestRow,
        DestCol):-
276         DeltaRow is abs(DestRow - SrcRow),
277         DeltaCol is abs(DestCol - SrcCol),
278         DeltaRow =< 1, DeltaCol =< 1,!.
279
280 validate_source_to_destiny_delta(_, _, _, _):-
281         write('Invalid destiny cell distance delta!'), nl,
282         fail.
283
284 validate_ortogonal_cell_contents(Row, Col, Board,
        ExpectedContent):-
285         (
286             (Row >= 0, Row =< 7, Col >= 0, Col =< 7) -> (
                    get_matrix_element(Row, Col, Board, Piece),
287                                                         Piece \=
                                                             ExpectedContent
                                                             , !)
                                                             ;
288             true
289         ), !.
290
291 validate_ortogonal_cell_contents(_, _, _, _):-
292         fail.
293
294 validate_force_jump_cell_contents(Row, Col, Board,
        ExpectedContent):-
295         get_matrix_element(Row, Col, Board, Piece),
296         Piece == ExpectedContent, !.
297
298 validate_force_jump_cell_contents(_, _, _, _):-
299         fail.
300
301 validate_cell_contents(Row, Col, Board, ExpectedContent):-
302         get_matrix_element(Row, Col, Board, Piece),
303         Piece == ExpectedContent, !.
304
305 validate_cell_contents(_, _, _, _):-
306         write('Invalid cell content type'), nl,
307         fail.
308
309 validate_destiny_cell_type(Row, Col, Board, Player):-
310         get_matrix_element(Row, Col, Board, Piece),
311         \+ piece_owned_by(Piece, Player), !.
312 %         piece_owned_by(NormalPiece, Player),
313 %         Piece \= NormalPiece, !.
314
315 validate_destiny_cell_type(_, _, _, _):-
```

```
316         write('Invalid destiny cell content type!'), nl,
317         fail.
318
319 validate_piece_owner(Row, Col, Board, Player):-
320         get_matrix_element(Row, Col, Board, Piece),
321         piece_owned_by(Piece, Player), !.
322
323 validate_piece_owner(_, _, _, _):-
324         write('Invalid piece!'), nl,
325         fail.
326
327 invalid_move:-
328         write('Invalid move!'), nl, fail.
```

A.7 utils.pl

```
1  get_list_element(0,[HeadElem|_],HeadElem).
2
3  get_list_element(Pos,[_|OtherElems],Symbol):-
4          Pos > 0,
5          Pos1 is Pos-1,
6          get_list_element(Pos1,OtherElems,Symbol).
7
8  get_matrix_element(0, ElemCol, [ListAtTheHead|_], Elem):-
9          get_list_element(ElemCol, ListAtTheHead, Elem).
10
11 get_matrix_element(ElemRow, ElemCol, [_|RemainingLists], Elem):-
12         ElemRow > 0,
13         ElemRow1 is ElemRow-1,
14         get_matrix_element(ElemRow1, ElemCol, RemainingLists,
               Elem).
15
16 set_matrix_element(0, ElemCol, NewElem, [RowAtTheHead|
       RemainingRows], [NewRowAtTheHead|RemainingRows]):-
17         set_list_element(ElemCol, NewElem, RowAtTheHead,
               NewRowAtTheHead).
18 set_matrix_element(ElemRow, ElemCol, NewElem, [RowAtTheHead|
       RemainingRows], [RowAtTheHead|ResultRemainingRows]):-
19         ElemRow > 0,
20         ElemRow1 is ElemRow-1,
21         set_matrix_element(ElemRow1, ElemCol, NewElem,
               RemainingRows, ResultRemainingRows).
22
23 set_list_element(0, Elem, [_|L], [Elem|L]).
24 set_list_element(I, Elem, [H|L], [H|ResL]):-
25         I > 0,
26         I1 is I-1,
27         set_list_element(I1, Elem, L, ResL).
28
29 initialize_random_seed:-
30         now(Usec), Seed is Usec mod 30269,
31         getrand(random(X, Y, Z, _)),
32         setrand(random(Seed, X, Y, Z)), !.
```