

Megateh

Relatório Final



Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Turma 1 - Grupo Megateh_2

Duarte Pinto Valente - up201504327@fe.up.pt
Luís Cruz - up201303248@fe.up.pt

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, s/n, 4200-465 Porto, Portugal

12 de Novembro de 2017

Resumo

No âmbito da unidade curricular Programação em Lógica foi proposto o desenvolvimento de uma aplicação em Prolog capaz de simular o jogo Megateh. Este trabalho teve em vista a consolidação e aprofundamento dos conceitos da linguagem Prolog expostos nas aulas. Nomeadamente, a manipulação de listas que teve um papel central no desenvolvimento das variadas funcionalidades do programa. Conseguiu-se desenvolver uma aplicação que simula fielmente o ciclo de jogo do Megateh, no qual cada jogador coloca uma peça no tabuleiro alternadamente. Quando o jogo termina, vence o último jogador a colocar uma peça antes de ser alcançado um dos padrões de vitória ou, caso nenhuma condição de vitória tenha sido verificada antes de se esgotarem as peças, o jogo termina em empate. Foi desenvolvida uma interface simples que apresenta uma representação do tabuleiro após cada jogada e permite a introdução da próxima jogada por parte do utilizador. As jogadas efetuadas pelo computador são aleatórias, pelo que não foram implementados diferentes níveis de dificuldade no modo humano contra computador.

Conteúdo

1	Introdução	4
2	O Jogo Megateh	4
2.1	Objetivo	4
2.2	Movimentos	4
2.3	Restrições	4
3	Lógica do Jogo	5
3.1	Representação do Estado do Jogo	5
3.1.1	Representação do estado inicial	5
3.2	Visualização do Tabuleiro	5
3.3	Lista de jogadas Válidas	6
3.4	Execução de Jogadas	6
3.5	Avaliação do Tabuleiro	7
3.6	Final do Jogo	7
3.7	Jogada do Computador	8
4	Interface com o Utilizador	9
5	Conclusões	9
6	Referências	9
7	Código Fonte	10
7.1	board.pl	10
7.2	cli.pl	12
7.3	computer.pl	14
7.4	game.pl	15
7.5	logic.pl	17
7.6	megateh.pl	27
7.7	utils.pl	29

1 Introdução

O presente relatório divide-se em três módulos principais. Primeiramente será efetuada uma breve apresentação sobre o jogo Megateh, abordando os seus objetivos, as jogadas possíveis e as suas restrições. De seguida será abordada a implementação da lógica do jogo, com destaque para a representação do estado do jogo, execução de jogadas e avaliação do tabuleiro. Finalmente, abordaremos as estratégias adotadas para o desenvolvimento de uma interface com o utilizador. Em anexo submetemos o código fonte do projeto.

2 O Jogo Megateh

O Megateh é um jogo de estratégia em tabuleiro quatro por quatro (4x4), semelhante ao jogo do galo, inventado por Mitsuo Yamamoto em 2017. O jogo foi desenvolvido tendo em vista a sua jogabilidade por portadores de deficiências visuais. Para este efeito, usam-se três tipos de peças, lisas de um nível, furadas de um nível e peças de dois níveis com uma face furada e outra lisa. No início de cada jogo estão disponíveis oito peças de cada tipo e o tabuleiro encontra-se vazio. Embora o jogo original permita um número de jogadores entre dois e seis, no presente projeto apenas foi implementada a funcionalidade para dois jogadores.

2.1 Objetivo

O jogo chega a um estado final de vitória quando um jogador, concretiza uma das seguintes situações:

1. Escada de três degraus

Colocação de uma peça completando uma escada de três níveis na direção horizontal ou vertical, independentemente do tipo de peças que a constituem.

2. Quatro em linha

Colocação de uma peça descrevendo uma linha de quatro peças do mesmo nível, furadas ou lisas, com orientação horizontal ou vertical.

3. Quatro em linha na diagonal

Colocação de uma peça de modo a concluir uma linha de quatro peças na diagonal principal do tabuleiro. As peças poderão ter altura variável. No entanto, no topo de cada célula da diagonal terão de estar peças do mesmo tipo.

2.2 Movimentos

Cada jogador pode efetuar os seguintes movimentos:

1. Colocar uma peça não jogada numa célula vazia;
2. Colocar uma peça de um nível no topo de outra peça de um nível;
3. Colocar uma peça de dois níveis no topo de uma peça de um nível;

2.3 Restrições

1. Não é permitida a colocação de mais de duas peças em qualquer célula do tabuleiro.
2. Não é permitido o empilhamento de peças de um nível em peças de dois níveis.

3 Lógica do Jogo

3.1 Representação do Estado do Jogo

Para a representação do tabuleiro de jogo foram usadas listas de listas. As peças jogadas são guardadas em *PieceBoard*, o nível de cada célula é guardado em *Heightboard* e o número de peças por célula é mantido por *NumPiecesBoard*. Esta organização permite efetuar a representação da peça no topo de cada célula e do nível em que esta se encontra.

3.1.1 Representação do estado inicial

```
inicial_board([[empty,empty,empty,empty],  
               [empty,empty,empty,empty],  
               [empty,empty,empty,empty],  
               [empty,empty,empty,empty]  
              ]).
```

4				
3				
2				
1				
	A	B	C	D

Figura 1: Representação do tabuleiro vazio

3.2 Visualização do Tabuleiro

O tabuleiro pode ser visualizado após cada jogada, tanto do utilizador como do computador. As linhas do tabuleiro estão numeradas ascendentemente e cada coluna é identificada por uma letra, facilitando a indicação da posição desejada para a peça a jogar.

4	1f	2df	2df	2df
3				
2	2df			1h
1	2df	1h	2df	
	A	B	C	D

Figura 2: Representação de estado intermédio do jogo

3.3 Lista de jogadas Válidas

A validação de cada jogada é efetuada pela função *validate_move*.

```
validate_move(SrcRow,SrcCol,NewPiece,PiecesBoard,NumPiecesBoard):-  
    validate_height(SrcRow,SrcCol,NewPiece,PiecesBoard),  
    validate_number_pieces(SrcRow,SrcCol,NumPiecesBoard).
```

A função *validate_height* é responsável por garantir que não podem ser empilhadas peças de um nível em cima de peças duplas. A função *validate_number_pieces* avalia a segunda restrição, verificando se a célula selecionada tem menos de duas peças.

3.4 Execução de Jogadas

A cada jogada é pedida a posição, letra da coluna seguido do número da linha, e o tipo de peça que se deseja colocar, f, h, df, dh.

```
First Player turn to play!  
Flat: 8      Holed: 8      2 Leveld: 8  
  
Enter the column and row (ex:a1) of the piece to be placed followed by <CR>:  
|: b2.  
Please enter the type of piece (ex: f, h df, dh) you wish to place:  
|: df.
```

Figura 3: Interface que recebe a jogada do utilizador

Internamente, é executada a instrução *make_move* que irá, após garantir a validade da jogada, mover a peça para a posição do tabuleiro indicada através de *move_piece*.

```
move_piece(SrcRow,SrcCol,NewPiece,PiecesBoard,HeightBoard,NumPiecesBoard,ModifiedPieceBoard,  
ModifiedHeightBoard,ModifiedNumPiecesBoard):-  
    %get the height from the new piece to be placed on board  
    get_cell_symbol(NewPiece,Symbol),  
    get_piece_height(Symbol,NewPieceHeight),  
  
    %get the height from the Source cell  
    get_matrix_element(SrcRow,SrcCol,HeightBoard,SrcHeight),  
    SrcHeight1 is SrcHeight + NewPieceHeight,  
  
    %get the number of pieces in that cell  
    get_matrix_element(SrcRow,SrcCol,NumPiecesBoard,SrcNumPieces),  
    SrcNumPieces1 is SrcNumPieces + 1,  
  
    %update the pieces board with the new top piece  
    set_matrix_element(SrcRow,SrcCol,NewPiece,PiecesBoard,ModifiedPieceBoard),  
  
    %update the height board after placing the new piece  
    set_matrix_element(SrcRow,SrcCol,SrcHeight1,HeightBoard,ModifiedHeightBoard),  
  
    %update the numPieces board after placing the new piece  
    set_matrix_element(SrcRow,SrcCol,SrcNumPieces1,NumPiecesBoard,ModifiedNumPiecesBoard).
```

3.5 Avaliação do Tabuleiro

Neste projeto são apenas efetuadas avaliações do tabuleiro para determinar a vitória de um dos jogadores. Após cada jogada, o predicado *determine_winner* é responsável por procurar no tabuleiro a ocorrência de uma condição de vitória. Para tal recorre-se aos seguintes predicados *three_step_stair*, *four_in_a_row* e *four_in_a_row_diagonal*.

```
four_in_a_row(Row,_,HeightBoard):-
    \+ four_in_a_row_horizontal(Row,HeightBoard).

four_in_a_row(_,Col,HeightBoard):-
    \+ four_in_a_row_vertical(Col,HeightBoard).

four_in_a_row_diagonal(_,_,Board):-
    four_in_a_row_descendent_diagonal(0,0,Board).

four_in_a_row_diagonal(_,_,Board):-
    four_in_a_row_ascendant_diagonal(3,0,Board).

three_step_stair(Row,Col,HeightBoard):-
    Col <= 1,
    Row \= 3,
    findall(Value, matrix(HeightBoard,Row,_,Value),ResultantHeightRow),
    findall(Value, matrix(HeightBoard,_,Col,Value),ResultantHeightCol),
    %If there is not a horizontal or vertical stair, continue to other values
    \+ check_horizontal_tree_step(Row,Col,ResultantHeightRow),
    \+ check_vertical_tree_step(Row,Col,ResultantHeightCol),
    \+ check_horizontal_reverse_tree_step(Row,Col,ResultantHeightRow),
    \+ check_vertical_reverse_tree_step(Row,Col,ResultantHeightCol),

    Col1 is Col + 1,
    three_step_stair(Row,Col1,HeightBoard).
```

3.6 Final do Jogo

O jogo termina com a vitória do jogador cuja jogada tenha verificado uma das condições de vitória. Caso se esgote o número de peças disponíveis sem a condição anterior ocorrer, o jogo terminará em empate.

```
determine_draw(Game):-
    \+determine_draw_Flat(Game),
    \+determine_draw_Holed(Game),
    \+determine_draw_doubleLevel(Game),
    display_game_draw,
    !.

determine_draw_Flat(Game):-
    get_list_element(6,Game,NumFlatPieces),!,
    NumFlatPieces \= 0.

determine_draw_Holed(Game):-
    get_list_element(7,Game,NumHoledPieces),!,
    NumHoledPieces \= 0.

determine_draw_doubleLevel(Game):-
    get_list_element(8,Game,NumDoubleLevelPieces),!,
    NumDoubleLevelPieces \= 0.
```

There is a three step stair!
Second Player Won!

4		3df	2df	1f
3				
2				
1	2h	1f	2df	
	A	B	C	D

Figura 4: Estado final do jogo

3.7 Jogada do Computador

Não foram implementados diferentes modos de dificuldade de jogo. O computador executa jogadas aleatoriamente.

```
computer_play(Game,ModifiedGame):-
    repeat,
    get_board(Game,PieceBoard),
    display_megateh_board(0,Game,PieceBoard,4),
    display_turn_info(Game),nl,
    random(0,4,SrcRow),
    write('Random Row: '), write(SrcRow),nl,
    random(0,4,SrcCol),
    write('Random Col: '),write(SrcCol),nl,
    random_piece(NewPiece),
    write('NewPiece is: '),write(NewPiece),nl,
    make_move(SrcRow,SrcCol,NewPiece,Game,ModifiedGame),
    !.

random_piece(Piece):-
    random(0,3,Num),
    choosePiece(Num,Piece).

choosePiece(Num,f):-
    Num == 0.

choosePiece(Num,h):-
    Num == 1.

choosePiece(Num,df):-
    Num == 2.

choosePiece(Num,dh):-
    Num == 3.
```


4 Interface com o Utilizador

No início do programa é solicitada ao utilizador a escolha de um modo de jogo de três possíveis. No modo de jogo humano contra humano, os dois jogadores deverão introduzir, na sua vez de jogar, o tipo de peça e das coordenadas onde a colocar. O modo humano contra computador apenas necessita da introdução de uma jogada por ciclo de jogo, uma vez que o computador irá efetuar uma jogada aleatória. No último modo, são efetuadas jogadas aleatórias consecutivas até se alcançar uma condição de terminação. Após cada jogada do utilizador ou do computador, o estado atual do tabuleiro, e o número de peças restantes, são impressos de modo a registar a evolução do jogo até ao seu término.

```
Megateh game
[1] Human vs. Human
[2] Human vs. Computer
[3] Computer vs. Computer
Enter game mode number:
|: ■
```

Figura 5: Menu inicial

5 Conclusões

No decorrer do desenvolvimento do projeto foram aplicados e aprofundados os conceitos de programação em lógica lecionados nas aulas. O resultado foi uma aplicação simples de simulação do jogo de tabuleiro Megateh para zero, um ou dois utilizadores. O programa desenvolvido respeita todas as regras e restrições do jogo original, implementa uma interface capaz de representar informação tridimensional num meio bidimensional e notifica o utilizador de todas as diferenças no estado do jogo a cada jogada. Foi também implementada uma solução para simular um número fixo de peças disponíveis, tendo em vista a aproximação desta versão do jogo à original. Em retrospectiva é de assinalar a significativa complexidade do trabalho e as várias adversidades ultrapassadas pelos elementos do grupo no desenvolvimento do mesmo.

6 Referências

Referências

- [1] Sterling, Leon *The Art of Prolog*, The MIT Press 2nd edition, 2000.
- [2] Megateh, <http://www.logygames.com/megateh/index-en.html> 10 11 2017
- [3] <https://stackoverflow.com/questions/34949724/prolog-iterate-through-matrix> 10 11 2017

7 Código Fonte

7.1 board.pl

```
1  %symbols
2  symbol(f,'f ').
3  symbol(h,'h ').
4  symbol(df,'df').
5  symbol(dh,'dh').
6  symbol(empty,' ').
7
8  %gets the cells symbols
9  get_cell_symbol(' ',empty).
10 get_cell_symbol('f',flat).
11 get_cell_symbol('h',holed).
12 get_cell_symbol('df',dF).
13 get_cell_symbol('dh',dH).
14
15 get_cell_display_symbol(' ',empty).
16 get_cell_display_symbol('f',flat).
17 get_cell_display_symbol('h',holed).
18 get_cell_display_symbol('df',dF).
19 get_cell_display_symbol('dh',dH).
20
21 get_piece_symbol(' ',empty).
22 get_piece_symbol('f',flat).
23 get_piece_symbol('h',holed).
24 get_piece_symbol('df',flat).
25 get_piece_symbol('dh',holed).
26
27 %gets the cells height
28 get_piece_height(empty,0).
29 get_piece_height(flat,1).
30 get_piece_height(holed,1).
31 get_piece_height(dF,2).
32 get_piece_height(dH,2).
33
34
35 matrix(Matrix,I,J,Value):-
36     nth0(I,Matrix,Row),
37     nth0(J,Row,Value).
38
39
40 %get the board cell
41 %decrements the row to get to the row we want
42 get_board_cell(0,Col,[HeadList|_],Symbol):-
43     get_list_element(Col,HeadList,Symbol).
44
45 get_board_cell(Row,Col,[_|TailList], Symbol):-
46     Row > 0,
47     Row1 is Row - 1,
48     get_board_cell(Row1,Col,TailList,Symbol).
```

```

49
50
51 %inicial board-----
52 inicial_board([[empty,empty,empty,empty],
53                [empty,empty,empty,empty],
54                [empty,empty,empty,empty],
55                [empty,empty,empty,empty]
56                ]).
57
58
59 %inicial height list-----
60 inicial_height([[0,0,0,0],
61                [0,0,0,0],
62                [0,0,0,0],
63                [0,0,0,0]
64                ]).
65
66
67 %inicial number pieces list-----
68 inicial_number_pieces([[0,0,0,0],
69                        [0,0,0,0],
70                        [0,0,0,0],
71                        [0,0,0,0]
72                        ]).
73
74 %display of board-----
75 display_megateh_board(4,_,[],_-):-
76     write(' '), write('
↪ -----'), nl, nl,
77     write(' '), write('      A      B      C      D
↪ '), nl.
78
79 display_megateh_board(Row,Game,[Head|Tail],R):-
80     get_list_element(1,Game,HeightBoard),
81     write(' '), write('
↪ -----'),nl,
82     write(' '), display_empty_line([]),
83     write(' '), write(R), display_line(Row,0,HeightBoard,Head), nl,
84     write(' '), display_empty_line([]),
85     Row1 is Row + 1,
86     R1 is R - 1,
87     display_megateh_board(Row1,Game,Tail,R1),!.
88
89
90
91 %display empty line-----
92 display_empty_line([]):-
93     write(' '), write('||'),write(' '),
94     write('||'), write(' '), write('||'),
95     write(' '), write('||'),write(' '),
96     write('||'), nl.
97

```

```

98
99
100 %display line-----
101 display_line(,4,_,[]) :-
102     write('    |    ').
103
104 display_line(Row,Col,HeightBoard,[H|T]) :-
105     findall(Value, matrix(HeightBoard,Row,Col,Value),Cell),
106     symbol(H,S),
107     get_list_element(0,Cell,CellHeight),
108     CellHeight \= 0,
109     write('    |    '),
110     write(CellHeight),
111     write(S),
112     Col1 is Col + 1,
113     display_line(Row,Col1,HeightBoard,T).
114
115 display_line(Row,Col,HeightBoard,[H|T]) :-
116     findall(Value, matrix(HeightBoard,Row,Col,Value),Cell),
117     symbol(H,S),
118     get_list_element(0,Cell,_),
119     write('    |    '),
120     write(' '),
121     write(S),
122     Col1 is Col + 1,
123     display_line(Row,Col1,HeightBoard,T).

```

7.2 cli.pl

```

1  %the main menu
2  megateh_main_menu(Input):-
3      Input == 1,
4      human_vs_human(Game),
5      game_loop(Game),!.
6
7  megateh_main_menu(Input):-
8      Input == 2,
9      human_vs_computer(Game),
10     game_loop(Game),!.
11
12 megateh_main_menu(Input):-
13     Input \= 3,
14     write('Wrong number try again'),nl,fail.
15
16 megateh_main_menu(Input):-
17     Input == 3,
18     computer_vs_computer(Game),
19     game_loop(Game),!.
20
21 %prints the menu
22 print_menu:-
23     write('          Megateh game'), nl, nl,

```

```

24         write(' [1] Human vs. Human'), nl,
25         write(' [2] Human vs. Computer'), nl,
26         write(' [3] Computer vs. Computer'), nl, nl,
27         write('Enter game mode number:'), nl.
28
29
30     %display the turns information
31     display_turn_info(Game):-
32         get_player_turn(Game,Player),
33         get_player_name(Player,PlayerName),
34
35         get_list_element(6,Game,NumFlatPieces),
36         get_list_element(7,Game,NumHoledPieces),
37         get_list_element(8,Game,NumDoubleLevelPieces),
38
39         nl,write(PlayerName),write(' turn to play! '),nl,
40         write('Flat: '),write(NumFlatPieces),write(' '),
41         write('Holed: '),write(NumHoledPieces),write(' '),
42         write('2 Leveld: '),write(NumDoubleLevelPieces),
43         nl.
44
45     %display the win information
46     display_game_won(Game,ModifiedGame):-
47         get_player_turn(Game,Player),
48         get_player_name(Player,PlayerName),
49         set_winner_value(Game,winner,ModifiedGame),
50         write(PlayerName),
51         write(' Won!'),nl,true,!.
52
53     %display the draw information
54     display_game_draw:-
55         write('No winning condition and no more pieces, its a DRAW!
56         ↵ '),nl,true,
57         !.
58
59     %get the type of piece from the player, to be placed on the board
60     get_new_type_piece(NewPiece):-
61         write('Please enter the type of piece (ex: f, h df, dh) you wish
62         ↵ to place: '),nl,
63         read(NewPiece),
64         get_return_key.
65
66     %get the Coordinates from the player in order to place a new piece on the
67     ↵ board
68     get_new_piece_source_coordinartes(SrcRow,SrcCol):-
69         write('Enter the column and row (ex:a1) of the piece to be placed
70         ↵ followed by <CR>: '),nl,
71         get_coordinates(SrcRow,SrcCol).

```

```

72  get_coordinates(Row,Col):-
73      get_integer(C),
74      get_integer(R),
75      get_return_key,
76      Row is 4-R,
77      Col is C-49.
78
79  %get the value of input
80  get_integer(Input):-
81      get_code(TempInput),
82      Input is TempInput - 48.
83
84  %ignore a key
85  get_return_key:-
86      get_code(_).

```

7.3 computer.pl

```

1  %makes a computer move
2  computer_play(Game,ModifiedGame):-
3      repeat,
4      get_board(Game,PieceBoard),
5      display_megateh_board(0,Game,PieceBoard,4),
6      display_turn_info(Game),nl,
7
8      random(0,4,SrcRow),
9      write('Random Row: '), write(SrcRow),nl,
10     random(0,4,SrcCol),
11     write('Random Col: '),write(SrcCol),nl,
12
13     random_piece(NewPiece),
14     write('NewPiece is: '),write(NewPiece),nl,
15
16     make_move(SrcRow,SrcCol,NewPiece,Game,ModifiedGame),
17     !.
18
19  random_piece(Piece):-
20     random(0,3,Num),
21     choosePiece(Num,Piece).
22
23  choosePiece(Num,f):-
24     Num == 0.
25
26  choosePiece(Num,h):-
27     Num == 1.
28
29  choosePiece(Num,df):-
30     Num == 2.
31
32  choosePiece(Num,dh):-
33     Num == 3.

```

7.4 game.pl

```
1  %Modified Game list Information
2  %Game = [boardGame,HeightBoard,NumPiecesBoard, Winner value, starPlayer,
   ↪ gameMode, num of flat pieces, num of holed pieces, num of double
   ↪ level reversible pieces]
3  %Game = [Board, HeightBoard, NumPicesBoard, noWinner, firstPlayer, hvh,
   ↪ 8, 8, 8]
4
5
6  %board procedures -----
7  get_board([Board|_],Board).
8
9  set_board(Board,Game,ModifiedGame):-
10     set_list_element(0,Board,Game,ModifiedGame).
11  %board procedures -----
12
13
14  %Game Mode Procedures-----
15  human_vs_human(Game):-
16     %unifies inicial_board with Board
17     inicial_board(PieceBoard),
18
19     %unifies inicial_height board with HeightBoard
20     inicial_height(HeightBoard),
21
22     %unifies inicial_number_pieces with NumPiecesBoard
23     inicial_number_pieces(NumPiecesBoard),
24
25     Game = [PieceBoard, HeightBoard, NumPiecesBoard, noWinner,
   ↪ firstPlayer, hvh,8 ,8, 8].
26
27  human_vs_computer(Game):-
28     %unifies inicial_board with Board
29     inicial_board(PieceBoard),
30
31     %unifies inicial_height board with HeightBoard
32     inicial_height(HeightBoard),
33
34     %unifies inicial_number_pieces with NumPiecesBoard
35     inicial_number_pieces(NumPiecesBoard),
36
37     Game = [PieceBoard, HeightBoard, NumPiecesBoard, noWinner,
   ↪ firstPlayer, hvc,8 ,8, 8].
38
39  computer_vs_computer(Game):-
40     %unifies inicial_board with Board
41     inicial_board(PieceBoard),
42
43     %unifies inicial_height board with HeightBoard
44     inicial_height(HeightBoard),
45
```

```

46         %unifies inicial_number_pieces with NumPiecesBoard
47         inicial_number_pieces(NumPiecesBoard),
48
49         Game = [PieceBoard, HeightBoard, NumPiecesBoard, noWinner,
50               ↪ firstPlayer, cvc,8 ,8, 8].
51
52 %get the game mode
53 get_game_mode(Game,Mode):-
54     get_list_element(3,Game,Mode).
55 %Game Mode Procedures-----
56 %Winner Procedures-----
57 get_winner_value(Game,WinnerValue):-
58     get_list_element(3,Game,WinnerValue).
59
60 set_winner_value(Game,WinnerValue,ModifiedGame):-
61     set_list_element(3,WinnerValue,Game,ModifiedGame).
62 %Winner Procedures-----
63
64 %Player Procedures -----
65 %gets what player turn
66 get_player_turn(Game,Player):-
67     get_list_element(4,Game,Player).
68
69 %sets the player turn
70 set_player_turn(Player,Game,ModifiedPlayerGame):-
71     set_list_element(4,Player,Game,ModifiedPlayerGame).
72
73 %change the players turn
74 change_player_turn(Temporary_Game,Modified_Game):-
75     get_player_turn(Temporary_Game,Old_Turn),
76     Old_Turn == firstPlayer,
77     NewPlayer = secondPlayer,
78     set_player_turn(NewPlayer,Temporary_Game,Modified_Game).
79
80 change_player_turn(Temporary_Game,Modified_Game):-
81     get_player_turn(Temporary_Game,Old_Turn),
82     Old_Turn == secondPlayer,
83     NewPlayer = firstPlayer,
84     set_player_turn(NewPlayer,Temporary_Game,Modified_Game).
85
86 %decrement the piece played from the pieces available
87 decrement_piece_number(Game,PieceType,ModifiedGame):-
88     PieceType == 'f',
89     get_list_element(6,Game,NumPieces),
90     NumPieces > 0,
91     NumPieces1 is NumPieces - 1,
92     set_list_element(6,NumPieces1,Game,ModifiedGame).
93
94 decrement_piece_number(Game,PieceType,ModifiedGame):-
95     PieceType == 'h',
96     get_list_element(7,Game,NumPieces),

```



```

97     NumPieces > 0,
98     NumPieces1 is NumPieces - 1,
99     set_list_element(7,NumPieces1,Game,ModifiedGame).
100
101 decrement_piece_number(Game,PieceType,ModifiedGame):-
102     PieceType == 'df',
103     get_list_element(8,Game,NumPieces),
104     NumPieces > 0,
105     NumPieces1 is NumPieces - 1,
106     set_list_element(8,NumPieces1,Game,ModifiedGame).
107
108 decrement_piece_number(Game,PieceType,ModifiedGame):-
109     PieceType == 'dh',
110     get_list_element(8,Game,NumPieces),
111     NumPieces > 0,
112     NumPieces1 is NumPieces - 1,
113     set_list_element(6,NumPieces1,Game,ModifiedGame).

```

7.5 logic.pl

```

1  %makes a move - this rule is in case the game has a winning condition
2  make_move(SrcRow,SrcCol,NewPiece,Game,ModifiedPlayerGame):-
3      %get the PieceBoard
4      get_list_element(0,Game,PieceBoard),
5      %get the HeightBoard
6      get_list_element(1,Game,HeightBoard),
7      %get the NumPiecesBoard
8      get_list_element(2,Game,NumPiecesBoard),
9
10     %validate the piece placement
11     validate_move(SrcRow, SrcCol, NewPiece, PieceBoard,
12     ↪ NumPiecesBoard),
13
14     %place the piece on the board
15     move_piece(SrcRow, SrcCol ,NewPiece, PieceBoard, HeightBoard,
16     ↪ NumPiecesBoard, ModifiedPieceBoard, ModifiedHeightBoard,
17     ↪ ModifiedNumPiecesBoard),
18
19     %decrement the new piece from the pieces available
20     decrement_piece_number(Game, NewPiece, TempModifiedGame),
21
22     %update the Piece Board on the Game
23     set_list_element(0, ModifiedPieceBoard, TempModifiedGame,
24     ↪ ModifiedGameWithPieces),
25     %update the Height Board on the Game
26     set_list_element(1, ModifiedHeightBoard, ModifiedGameWithPieces,
27     ↪ ModifiedGameWithHeight),
28     %update the NumPieces Board on the Game
29     set_list_element(2, ModifiedNumPiecesBoard,
30     ↪ ModifiedGameWithHeight, ModifiedGame),
31
32     %determin if there is a winning condition on this turn

```

```

27         %if it fails continue playing
28
29         ↪ determine_winner(0,0,ModifiedHeightBoard,ModifiedPieceBoard,ModifiedGame,ModifiedWin
30
31         %change the player
32         change_player_turn(ModifiedWinnerGame,ModifiedPlayerGame).
33
34     %makes a move - this rule is in case the game does not have a winning
35     ↪ condition
36     make_move(SrcRow,SrcCol,NewPiece,Game,ModifiedPlayerGame):-
37         %get the PieceBoard
38         get_list_element(0,Game,PieceBoard),
39         %get the HeightBoard
40         get_list_element(1,Game,HeightBoard),
41         %get the NumPiecesBoard
42         get_list_element(2,Game,NumPiecesBoard),
43
44         %validate the piece placement
45         validate_move(SrcRow, SrcCol, NewPiece, PieceBoard,
46         ↪ NumPiecesBoard),
47
48         %place the piece on the board
49         move_piece(SrcRow, SrcCol ,NewPiece, PieceBoard, HeightBoard,
50         ↪ NumPiecesBoard, ModifiedPieceBoard, ModifiedHeightBoard,
51         ↪ ModifiedNumPiecesBoard),
52
53         %decrement the new piece from the pieces available
54         decrement_piece_number(Game, NewPiece, TempModifiedGame),
55
56         %update the Piece Board on the Game
57         set_list_element(0, ModifiedPieceBoard, TempModifiedGame,
58         ↪ ModifiedGameWithPieces),
59         %update the Height Board on the Game
60         set_list_element(1, ModifiedHeightBoard, ModifiedGameWithPieces,
61         ↪ ModifiedGameWithHeight),
62         %update the NumPieces Board on the Game
63         set_list_element(2, ModifiedNumPiecesBoard,
64         ↪ ModifiedGameWithHeight, ModifiedGame),
65
66         %change the player
67         change_player_turn(ModifiedGame,ModifiedPlayerGame).
68
69     %places the piece on the board
70     move_piece(SrcRow,SrcCol,NewPiece,PiecesBoard,HeightBoard,NumPiecesBoard,ModifiedPieceBoard,Mod
71         %get the height from the new piece to be placed on board
72         get_cell_symbol(NewPiece,Symbol),
73         get_piece_height(Symbol,NewPieceHeight),
74
75         %get the height from the Source cell
76         get_matrix_element(SrcRow,SrcCol,HeightBoard,SrcHeight),
77         SrcHeight1 is SrcHeight + NewPieceHeight,

```

```

71
72     %get the number of pieces in that cell
73     get_matrix_element(SrcRow,SrcCol,NumPiecesBoard,SrcNumPieces),
74     SrcNumPieces1 is SrcNumPieces + 1,
75
76     %update the pieces board with the new top piece
77
78     ⇨ set_matrix_element(SrcRow,SrcCol,NewPiece,PiecesBoard,ModifiedPieceBoard),
79
80     %update the height board after placing the new piece
81
82     ⇨ set_matrix_element(SrcRow,SrcCol,SrcHeight1,HeightBoard,ModifiedHeightBoard),
83
84     %update the numPieces board after placing the new piece
85
86     ⇨ set_matrix_element(SrcRow,SrcCol,SrcNumPieces1,NumPiecesBoard,ModifiedNumPiecesBoard),
87
88 %validate the piece placement according to restrictions
89 validate_move(SrcRow,SrcCol,NewPiece,PiecesBoard,NumPiecesBoard):-
90     validate_high(SrcRow,SrcCol,NewPiece,PiecesBoard),
91     validate_number_pieces(SrcRow,SrcCol,NumPiecesBoard).
92
93 %Restriction1-
94 %It is not permitted to place single level pieces on top of two level
95 ⇨ pieces
96 validate_high(SrcRow,SrcCol,NewPiece,PiecesBoard):-
97     %check if the piece is double level
98     NewPiece == 'f',
99
100     %the the type of piece on the cell
101     get_matrix_element(SrcRow,SrcCol,PiecesBoard,SrcPiece),!,
102
103     SrcPiece \= 'df',
104     SrcPiece \= 'dh'.
105
106 validate_high(SrcRow,SrcCol,NewPiece,PiecesBoard):-
107     %check if the piece is double level
108     NewPiece == 'f',
109
110     %the the type of piece on the cell
111     get_matrix_element(SrcRow,SrcCol,PiecesBoard,SrcPiece),!,
112
113     SrcPiece == 'df',
114     write('Invalid move! Cant place single-level on top of
115     ⇨ double-level. '),nl,
116     fail.
117
118 validate_high(SrcRow,SrcCol,NewPiece,PiecesBoard):-
119     %check if the piece is double level
120     NewPiece == 'f',

```

```

118     %the the type of piece on the cell
119     get_matrix_element(SrcRow,SrcCol,PiecesBoard,SrcPiece),!,
120
121     SrcPiece == 'dh',
122     write('Invalid move! Cant place single-level on top of
123           ↳ double-level. '),nl,
124     fail.
125
126 validate_high(SrcRow,SrcCol,NewPiece,PiecesBoard):-
127     %check if the piece is double level
128     NewPiece == 'h',
129
130     %the the type of piece on the cell
131     get_matrix_element(SrcRow,SrcCol,PiecesBoard,SrcPiece),!,
132
133     SrcPiece \= 'df',
134     SrcPiece \= 'dh'.
135
136 validate_high(SrcRow,SrcCol,NewPiece,PiecesBoard):-
137     %check if the piece is double level
138     NewPiece == 'h',
139
140     %the the type of piece on the cell
141     get_matrix_element(SrcRow,SrcCol,PiecesBoard,SrcPiece),!,
142
143     SrcPiece == 'df',
144     write('Invalid move! Cant place single-level on top of
145           ↳ double-level. '),nl,
146     fail.
147
148 validate_high(SrcRow,SrcCol,NewPiece,PiecesBoard):-
149     %check if the piece is double level
150     NewPiece == 'h',
151
152     %the the type of piece on the cell
153     get_matrix_element(SrcRow,SrcCol,PiecesBoard,SrcPiece),!,
154
155     SrcPiece == 'dh',
156     write('Invalid move! Cant place single-level on top of
157           ↳ double-level. '),nl,
158     fail.
159
160 validate_high(SrcRow,SrcCol,NewPiece,PiecesBoard):-
161     %check if the piece is double level
162     NewPiece == 'df',
163
164     %the the type of piece on the cell
165     get_matrix_element(SrcRow,SrcCol,PiecesBoard,SrcPiece),!,
166
167     SrcPiece \= 'df',
168     SrcPiece \= 'dh'.

```

```

167 validate_high(SrcRow,SrcCol,NewPiece,PiecesBoard):-
168     %check if the piece is double level
169     NewPiece == 'dh',
170
171     %the the type of piece on the cell
172     get_matrix_element(SrcRow,SrcCol,PiecesBoard,SrcPiece),!,
173
174     SrcPiece \= 'df',
175     SrcPiece \= 'dh'.
176
177
178 %Restriction2-
179 %No board cell should have more than two pieces.
180 validate_number_pieces(SrcRow,SrcCol,NumPieceBoard):-
181     %get the number of pieces of the board cell
182     get_matrix_element(SrcRow,SrcCol,NumPieceBoard,SrcNumPieces),!,
183     SrcNumPieces < 2.
184
185 validate_number_pieces(_,_,_):-
186     write('Invalid move! Cant place more pieces there. '),nl,
187     fail.
188
189 %determine if there is a draw
190 determine_draw(Game):-
191     \+determine_draw_Flat(Game),
192     \+determine_draw_Holed(Game),
193     \+determine_draw_doubleLevel(Game),
194     display_game_draw,
195     !.
196
197 determine_draw_Flat(Game):-
198     get_list_element(6,Game,NumFlatPieces),!,
199     NumFlatPieces \= 0.
200
201 determine_draw_Holed(Game):-
202     get_list_element(7,Game,NumHoledPieces),!,
203     NumHoledPieces \= 0.
204
205 determine_draw_doubleLevel(Game):-
206     get_list_element(8,Game,NumDoubleLevelPieces),!,
207     NumDoubleLevelPieces \= 0.
208
209 %determine the winner
210 determine_winner(Row,Col,HeightBoard,_,Game,ModifiedGame):-
211     \+ three_step_stair(Row,Col,HeightBoard),
212     write('There is a three step stair!'),nl,
213     display_game_won(Game,ModifiedGame),
214     !.
215
216 determine_winner(Row,Col,HeightBoard,_,Game,ModifiedGame):-
217     four_in_a_row(Row,Col,HeightBoard),
218     write('There is a four in a row of the same height!'),nl,

```

```

219         display_game_won(Game,ModifiedGame),
220         !.
221
222     determine_winner(Row,Col,_,ModifiedPieceBoard,Game,ModifiedGame):-
223         four_in_a_row_diagonal(Row,Col,ModifiedPieceBoard),
224         write('There is a diagonal four in a row with the same piece
↳ type!'),nl,
225         display_game_won(Game,ModifiedGame),
226         !.
227
228
229
230     %winning conditions
↳ -----
231     three_step_stair(3,2,_):-
232         true.
233
234     three_step_stair(Row,Col,HeightBoard):-
235         Col <= 1,
236         Row \= 3,
237         findall(Value,
↳ matrix(HeightBoard,Row,_,Value),ResultantHeightRow),
238         findall(Value,
↳ matrix(HeightBoard,_,Col,Value),ResultantHeightCol),
239
240         %If there is not a horizontal or vertical stair, continue to other
↳ values
241         \+ check_horizontal_tree_step(Row,Col,ResultantHeightRow),
242         \+ check_vertical_tree_step(Row,Col,ResultantHeightCol),
243         \+ check_horizontal_reverse_tree_step(Row,Col,ResultantHeightRow),
244         \+ check_vertical_reverse_tree_step(Row,Col,ResultantHeightCol),
245
246         Col1 is Col + 1,
247         three_step_stair(Row,Col1,HeightBoard).
248
249     three_step_stair(Row,Col,HeightBoard):-
250         Col = 2,
251         Row < 2,
252
253         findall(Value,
↳ matrix(HeightBoard,_,Col,Value),ResultantHeightCol),
254         \+ check_vertical_tree_step(Row,Col,ResultantHeightCol),
255         \+ check_vertical_reverse_tree_step(Row,Col,ResultantHeightCol),
256         Col1 is Col + 1,
257         three_step_stair(Row,Col1,HeightBoard).
258
259     three_step_stair(Row,Col,HeightBoard):-
260         Row == 2,
261         Col < 2,
262         findall(Value,
↳ matrix(HeightBoard,Row,_,Value),ResultantHeightRow),
263

```

```

264     \+ check_horizontal_tree_step(Row,Col,ResultantHeightRow),
265     \+ check_horizontal_reverse_tree_step(Row,Col,ResultantHeightRow),
266     Col1 is Col + 1,
267     three_step_stair(Row,Col1,HeightBoard).
268
269 three_step_stair(Row,Col,HeightBoard):-
270     Col == 3 ,
271
272     findall(Value,
273         ↪ matrix(HeightBoard,_,Col,Value),ResultantHeightCol),
274
275     \+ check_vertical_tree_step(Row,Col,ResultantHeightCol),
276     \+ check_vertical_reverse_tree_step(Row,Col,ResultantHeightCol),
277
278     Row1 is Row + 1,!,
279     three_step_stair(Row1,0,HeightBoard).
280
281 three_step_stair(Row,Col,HeightBoard):-
282     Row == 2,
283     Col == 2,
284     Row1 is Row + 1,
285     three_step_stair(Row1,0,HeightBoard).
286
287 three_step_stair(Row,Col,HeightBoard):-
288     Row == 3,
289     findall(Value,
290         ↪ matrix(HeightBoard,Row,_,Value),ResultantHeightRow),
291
292     \+ check_horizontal_tree_step(Row,Col,ResultantHeightRow),
293     \+ check_horizontal_reverse_tree_step(Row,Col,ResultantHeightRow),
294
295     Col1 is Col + 1,
296     three_step_stair(Row,Col1,HeightBoard).
297
298 %check if, in a list that represents a row of the board, there is a
299 ↪ horizontal three step stair
300 check_horizontal_reverse_tree_step(_,Col,ResultantHeightRow):-
301     %get the first element of the row
302     get_list_element(Col,ResultantHeightRow,RowHeightValue1),!,
303     RowHeightValue1 == 3,
304     %get the second element of the row
305     Col1 is Col + 1,
306     get_list_element(Col1,ResultantHeightRow,RowHeightValue2),!,
307     RowHeightValue2 == 2,
308     %get the third element of the row
309     Col2 is Col1 + 1,
310     get_list_element(Col2,ResultantHeightRow,RowHeightValue3),!,
311     RowHeightValue3 == 1.
312
313 %check if, in a list that represents a row of the board, there is a
314 ↪ vertical three step stair
315 check_vertical_reverse_tree_step(Row,_,ResultantHeightCol):-

```

```

312         %get the first element of the col
313         get_list_element(Row,ResultantHeightCol,ColHeightValue1),!,
314         ColHeightValue1 == 3,
315         %get the secound element of the column
316         Row1 is Row + 1,
317         get_list_element(Row1,ResultantHeightCol,ColHeightValue2),!,
318         ColHeightValue2 == 2,
319         %get the third element of the column
320         Row2 is Row1 + 1,
321         get_list_element(Row2,ResultantHeightCol,ColHeightValue3),!,
322         ColHeightValue3 == 1.
323
324 %check if, in a list that represents a row of the board, there is a
    ↪ horizontal three step stair
325 check_horizontal_tree_step(_,Col,ResultantHeightRow):-
326     %get the first element of the row
327     get_list_element(Col,ResultantHeightRow,RowHeightValue1),!,
328     RowHeightValue1 == 1,
329     %get the secound element of the row
330     Col1 is Col + 1,
331     get_list_element(Col1,ResultantHeightRow,RowHeightValue2),!,
332     RowHeightValue2 == 2,
333     %get the third element of the row
334     Col2 is Col1 + 1,
335     get_list_element(Col2,ResultantHeightRow,RowHeightValue3),!,
336     RowHeightValue3 == 3.
337
338
339 %check if, in a list that represents a column of the board, there is a
    ↪ vertical three step stair
340 check_vertical_tree_step(Row,_,ResultantHeightCol):-
341     %get the first element of the col
342     get_list_element(Row,ResultantHeightCol,ColHeightValue1),!,
343     ColHeightValue1 == 1,
344     %get the secound element of the column
345     Row1 is Row + 1,
346     get_list_element(Row1,ResultantHeightCol,ColHeightValue2),!,
347     ColHeightValue2 == 2,
348     %get the third element of the column
349     Row2 is Row1 + 1,
350     get_list_element(Row2,ResultantHeightCol,ColHeightValue3),!,
351     ColHeightValue3 == 3.
352
353
354 %goes through the board diagonals and check for a four-in-a-row of the
    ↪ same piece type
355 four_in_a_row_diagonal(_,_,Board):-
356     four_in_a_row_descendent_diagonal(0,0,Board).
357
358 four_in_a_row_diagonal(_,_,Board):-
359     four_in_a_row_ascendant_diagonal(3,0,Board).
360

```



```

361 four_in_a_row_descendent_diagonal(3,3,_).
362
363 four_in_a_row_descendent_diagonal(Row,Col,PiecesBoard):-
364     findall(Value, matrix(PiecesBoard,Row,Col,Value),DiagonalPiece1),
365
366     get_list_element(0,DiagonalPiece1,Piece1),
367     get_piece_symbol(Piece1,Symbol1),
368     Piece1 \= empty,
369
370     Row1 is Row + 1,
371     Col1 is Col + 1,
372
373     findall(Value,
374         ↪ matrix(PiecesBoard,Row1,Col1,Value),DiagonalPiece2),
375
376     get_list_element(0,DiagonalPiece2,Piece2),
377     get_piece_symbol(Piece2,Symbol2),
378
379     Symbol1 == Symbol2,
380
381     four_in_a_row_descendent_diagonal(Row1,Col1,PiecesBoard).
382
383 four_in_a_row_ascendant_diagonal(0,3,_).
384
385 four_in_a_row_ascendant_diagonal(Row,Col,PiecesBoard):-
386     findall(Value, matrix(PiecesBoard,Row,Col,Value),DiagonalPiece1),
387
388     get_list_element(0,DiagonalPiece1,Piece1),
389     get_piece_symbol(Piece1,Symbol1),
390     Piece1 \= empty,
391
392     Row1 is Row - 1,
393     Col1 is Col + 1,
394
395     findall(Value,
396         ↪ matrix(PiecesBoard,Row1,Col1,Value),DiagonalPiece2),
397
398     get_list_element(0,DiagonalPiece2,Piece2),
399     get_piece_symbol(Piece2,Symbol2),
400
401     Symbol1 == Symbol2,
402
403     four_in_a_row_ascendant_diagonal(Row1,Col1,PiecesBoard).
404
405 %check if where is a vertical or horizontal four-in-a-row
406 four_in_a_row(Row,_,HeightBoard):-
407     %if it failed that means that there is a horizontal four in a row
408     ↪ four_in_a_row_horizontal(Row,HeightBoard).
409
410 four_in_a_row(_,Col,HeightBoard):-
411     %if it failed that means that there is a vertical four in a row
412     ↪ four_in_a_row_vertical(Col,HeightBoard).

```

```

411
412
413 %check if there is a horizontal four-in-a-row
414 %all peices must have the same height
415 four_in_a_row_horizontal(4,_):-
416     true.
417
418 four_in_a_row_horizontal(Row,HeightBoard):-
419     findall(Value,
420         ↪ matrix(HeightBoard,Row,_,Value),ResultantHeightRow),
421
422     %if he does not succeed then continue seaching
423     ↪+ go_through_row(0,ResultantHeightRow),
424     Row1 is Row + 1,
425     Row1 <= 4,
426     four_in_a_row_horizontal(Row1,HeightBoard).
427
428 %check if there is a vertical four-in-a-row
429 %all peices must have the same height
430 four_in_a_row_vertical(4,_):-
431     true.
432
433 four_in_a_row_vertical(Col,HeightBoard):-
434     findall(Value,
435         ↪ matrix(HeightBoard,_,Col,Value),ResultantHeightCol),
436
437     %if he does not succeed then continue seaching
438     ↪+ go_through_row(0,ResultantHeightCol),
439     Col1 is Col + 1,
440     Col1 <= 4,
441     four_in_a_row_vertical(Col1,HeightBoard).
442
443 %goes through the list representing a row or col
444 go_through_row(3,_).
445
446 go_through_row(Col,Row):-
447     %get the fist element of the row
448     get_list_element(Col,Row,ResultantHeightValue1),
449     ResultantHeightValue1 > 0,
450     Col1 is Col + 1,
451     Col1 <= 3,
452     %get the consecutive element of the row
453     get_list_element(Col1,Row,ResultantHeightValue2),!,
454     ResultantHeightValue2 > 0,
455     % see if both have the same height
456     ResultantHeightValue1 == ResultantHeightValue2,
457     go_through_row(Col1,Row).

```

7.6 megateh.pl

```
1  %includes
2  :- use_module(library(random)).
3  :- use_module(library(system)).
4  :-use_module(library(lists)).
5  :-use_module(library(det)).
6  :- include('board.pl').
7  :- include('cli.pl').
8  :- include('computer.pl').
9  :- include('game.pl').
10 :- include('logic.pl').
11 :- include('utils.pl').
12
13 %game
14 megateh:-
15     initialize_random_seed,
16     print_menu,
17     repeat,
18     read(Input),
19     get_return_key,
20     megateh_main_menu(Input),!.
21
22 %players
23 player(firstPlayer).
24 player(secondPlayer).
25
26 %get the players names
27 get_player_name(firstPlayer,'First Player').
28 get_player_name(secondPlayer,'Second Player').
29
30
31 %The main game loop for human_vs_human gameplay
32 game_loop(Game):-
33     get_list_element(5,Game,Mode),
34     Mode = hvh,
35
36     get_winner_value(Game,WinnerValue),
37     WinnerValue \= winner,
38
39     \+ determine_draw(Game),
40
41     human_play(Game,ModifiedGame),
42     game_loop(ModifiedGame).
43
44 %The main game loop for human_vs_computer gameplay
45 game_loop(Game):-
46     get_list_element(5,Game,Mode),
47     Mode = hvc,
48
49     get_winner_value(Game,WinnerValue),
50     WinnerValue \= winner,
```

```

51
52     \+ determine_draw(Game),
53
54     get_player_turn(Game,Player),
55     Player == firstPlayer,
56
57     human_play(Game,ModifiedGame),
58
59     game_loop(ModifiedGame).
60
61 game_loop(Game):-
62     get_list_element(5,Game,Mode),
63     Mode = hvc,
64
65     get_winner_value(Game,WinnerValue),
66     WinnerValue \= winner,
67
68     \+ determine_draw(Game),
69
70     get_player_turn(Game,Player),
71     Player == secondPlayer,
72
73     computer_play(Game,ModifiedGame),
74
75     game_loop(ModifiedGame).
76
77 %The main game loop for computer_vs_computer gameplay
78 game_loop(Game):-
79     get_list_element(5,Game,Mode),
80     Mode = cvc,
81
82     get_winner_value(Game,WinnerValue),
83     WinnerValue \= winner,
84
85     \+ determine_draw(Game),
86
87     computer_play(Game,ModifiedGame),
88
89     game_loop(ModifiedGame).
90
91 %The game loop end condition
92 game_loop(Game):-
93     get_board(Game,PieceBoard),
94     display_megateh_board(0,Game,PieceBoard,4),
95     get_winner_value(Game,WinnerValue),
96     WinnerValue == winner,
97     true.
98
99 game_loop(Game):-
100     get_board(Game,PieceBoard),
101     display_megateh_board(0,Game,PieceBoard,4),
102     \+ determine_draw(Game),

```

```

103         true.
104
105 human_play(Game,ModifiedGame):-
106     repeat,
107     get_board(Game,PieceBoard),
108     display_megateh_board(0,Game,PieceBoard,4),
109     display_turn_info(Game),nl,
110     get_new_piece_source_coordinartes(SrcRow,SrcCol),
111     get_new_type_piece(NewPiece),
112     make_move(SrcRow,SrcCol,NewPiece,Game,ModifiedGame),
113     !.

```

7.7 utils.pl

```

1  %get list element (element position, list, query element)
2  get_list_element(0,[HeadElement|_],HeadElement).
3
4  get_list_element(Pos,[_|OtherElems],Symbol):-
5      Pos > 0,
6      Pos1 is Pos-1,
7      get_list_element(Pos1,OtherElems,Symbol).
8
9
10 %get matrix element (element row, element col, remainingLists, query
    ↪ element)
11 get_matrix_element(0, ElemCol, [ListAtTheHead|_], Elem):-
12     get_list_element(ElemCol, ListAtTheHead, Elem).
13
14 get_matrix_element(ElemRow, ElemCol, [_|RemainingLists], Elem):-
15     ElemRow > 0,
16     ElemRow1 is ElemRow-1,
17     get_matrix_element(ElemRow1, ElemCol, RemainingLists, Elem).
18
19
20 %set list element(index replacement, list, ResultantList)
21 set_list_element(0, Elem, [_|L], [Elem|L]).
22 set_list_element(I, Elem, [H|L], [H|ResL]):-
23     I > 0,
24     I1 is I-1,
25     set_list_element(I1, Elem, L, ResL).
26
27
28
29 %set matrix element (element row, element col, current list, resultant
    ↪ list)
30 set_matrix_element(0, ElemCol, NewElem, [RowAtTheHead|RemainingRows],
    ↪ [NewRowAtTheHead|RemainingRows]):-
31     set_list_element(ElemCol, NewElem, RowAtTheHead,
    ↪ NewRowAtTheHead).
32
33 set_matrix_element(ElemRow, ElemCol, NewElem,
    ↪ [RowAtTheHead|RemainingRows], [RowAtTheHead|ResultRemainingRows]):-

```

```

34     ElemRow > 0,
35     ElemRow1 is ElemRow-1,
36     set_matrix_element(ElemRow1, ElemCol, NewElem, RemainingRows,
    ↪     ResultRemainingRows).
37
38
39
40 initialize_random_seed:-
41     now(Usec), Seed is Usec mod 30269,
42     getrand(random(X, Y, Z, _)),
43     setrand(random(Seed, X, Y, Z)), !.

```