Azacru_1

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo azacru_1:
Nuno Miguel Outeiro Pereira – up201506265
Ricardo José Santos Pereira- up201503716

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

12 de novembro de 2017

Resumo

O trabalho desenvolvido consiste num jogo de tabuleiro para ser jogado entre duas pessoas denominado Azacru, um dos jogos pertencentes ao grupo Pacru, que tem como espaço de execução a linha de comandos.

Trata-se de um jogo onde o objetivo é deixar o adversário incapaz de realizar qualquer tipo de jogada, como tal, é um jogo que requer alguma preparação e previsão mental.

Tendo sido este jogo desenvolvido em PROLOG, uma linguagem de índole bastante diferente da nossa zona de conforto, deparámo-nos com alguns problemas de implementação, mas que através de alguma pesquisa e consulta foram ultrapassados.

Após a implementação deste jogo, podemos, de facto, assumir que a nossa destreza e conhecimento em PROLOG aumentou, visto que nos deparamos com várias situações que obrigaram a busca de conhecimento e tentativas de implementação que contornassem os entraves que foram aparecendo.

Conteúdo

1 Introdução

No âmbito da unidade curricular de Programação em Lógica, foi-nos concebida a tarefa de implementar um de uma lista de jogos de tabuleiro. O jogo selecionado por nós foi o Azacru.

O que nos motivou à sua implementação foi o aumento de conhecimento que ele proporcionou pois não conhecíamos o jogo nem nenhum do pacote de jogos Pacru. É um jogo que, como qualquer jogo de tabuleiro, obriga a prática e previsão mental, assim como à implementação de uma estratégia.

Os principais objetivos deste trabalho baseiam-se na consolidação de conhecimentos adquiridos durante o semestre.

Este relatório encontra-se estruturado em várias secções como é possível detetar pelo índice. Existem divisões para a história e regras do jogo, para a lógica por detrás do jogo assim como para a sua implementação em PROLOG, para o modo de interação com o utilizador, para conclusões retiradas após elaboração do trabalho e também secções para a bibliografia e anexos do código desenvolvido.

2 Azacru

História

O Azacru é um jogo que foi lançado em 2005 como parte integrante de um conjunto de jogos chamado Pacru Series 302 que continha vários jogos que partilhavam o mesmo tabuleiro, assim como as mesmas peças e marcos, conjunto este inventado por Mike Wellman.

Este jogo tem também o seu mestre, que neste caso é Martyn J.Hamer que já foi múltiplas vezes campeão do mundo.

Regras do Jogo

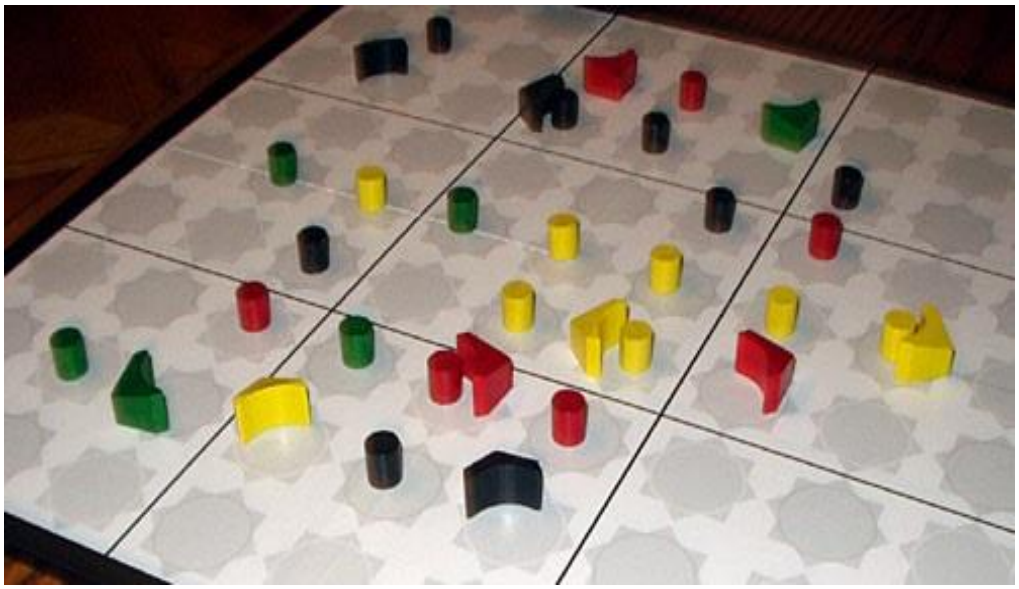
O Azacru é um jogo multi-jogador no qual cada um dos jogadores tem peças e marcadores de cor diferente. A ordem pela qual começam a jogar é deixada ao critério dos jogadores. Neste trabalho, vamos apenas implementar o jogo para 2 jogadores, sendo que por isso, cada jogador começa com 4 peças.

O jogo vai-se jogando alternadamente entre os jogadores, sendo que só é possível passar a vez caso o jogador esteja numa posição onde não consegue realizar qualquer jogada.

Os movimentos permitidos à peça são 3, sendo eles, em relação à posição da peça original, em frente, 45° à esquerda ou 45° à direita. Ao fim da jogada a peça fica a apontar na última direção em que foi jogada, sendo também colocado um marcador nessa posição do tabuleiro, exceto quando no final da jogada a peça tiver parado noutro sector.

O poder de alcance da peça numa jogada depende do sector em que se localiza, sendo que é o mesmo em casas à soma do número de marcadores da sua cor nesse sector. Quando não existem marcadores da cor da peça nesse sector o poder de alcance da peça é 1. Não é obrigatório percorrer todas as casas até ao poder máximo, a jogada pode ser de 1 casa até o número do alcance máximo.

Em baixo segue uma foto dum tabuleiro aquando da realização de um jogo entre 4 jogadores para se perceber melhor do que se fala quando se referem marcadores e sectores.

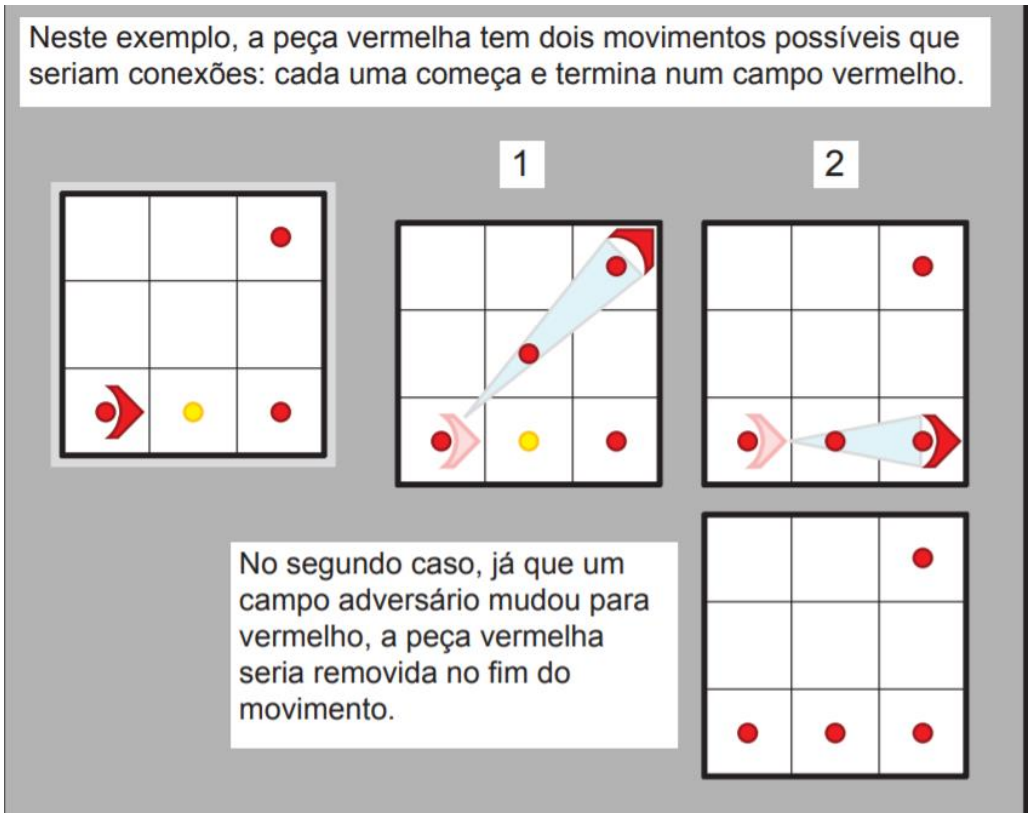


Quando a jogada inclui atravessar mais do que uma casa, não é permitido trocar a direção a meio do movimento, a direção pela qual se inicia o movimento é a direção pela qual a peça passará todas as casas.

Se no final da jogada a peça tiver passado para outro sector pode não terminar o seu movimento na posição em que iniciou a jogada, sendo que neste caso o jogador pode rodar a peça 45° no final da jogada quer seja para a esquerda ou para a direita.

Quando o jogador realiza um movimento da sua peça entre um campo da sua cor e outro dessa mesma cor, e não se passou através de uma peça, faz com que todos os campos intermédios fiquem também da cor da sua peça. Esta jogada pode também ser realizada saltando peças, contudo as casas intermédias entre a 1ª e a última não iram se alterar.

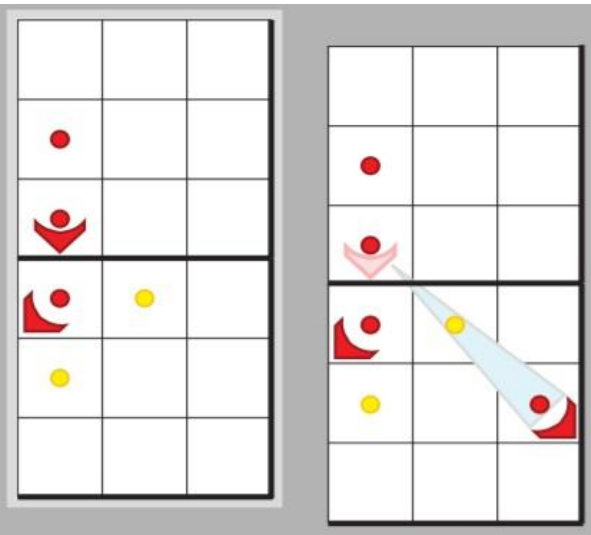
Quando, na jogada, são trocadas casas de cor do adversário para a cor da peça dessa jogada, a peça é então extraída do tabuleiro.



É proibido mover uma peça para uma casa com a cor do adversário, não se pode também mover uma peça para um campo ocupado por outra. Só é permitido saltar por cima de uma peça, se a casa onde se for parar já for da cor da peça jogada, de outra forma não é possível.

O jogo termina quando um dos jogadores não for capaz de realizar nenhuma jogada ou quando todas as suas peças foram retiradas do jogo. Quando tal se sucede, o jogador passa a vez, os restantes jogadores jogam mais um turno e depois disso contabilizam-se os marcadores de cada jogador e quem tiver mais acumulados ganha o jogo.

De seguida, segue-se um exemplo de um término de jogo:



Exemplo de fim de jogo (seja a ordem do turno: vermelha, verde, amarela)

Vermelha move-se

Verde move-se (e remove a última peça verde do tabuleiro)

Amarela move-se

Vermelha move-se

Verde anuncia 'não posso mover-me' ou 'passo'

Amarela move-se

Vermelha move-se

O jogo termina e os marcadores de cada jogador são contabilizados.

3 Lógica do Jogo

3.1 Representação do Estado do Jogo

Detalhando agora a implementação em Prolog do jogo, o tabuleiro traduzir-se-á por um aglomerado de 81 predicados de aridade 2 definidos dinamicamente, sendo compostos por duas listas de números inteiros representativos, respetivamente, de cada campo no tabuleiro, e contendo informação sobre qualquer peça ou marcador que se possa lá encontrar. Os predicados tem o formato seguinte:

board([CoordX, CoordY], [O, P, M])

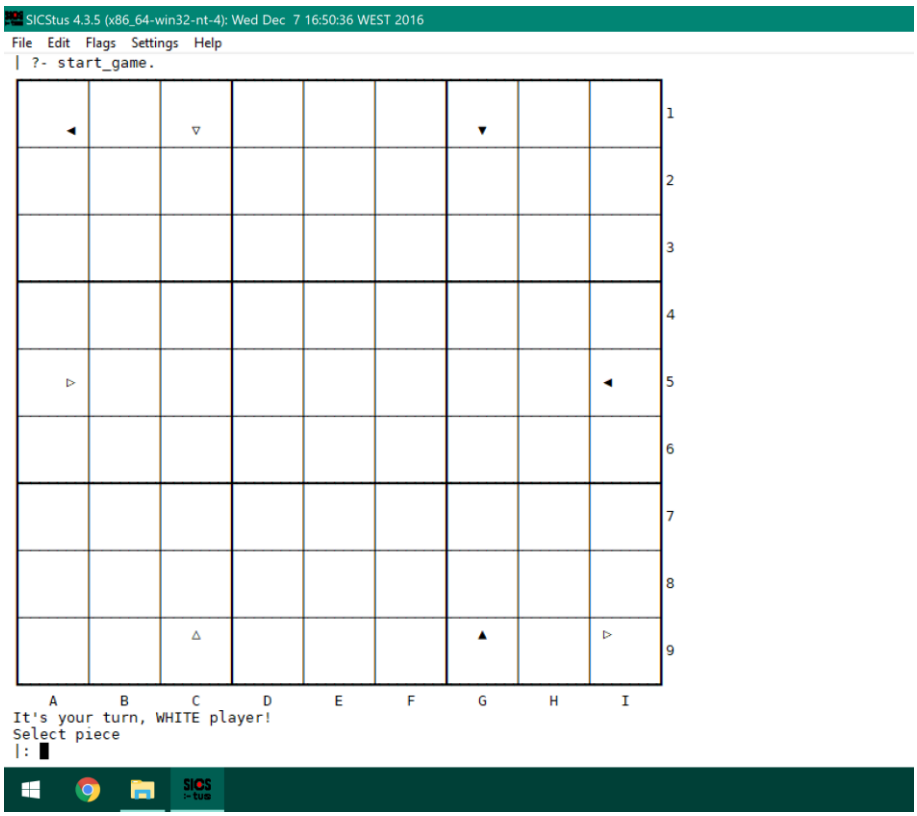
- CoordX é um número entre 1 e 9 representativo da coordenada em X da informação contida na segunda lista.
- CoordY é um número entre 1 e 9 representativo da coordenada em Y da informação contida na segunda lista.
- O é um número entre 1 e 8, correspondendo à orientação da peça.
- P é um número entre 1 e 2, correspondendo à identidade da peça que aí se encontra (caso pertença ao jogador 1 ou 2).
- M é um número entre 0 e 2, correspondendo à identidade do marcador que aí se encontra (sendo 0 caso o campo esteja vazio).
- Se não estiver nenhuma peça presente nesse campo, então O e P terão valor 0.

Adicionalmente, são também definidos dinamicamente 18 predicados de aridade 3 que guardam o 'poder de movimento' de cada jogador em cada sector, sob a forma:

sector(Sector, Player, Power)

Através de um predicado *check_sector*/3 existente no código, o acesso a estes é efetuado pela conversão de coordenadas em número de sector.

3.2 Visualização do Tabuleiro



O tabuleiro é representado através de chamadas a *put_code*/1 com códigos decimais referentes a caracteres Unicode (encoding utf8). Estas chamadas são sequenciadas através de uma série de predicados construidos com esse propósito; a complexidade do tabuleiro de jogo obriga a que a representação de cada campo seja feita através de vários caracteres ao longo de várias linhas.

3.3 Lista de Jogadas Válidas

Apesar de não ser possível obter uma lista de jogadas de forma intuitiva, podemos inquirir, relativamente a qualquer peça, sobre a existência de uma qualquer jogada válida através do predicado check_play/5 – esta é a forma de verificar se uma chamada a 'pass.' (passar a jogada) é válida, sendo isto determinante para o termino do jogo.

3.4 Execução de Jogadas

Cada ciclo de jogo, ou seja, cada turno é resolvido através de uma sequência de chamadas a certos predicados. A um nível mais alto, essa sequência poder-se-ia refletir por:

display_board/1 → *input*/7 → *update_board*/6

Uma verificação exterior antes de cada ciclo é feita com recurso a um dos valores devolvidos por *input*/7 para determinar a continuação do jogo.

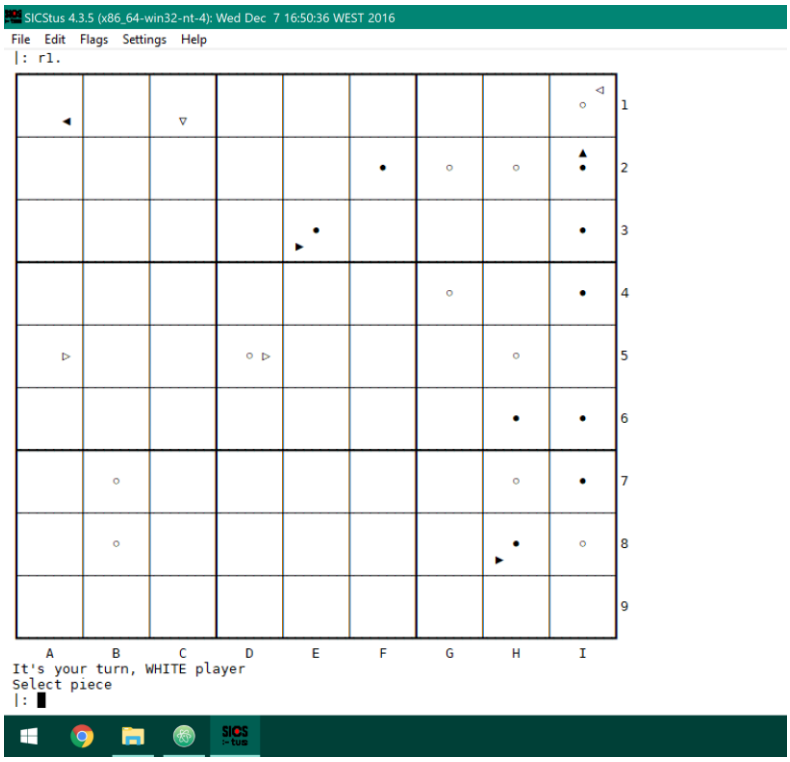
Tanto *display_board*/1 como *update_board*/6 são responsáveis simplesmente pela reprodução do tabuleiro, recorrendo a predicados interiores que não têm grande interesse do ponto de vista do funcionamento lógico da implementação; os predicados que validam e avaliam o *input* do utilizador estão, como seria de esperar, em *input*/7 que, internamente, omitindo algumas verificações e condições, assemelha-se a algo como:

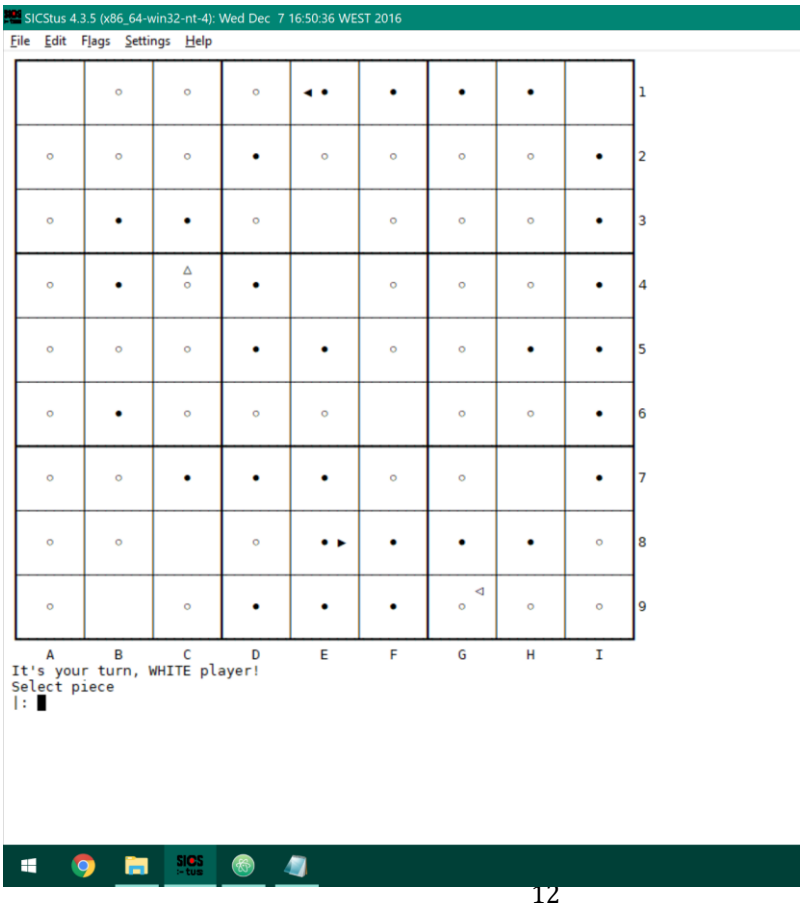
piece_input/4 → *movement_input*/4 [→ *orientation_input*/3]

Sendo que cada um destes abstrai uma sequência de predicados parecida com:

read/1 → *validate_X_input*/Y → *evaluate_X_input*/Z

onde X diz respeito ao predicado que chamou a sequência: se foi *piece_input*, então os predicados são *validate_piece_input* e *evaluate_piece_input*. Os predicados do tipo *validate_X_input*/Y testam o *input* relativamente à validade no contexto sintático, ou seja, se segue o aspeto do que seria esperado segundo as regras; já os do tipo *evaluate_X_input*/Z procuram testar a validade no contexto semântico, por outras palavras, se a jogada que está a ser descrita não viola nenhuma regra no contexto atual do jogo (exceção: não existe o predicado *evaluate_orientation_input*, uma vez que qualquer *input* de orientação válido sintaticamente também o é semanticamente).


SICStus 4.3.5 (x86_64-win32-nt-4): Wed Dec 7 16:50:36 WEST 2016
File Edit Flags Settings Help
| : rl.
A B C D E F G H I
It's your turn, WHITE player
Select piece
| :

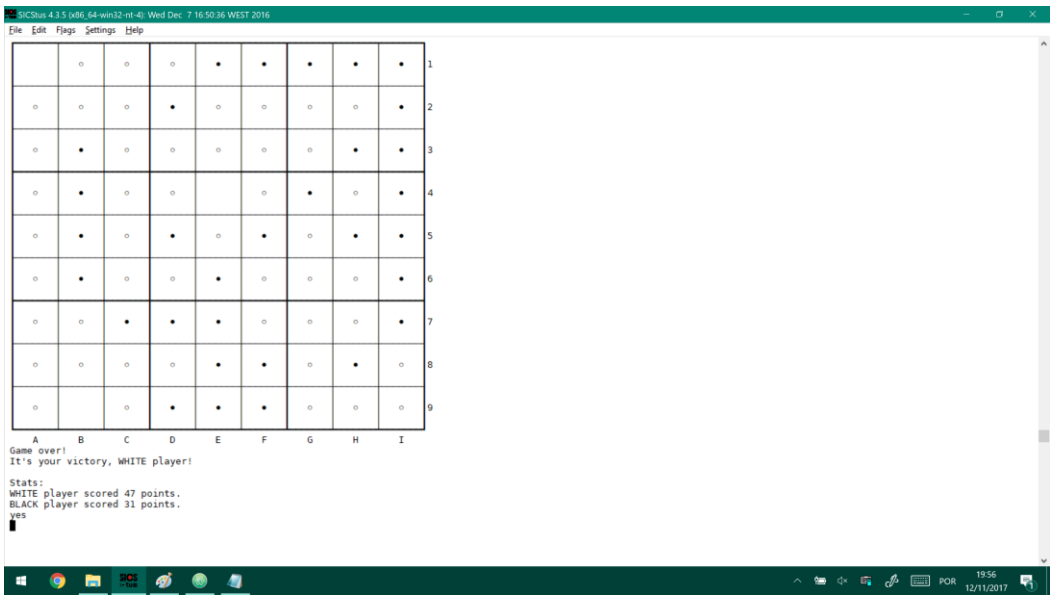

SICStus 4.3.5 (x86_64-win32-nt-4): Wed Dec 7 16:50:36 WEST 2016
File Edit Flags Settings Help
A B C D E F G H I
It's your turn, WHITE player!
Select piece
| :

3.5 Avaliação do Tabuleiro

Nenhuma avaliação do tabuleiro é feita durante o jogo; a implementação mantém-se relativamente ignorante das estatísticas do jogo que permitam ver qual dos jogadores tem vantagem sobre o outro; porque existe uma considerável variedade de jogadas disponíveis, sem que haja uma claramente vantajosa sobre qualquer outra, durante grande parte do jogo, seria impossível manter informação sobre todas elas sem se verificar um grande aumento na complexidade temporal e espacial do jogo.

3.6 Final do Jogo

O final do jogo é marcado pelo *input* de 'pass' por um dos jogadores: quando um jogador passa o turno (que implica que não existe nenhuma outra jogada válida para esse jogador; caso contrário, o *input* falha), o outro jogador só pode efetuar mais uma jogada, ao fim do qual o jogo termina, o tabuleiro é avaliado e o vencedor é determinado: esta avaliação consiste na soma do 'poder de movimento' de todos os sectores para cada jogador; o vencedor é aquele cuja soma seja mais elevada.



4 Interface com o Utilizador

É possível iniciar o jogo com o comando start_game, sendo que será apresentado o tabuleiro inicial, seguindo-se o menu para o jogador escolher peça que pretende mover. Após essa escolha, é pedido ao utilizador para indicar se se pretende mover para a frente, indicando a letra "f" seguida do número de casas a avançar, sendo que este número vai ser condicionado pelo seu poder de movimento naquele momento do jogo, ou "r" ou "l" consoante queria fazer o movimento de 45º para a direita ou esquerda, respetivamente, sendo também acrescentado à frente o número de casas.

O programa responde se é uma jogada válida e, de facto, a efetua ou, caso contrário, é apresentada uma mensagem a indicar que é uma jogada inválida, pedindo novo input ao utilizador.

É dada a vez ao outro jogador, e a partir daqui as peças passam a estar atribuídas a cada um dos jogadores, sendo que se um jogador tentar mover as peças do outro, o programa apresenta a mensagem de erro "Invalid piece", pedindo de novo input para escolher uma peça válida.

O programa vai alternando entre jogadores, seguindo-se o esquema aqui apresentado.

No caso de, após uma jogada, a peça trocar de sector, é perguntado ao utilizador para escolher a orientação que pretende dar à peça, seja ela, manter a direção com um "f", rodar 45º à esquerda com um "l" ou rodar 45º à direita com um "r".

Adicionalmente, o utilizador tem a hipótese de passar o turno através de "pass" quando não tem mais nenhuma jogada válida a fazer; tem também a possibilidade de corrigir alguma seleção, até ao final do turno, que não pretendesse fazer, através de 'back'.

5 Conclusões

Este projeto foi bastante útil visto que forçou a adquirição de novos conhecimentos e aumento da destreza em PROLOG.

Devido às dificuldades descritas no ponto 5 do capítulo, nenhum AI foi desenvolvido para jogar com o utilizador; consideramos adicionalmente que criar um AI simples que jogasse aleatoriamente não criaria grande desafio.

Contudo, consideramos que o balanço da implementação do jogo foi positivo.

Bibliografia

Sites:

- http://www.boardability.com/game.php?id=azacru
- http://www.pacru.com/rulesPT.pdf

```
/* :-encoding(utf8). */
/* Use font DejaVu Sans Mono, size 10 */

/* ====================== */
/* ===== Utilities ===== */
/* ====================== */

saferead(Read):-
    catch(read(Read), _Throw, (write('Invalid input!'), nl, !, fail)).

assertz_board([]).
assertz_board([H|T]):-
    H = [Coord, Info],
    assertz(board(Coord, Info)),
    assertz_board(T).
assertz_sector([]).
assertz_sector([H|T]):-
    H = [Sector, Player, Power],
    assertz(sector(Sector, Player, Power)),
    assertz_sector(T).

writecode(C):-
    put_code(C).
writecode(1, C):-
    put_code(C).
writecode(N, C):-
    put_code(C),
    Next is N - 1,
    writecode(Next, C).

nth(_, [], 0, _).
nth(E, [E|_], Nth, Nth).
nth(E, [_|L], Nth, N):-
    Next is N + 1,
    nth(E, L, Nth, Next).
nth(E, L, N):-
    Next is 1,
    nth(E, L, N, Next).

condition(Cond, True):-
    condition(Cond, True, true).
condition(Cond, True, False):-
    (Cond,!,
        True
    );(
        False
    ).

/* ================== */
/* ===== Options ===== */
/* ================== */

:-dynamic
    debug/1,
    player/1,
    board/2,
    sector/3.

start_game:-
    clear_game,
    assertz(player([1, 2])),
```

```
62      assertz_board([[[1, 1], [4, 2, 0]], [[2, 1], [0, 0, 0]], [[3, 1], [5, 1, 0]], [[4,
          1], [0, 0, 0]], [[5, 1], [0, 0, 0]], [[6, 1], [0, 0, 0]], [[7, 1], [5, 2, 0]], [[8,
           1], [0, 0, 0]], [[9, 1], [0, 0, 0]],
63                      [[1, 2], [0, 0, 0]], [[2, 2], [0, 0, 0]], [[3, 2], [0, 0, 0]], [[4,
                          2], [0, 0, 0]], [[5, 2], [0, 0, 0]], [[6, 2], [0, 0, 0]], [[7, 2],
                          [0, 0, 0]], [[8, 2], [0, 0, 0]], [[9, 2], [0, 0, 0]],
64                      [[1, 3], [0, 0, 0]], [[2, 3], [0, 0, 0]], [[3, 3], [0, 0, 0]], [[4,
                          3], [0, 0, 0]], [[5, 3], [0, 0, 0]], [[6, 3], [0, 0, 0]], [[7, 3],
                          [0, 0, 0]], [[8, 3], [0, 0, 0]], [[9, 3], [0, 0, 0]],
65                      [[1, 4], [0, 0, 0]], [[2, 4], [0, 0, 0]], [[3, 4], [0, 0, 0]], [[4,
                          4], [0, 0, 0]], [[5, 4], [0, 0, 0]], [[6, 4], [0, 0, 0]], [[7, 4],
                          [0, 0, 0]], [[8, 4], [0, 0, 0]], [[9, 4], [0, 0, 0]],
66                      [[1, 5], [3, 1, 0]], [[2, 5], [0, 0, 0]], [[3, 5], [0, 0, 0]], [[4,
                          5], [0, 0, 0]], [[5, 5], [0, 0, 0]], [[6, 5], [0, 0, 0]], [[7, 5],
                          [0, 0, 0]], [[8, 5], [0, 0, 0]], [[9, 5], [7, 2, 0]],
67                      [[1, 6], [0, 0, 0]], [[2, 6], [0, 0, 0]], [[3, 6], [0, 0, 0]], [[4,
                          6], [0, 0, 0]], [[5, 6], [0, 0, 0]], [[6, 6], [0, 0, 0]], [[7, 6],
                          [0, 0, 0]], [[8, 6], [0, 0, 0]], [[9, 6], [0, 0, 0]],
68                      [[1, 7], [0, 0, 0]], [[2, 7], [0, 0, 0]], [[3, 7], [0, 0, 0]], [[4,
                          7], [0, 0, 0]], [[5, 7], [0, 0, 0]], [[6, 7], [0, 0, 0]], [[7, 7],
                          [0, 0, 0]], [[8, 7], [0, 0, 0]], [[9, 7], [0, 0, 0]],
69                      [[1, 8], [0, 0, 0]], [[2, 8], [0, 0, 0]], [[3, 8], [0, 0, 0]], [[4,
                          8], [0, 0, 0]], [[5, 8], [0, 0, 0]], [[6, 8], [0, 0, 0]], [[7, 8],
                          [0, 0, 0]], [[8, 8], [0, 0, 0]], [[9, 8], [0, 0, 0]],
70                      [[1, 9], [0, 0, 0]], [[2, 9], [0, 0, 0]], [[3, 9], [1, 1, 0]], [[4,
                          9], [0, 0, 0]], [[5, 9], [0, 0, 0]], [[6, 9], [0, 0, 0]], [[7, 9],
                          [1, 2, 0]], [[8, 9], [0, 0, 0]], [[9, 9], [8, 1, 0]]]),
71      assertz_sector([[1, 1, 0], [2, 1, 0], [3, 1, 0], [4, 1, 0], [5, 1, 0], [6, 1, 0], [7,
           1, 0], [8, 1, 0], [9, 1, 0], [1, 2, 0], [2, 2, 0], [3, 2, 0], [4, 2, 0], [5, 2,
          0], [6, 2, 0], [7, 2, 0], [8, 2, 0], [9, 2, 0]]),
72      condition((\+ debug(on)),
73      (
74          game_engine,
75          clear_game
76      )).
77
78  continue_game:-
79      condition((\+ debug(on)),
80      (
81          game_engine,
82          clear_game
83      )).
84
85  clear_game:-
86      retractall(player(_)),
87      retractall(board(_, _)),
88      retractall(sector(_, _, _)).
89
90  /* ================== */
91  /* ===== Engine ===== */
92  /* ================== */
93
94  game_engine(Player, NextTurnPlayer):-
95      display_board(Player),
96      input(Player, Position, FinalPosition, Orientation, FinalOrientation, Type,
          NextTurnPlayer),
97      update_board(Player, Position, FinalPosition, Orientation, FinalOrientation, Type),
98      !.
99
100 game_engine:-
101     repeat,
```

```
102     player([Player, NextPlayer]),
103     condition((Player \== 0),
104     (
105         game_engine(Player, NextTurnPlayer),
106         retractall(player(_)),
107         assertz(player([NextPlayer, NextTurnPlayer])),
108         fail
109     )),
110     display_board(Player).
111
112 /* ================== */
113 /* ===== Display ===== */
114 /* ================== */
115
116 display_board(0):-
117     print_board,
118     write('Game over!'), nl, write('It\'s '),
119     sector(1, 1, WS1),
120     sector(2, 1, WS2),
121     sector(3, 1, WS3),
122     sector(4, 1, WS4),
123     sector(5, 1, WS5),
124     sector(6, 1, WS6),
125     sector(7, 1, WS7),
126     sector(8, 1, WS8),
127     sector(9, 1, WS9),
128     sector(1, 2, BS1),
129     sector(2, 2, BS2),
130     sector(3, 2, BS3),
131     sector(4, 2, BS4),
132     sector(5, 2, BS5),
133     sector(6, 2, BS6),
134     sector(7, 2, BS7),
135     sector(8, 2, BS8),
136     sector(9, 2, BS9),
137     WhiteScore is WS1 + WS2 + WS3 + WS4 + WS5 + WS6 + WS7 + WS8 + WS9,
138     BlackScore is BS1 + BS2 + BS3 + BS4 + BS5 + BS6 + BS7 + BS8 + BS9,
139     condition((WhiteScore > BlackScore),
140     (
141         write('your victory, WHITE player!'), nl, nl, write('Stats:'), nl,
142         write('WHITE player scored '), write(WhiteScore), write(' points.'), nl,
143         write('BLACK player scored '), write(BlackScore), write(' points.'), nl
144     ),
145     condition((BlackScore > WhiteScore),
146     (
147         write('your victory, BLACK player!'), nl, nl, write('Stats:'), nl,
148         write('BLACK player scored '), write(BlackScore), write(' points.'), nl,
149         write('WHITE player scored '), write(WhiteScore), write(' points.'), nl
150     ),
151     (
152         write('a draw!'), nl, nl, write('Stats:'), nl,
153         write('BLACK player scored '), write(BlackScore), write(' points.'), nl,
154         write('WHITE player scored '), write(WhiteScore), write(' points.'), nl
155     ))).
156
157 display_board(Player):-
158     print_board,
159     write('It\'s your turn, '),
160     condition((Player == 1),
161     (
162         write('WHITE')
```

```
163     ),
164     (
165         write('BLACK')
166     )),
167     write(' player!'), nl.
168
169 print_board:-
170     writecode(9487), writecode(7, 9473), writecode(9519), writecode(7, 9473), writecode
          (9519), writecode(7, 9473),
171     writecode(9523), writecode(7, 9473), writecode(9519), writecode(7, 9473), writecode
          (9519), writecode(7, 9473),
172     writecode(9523), writecode(7, 9473), writecode(9519), writecode(7, 9473), writecode
          (9519), writecode(7, 9473),
173     writecode(9491), nl,
174     print_board(1, 0).
175
176 print_board(10, _):-
177     writecode(9495), writecode(7, 9473), writecode(9527), writecode(7, 9473), writecode
          (9527), writecode(7, 9473),
178     writecode(9531), writecode(7, 9473), writecode(9527), writecode(7, 9473), writecode
          (9527), writecode(7, 9473),
179     writecode(9531), writecode(7, 9473), writecode(9527), writecode(7, 9473), writecode
          (9527), writecode(7, 9473),
180     writecode(9499), nl,
181     write('    A       B       C       D       E       F       G       H       I'), nl.
182 print_board(R1, 0):-
183     R2 is R1 + 1,
184     R3 is R2 + 1,
185     RNext is R3 + 1,
186     print_board(R1),
187     writecode(9504), writecode(7, 9472), writecode(9532), writecode(7, 9472), writecode
          (9532), writecode(7, 9472),
188     writecode(9538), writecode(7, 9472), writecode(9532), writecode(7, 9472), writecode
          (9532), writecode(7, 9472),
189     writecode(9538), writecode(7, 9472), writecode(9532), writecode(7, 9472), writecode
          (9532), writecode(7, 9472),
190     writecode(9512), nl,
191     print_board(R2),
192     writecode(9504), writecode(7, 9472), writecode(9532), writecode(7, 9472), writecode
          (9532), writecode(7, 9472),
193     writecode(9538), writecode(7, 9472), writecode(9532), writecode(7, 9472), writecode
          (9532), writecode(7, 9472),
194     writecode(9538), writecode(7, 9472), writecode(9532), writecode(7, 9472), writecode
          (9532), writecode(7, 9472),
195     writecode(9512), nl,
196     print_board(R3),
197     print_board(RNext, 1).
198 print_board(RNext, 1):-
199     writecode(9507), writecode(7, 9473), writecode(9535), writecode(7, 9473), writecode
          (9535), writecode(7, 9473),
200     writecode(9547), writecode(7, 9473), writecode(9535), writecode(7, 9473), writecode
          (9535), writecode(7, 9473),
201     writecode(9547), writecode(7, 9473), writecode(9535), writecode(7, 9473), writecode
          (9535), writecode(7, 9473),
202     writecode(9515), nl,
203     print_board(RNext, 0).
204
205 print_board(R):-
206     writecode(9475), print_element(1, R, 1), writecode(9474), print_element(2, R, 1),
          writecode(9474), print_element(3, R, 1),
207     writecode(9475), print_element(4, R, 1), writecode(9474), print_element(5, R, 1),
```

```
          writecode(9474), print_element(6, R, 1),
208       writecode(9475), print_element(7, R, 1), writecode(9474), print_element(8, R, 1),
          writecode(9474), print_element(9, R, 1),
209       writecode(9475), nl,
210       writecode(9475), print_element(1, R, 2), writecode(9474), print_element(2, R, 2),
          writecode(9474), print_element(3, R, 2),
211       writecode(9475), print_element(4, R, 2), writecode(9474), print_element(5, R, 2),
          writecode(9474), print_element(6, R, 2),
212       writecode(9475), print_element(7, R, 2), writecode(9474), print_element(8, R, 2),
          writecode(9474), print_element(9, R, 2),
213       writecode(9475), write(R), nl,
214       writecode(9475), print_element(1, R, 3), writecode(9474), print_element(2, R, 3),
          writecode(9474), print_element(3, R, 3),
215       writecode(9475), print_element(4, R, 3), writecode(9474), print_element(5, R, 3),
          writecode(9474), print_element(6, R, 3),
216       writecode(9475), print_element(7, R, 3), writecode(9474), print_element(8, R, 3),
          writecode(9474), print_element(9, R, 3),
217       writecode(9475), nl.
218
219   print_element(X, Y, R):-
220       board([X, Y], [O, P, M]),
221       print_element(O, P, M, R).
222   print_element(1, P, _M, 1):-
223       write('   '), print_piece(1, P), write('   ').
224   print_element(2, P, _M, 1):-
225       write('     '), print_piece(2, P), write(' ').
226   print_element(8, P, _M, 1):-
227       write(' '), print_piece(8, P), write('     ').
228   print_element(3, P, M, 2):-
229       write('   '), print_marker(M), write(' '), print_piece(3, P), write(' ').
230   print_element(7, P, M, 2):-
231       write(' '), print_piece(7, P), write(' '), print_marker(M), write('   ').
232   print_element(_O, _P, M, 2):-
233       write('   '), print_marker(M), write('   ').
234   print_element(4, P, _M, 3):-
235       write('     '), print_piece(4, P), write(' ').
236   print_element(5, P, _M, 3):-
237       write('   '), print_piece(5, P), write('   ').
238   print_element(6, P, _M, 3):-
239       write(' '), print_piece(6, P), write('     ').
240   print_element(_O, _P, _M, _R):-
241       write('       ').
242
243   print_piece(1, P):-
244       C is 9652 - P,
245       writecode(C).
246   print_piece(2, P):-
247       C is 9666 - P,
248       writecode(C).
249   print_piece(3, P):-
250       C is 9656 - P,
251       writecode(C).
252   print_piece(4, P):-
253       C is 9666 - P,
254       writecode(C).
255   print_piece(5, P):-
256       C is 9662 - P,
257       writecode(C).
258   print_piece(6, P):-
259       C is 9656 - P,
260       writecode(C).
```

```
261 print_piece(7, P):-
262     C is 9666 - P,
263     writecode(C).
264 print_piece(8, P):-
265     C is 9656 - P,
266     writecode(C).
267 
268 print_marker(0):-
269     write(' ').
270 print_marker(M):-
271     C is 9897 + M,
272     writecode(C).
273 
274 /* ================= */
275 /* ===== Input ===== */
276 /* ================= */
277 
278 input(Player, Position, FinalPosition, Orientation, FinalOrientation, Type,
      NextTurnPlayer):-
279     repeat,
280     piece_input(Player, Position, Power, Pass),
281     condition((Pass == 1),
282     (
283         Type = pass,
284         NextTurnPlayer is 0
285     ),
286     (
287         movement_input(Player, Position, Power, FinalPosition, Orientation, Type, Rotate,
            Back1),
288         condition((Back1 == 0),
289         (
290             condition((Rotate == 0; Type == drop),
291             (
292                 FinalOrientation is Orientation,
293                 NextTurnPlayer is Player
294             ),
295             (
296                 orientation_input(Orientation, FinalOrientation, Back2),
297                 condition((Back2 == 0),
298                 (
299                     NextTurnPlayer is Player
300                 ),
301                 (
302                     fail
303                 ))
304             ))
305         ),
306         (
307             fail
308         ))
309     )).
310 
311 piece_input(Player, Position, Power, Pass):-
312     repeat,
313     write('Select piece'), nl,
314     saferead(RawPosition),
315     validate_piece_input(RawPosition, Player, Position, Pass),
316     condition((Pass == 0),
317     (
318         evaluate_piece_input(Player, Position, Power)
319     )),
```

```
320     !.
321
322 movement_input(Player, Position, Power, FinalPosition, Orientation, Type, Rotate, Back):-
323     repeat,
324     write('Define movement'), nl,
325     saferead(RawMovement),
326     validate_movement_input(RawMovement, Movement, Back),
327     condition((Back == 0),
328     (
329         evaluate_movement_input(Player, Movement, Position, Power, FinalPosition,
            Orientation, Type, Rotate)
330     )),
331     !.
332
333 orientation_input(Orientation, FinalOrientation, Back):-
334     repeat,
335     write('Define orientation'), nl,
336     saferead(RawOrientation),
337     validate_orientation_input(RawOrientation, Back),
338     condition((Back == 0),
339     (
340         check_orientation(Orientation, RawOrientation, FinalOrientation)
341     )),
342     !.
343
344 /* ===== Input validation ===== */
345
346 validate_piece_input(pass, Player, _, Pass):-
347     \+ check_play(Player),
348     Pass is 1,
349     !.
350 validate_piece_input(pass, _, _, _):-
351     write('Invalid play!'), nl,
352     !,
353     fail.
354 validate_piece_input(RawPosition, _, [PositionX, PositionY], Pass):-
355     atom(RawPosition),
356     atom_chars(RawPosition, Position),
357     length(Position, 2),
358     [RawPositionX, RawPositionY] = Position,
359     nth(RawPositionX, ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i'], PositionX),
360     PositionX \== 0,
361     number_chars(PositionY, [RawPositionY]),
362     PositionY \== 0,
363     board([PositionX, PositionY], [_, P, _]),
364     P \== 0,
365     Pass is 0,
366     !.
367 validate_piece_input(_, _, _, _):-
368     write('Invalid input!'), nl,
369     !,
370     fail.
371
372 validate_movement_input(back, _, Back):-
373     Back is 1,
374     !.
375 validate_movement_input(RawMovement, [Direction, Distance], Back):-
376     atom(RawMovement),
377     atom_chars(RawMovement, Movement),
378     length(Movement, 2),
379     [Direction, RawDistance] = Movement,
```

```
380     member(Direction, ['l', 'f', 'r']),
381     member(RawDistance, ['1', '2', '3', '4', '5', '6', '7', '8']),
382     number_chars(Distance, [RawDistance]),
383     Back is 0,
384     !.
385 validate_movement_input(_, _, _):-
386     write('Invalid input!'), nl,
387     !,
388     fail.
389 
390 validate_orientation_input(back, Back):-
391     Back is 1,
392     !.
393 validate_orientation_input(RawOrientation, Back):-
394     atom(RawOrientation),
395     member(RawOrientation, ['l', 'f', 'r']),
396     Back is 0,
397     !.
398 validate_orientation_input(_, _):-
399     write('Invalid input!'), nl,
400     !,
401     fail.
402 
403 /* ===== Input evaluation ===== */
404 
405 evaluate_piece_input(Player, [PositionX, PositionY], Power):-
406     board([PositionX, PositionY], [OForward, Player, _]),
407     check_orientation(OForward, 'l', OLeft),
408     check_orientation(OForward, 'r', ORight),
409     check_limit(PositionX, PositionY, OLeft, LLeft),
410     check_limit(PositionX, PositionY, OForward, LForward),
411     check_limit(PositionX, PositionY, ORight, LRight),
412     \+ sort([LLeft, LForward, LRight], [0]),
413     check_sector(PositionX, PositionY, S),
414     sector(S, Player, SPower),
415     Power is max(SPower, 1),
416     check_play(PositionX, PositionY, OForward, Player, Power),
417     !.
418 evaluate_piece_input(_, _, _):-
419     write('Invalid piece!'), nl,
420     !,
421     fail.
422 
423 evaluate_movement_input(Player, [Direction, Distance], [PositionX, PositionY], Power,
      [FinalPositionX, FinalPositionY], Orientation, Type, Rotate):-
424     Distance =< Power,
425     board([PositionX, PositionY], [O, _, _]),
426     check_orientation(O, Direction, Orientation),
427     check_limit(PositionX, PositionY, Orientation, Limit),
428     Distance =< Limit,
429     check_movement(PositionX, PositionY, Distance, Orientation, Player, FinalPositionX,
        FinalPositionY, Type),
430     check_sector(PositionX, PositionY, Sector),
431     check_sector(FinalPositionX, FinalPositionY, FinalSector),
432     condition((FinalSector \== Sector),
433     (
434         Rotate is 1
435     ),
436     (
437         Rotate is 0
438     )),
```

```
439     !.
440 evaluate_movement_input(_, _, _, _, _, _, _, _):-
441     write('Invalid movement!'), nl,
442     !,
443     fail.
444 
445 /* ================== */
446 /* ===== Update ===== */
447 /* ================== */
448 
449 update_coords(X, Y, X, NY, 1):-
450     NY is Y - 1.
451 update_coords(X, Y, NX, NY, 2):-
452     NX is X + 1,
453     NY is Y - 1.
454 update_coords(X, Y, NX, Y, 3):-
455     NX is X + 1.
456 update_coords(X, Y, NX, NY, 4):-
457     NX is X + 1,
458     NY is Y + 1.
459 update_coords(X, Y, X, NY, 5):-
460     NY is Y + 1.
461 update_coords(X, Y, NX, NY, 6):-
462     NX is X - 1,
463     NY is Y + 1.
464 update_coords(X, Y, NX, Y, 7):-
465     NX is X - 1.
466 update_coords(X, Y, NX, NY, 8):-
467     NX is X - 1,
468     NY is Y - 1.
469 
470 update_board(_, FinalPosition, FinalPosition, _).
471 update_board(Player, [PositionX, PositionY], [FinalPositionX, FinalPositionY],
      Orientation):-
472     retract(board([PositionX, PositionY], [_, _, M])),
473     assertz(board([PositionX, PositionY], [0, 0, Player])),
474     condition((M \== Player),
475     (
476         check_sector(PositionX, PositionY, S),
477         sector(S, Player, Power),
478         NewPower is Power + 1,
479         retract(sector(S, Player, Power)),
480         assertz(sector(S, Player, NewPower)),
481         condition((M \== 0),
482         (
483             check_sector(PositionX, PositionY, OS),
484             sector(OS, M, OtherPower),
485             NewOtherPower is OtherPower - 1,
486             retract(sector(OS, M, OtherPower)),
487             assertz(sector(OS, M, NewOtherPower))
488         ))
489     )),
490     update_coords(PositionX, PositionY, PositionXNext, PositionYNext, Orientation),
491     update_board(Player, [PositionXNext, PositionYNext], [FinalPositionX,
        FinalPositionY], Orientation).
492 
493 update_board(_, _, _, _, _, pass).
494 update_board(Player, Position, FinalPosition, Orientation, FinalOrientation, rush):-
495     update_board(Player, Position, FinalPosition, Orientation),
496     retract(board(FinalPosition, [_, _, _])),
497     assertz(board(FinalPosition, [FinalOrientation, Player, Player])).
```

```
update_board(Player, Position, FinalPosition, Orientation, _, drop):-
    update_board(Player, Position, FinalPosition, Orientation).
update_board(Player, Position, FinalPosition, _, FinalOrientation, Type):-
    retract(board(Position, [_, _, M])),
    assertz(board(Position, [0, 0, M])),
    retract(board(FinalPosition, [_, _, _])),
    assertz(board(FinalPosition, [FinalOrientation, Player, Player])),
    condition((Type == move),
    (
        [FX, FY] = FinalPosition,
        check_sector(FX, FY, S),
        sector(S, Player, Power),
        NewPower is Power + 1,
        retract(sector(S, Player, Power)),
        assertz(sector(S, Player, NewPower))
    )).

/* ================== */
/* ===== Checks ===== */
/* ================== */

check_play(P):-
    board([X, Y], [O, P, _]),
    check_sector(X, Y, S),
    sector(S, P, SPower),
    Power is max(SPower, 1),
    check_play(X, Y, O, P, Power).

check_play(X, Y, O1, P, Power):-
    check_orientation(O1, 'l', O2),
    check_orientation(O1, 'r', O3),
    check_limit(X, Y, O1, L1),
    check_limit(X, Y, O2, L2),
    check_limit(X, Y, O3, L3),
    Limit1 is min(Power, L1),
    Limit2 is min(Power, L2),
    Limit3 is min(Power, L3),
    condition((Limit1 == 0; \+ check_play(X, Y, 0, O1, P, Limit1)),
    (
        condition((Limit2 == 0; \+ check_play(X, Y, 0, O2, P, Limit2)),
        (
            condition((Limit3 == 0; \+ check_play(X, Y, 0, O3, P, Limit3)),
            (
                !,
                fail
            ))
        ))
    )).
check_play(X, Y, L, O, P, L):-
    !,
    check_movement(X, Y, L, O, P, _, _, _).
check_play(X, Y, D, O, P, L):-
    \+ check_movement(X, Y, D, O, P, _, _, _),
    DNext is D + 1,
    !,
    check_play(X, Y, DNext, O, P, L).
check_play(_, _, _, _, _, _).

check_sector(X, Y, S):-
    S is div(X + 2, 3) + div(Y - 1, 3) * 3.

```

```
559 check_orientation(O, 'l', NewO):-
560     NewO is mod(O + 6, 8) + 1.
561 check_orientation(O, 'f', O).
562 check_orientation(O, 'r', NewO):-
563     NewO is mod(O, 8) + 1.
564
565 check_limit(_, Y, 1, L):-
566     L is Y - 1.
567 check_limit(X, Y, 2, L):-
568     L is min(9 - X, Y - 1).
569 check_limit(X, _, 3, L):-
570     L is 9 - X.
571 check_limit(X, Y, 4, L):-
572     L is min(9 - X, 9 - Y).
573 check_limit(_, Y, 5, L):-
574     L is 9 - Y.
575 check_limit(X, Y, 6, L):-
576     L is min(X - 1, 9 - Y).
577 check_limit(X, _, 7, L):-
578     L is X - 1.
579 check_limit(X, Y, 8, L):-
580     L is min(X - 1, Y - 1).
581
582 check_movement(X, Y, D, 1, P, X, FY, T):-
583     FY is Y - D,
584     check_field(X, FY, P, OldT),
585     YNext is Y - 1,
586     DNext is D - 1,
587     check_movement(X, YNext, DNext, 1, P, OldT, T).
588 check_movement(X, Y, D, 2, P, FX, FY, T):-
589     FX is X + D,
590     FY is Y - D,
591     check_field(FX, FY, P, OldT),
592     XNext is X + 1,
593     YNext is Y - 1,
594     DNext is D - 1,
595     check_movement(XNext, YNext, DNext, 2, P, OldT, T).
596 check_movement(X, Y, D, 3, P, FX, Y, T):-
597     FX is X + D,
598     check_field(FX, Y, P, OldT),
599     XNext is X + 1,
600     DNext is D - 1,
601     check_movement(XNext, Y, DNext, 3, P, OldT, T).
602 check_movement(X, Y, D, 4, P, FX, FY, T):-
603     FX is X + D,
604     FY is Y + D,
605     check_field(FX, FY, P, OldT),
606     XNext is X + 1,
607     YNext is Y + 1,
608     DNext is D - 1,
609     check_movement(XNext, YNext, DNext, 4, P, OldT, T).
610 check_movement(X, Y, D, 5, P, X, FY, T):-
611     FY is Y + D,
612     check_field(X, FY, P, OldT),
613     YNext is Y + 1,
614     DNext is D - 1,
615     check_movement(X, YNext, DNext, 5, P, OldT, T).
616 check_movement(X, Y, D, 6, P, FX, FY, T):-
617     FX is X - D,
618     FY is Y + D,
619     check_field(FX, FY, P, OldT),
```

```
620     XNext is X - 1,
621     YNext is Y + 1,
622     DNext is D - 1,
623     check_movement(XNext, YNext, DNext, 6, P, OldT, T).
624 check_movement(X, Y, D, 7, P, FX, Y, T):-
625     FX is X - D,
626     check_field(FX, Y, P, OldT),
627     XNext is X - 1,
628     DNext is D - 1,
629     check_movement(XNext, Y, DNext, 7, P, OldT, T).
630 check_movement(X, Y, D, 8, P, FX, FY, T):-
631     FX is X - D,
632     FY is Y - D,
633     check_field(FX, FY, P, OldT),
634     XNext is X - 1,
635     YNext is Y - 1,
636     DNext is D - 1,
637     check_movement(XNext, YNext, DNext, 8, P, OldT, T).
638
639 check_movement(_, _, 0, _, _, T, T).
640 check_movement(X, Y, D, 1, P, OldT, T):-
641     check_field(X, Y, P, OldT, NewT),
642     YNext is Y - 1,
643     DNext is D - 1,
644     check_movement(X, YNext, DNext, 1, P, NewT, T).
645 check_movement(X, Y, D, 2, P, OldT, T):-
646     check_field(X, Y, P, OldT, NewT),
647     XNext is X + 1,
648     YNext is Y - 1,
649     DNext is D - 1,
650     check_movement(XNext, YNext, DNext, 2, P, NewT, T).
651 check_movement(X, Y, D, 3, P, OldT, T):-
652     check_field(X, Y, P, OldT, NewT),
653     XNext is X + 1,
654     DNext is D - 1,
655     check_movement(XNext, Y, DNext, 3, P, NewT, T).
656 check_movement(X, Y, D, 4, P, OldT, T):-
657     check_field(X, Y, P, OldT, NewT),
658     XNext is X + 1,
659     YNext is Y + 1,
660     DNext is D - 1,
661     check_movement(XNext, YNext, DNext, 4, P, NewT, T).
662 check_movement(X, Y, D, 5, P, OldT, T):-
663     check_field(X, Y, P, OldT, NewT),
664     YNext is Y + 1,
665     DNext is D - 1,
666     check_movement(X, YNext, DNext, 5, P, NewT, T).
667 check_movement(X, Y, D, 6, P, OldT, T):-
668     check_field(X, Y, P, OldT, NewT),
669     XNext is X - 1,
670     YNext is Y + 1,
671     DNext is D - 1,
672     check_movement(XNext, YNext, DNext, 6, P, NewT, T).
673 check_movement(X, Y, D, 7, P, OldT, T):-
674     check_field(X, Y, P, OldT, NewT),
675     XNext is X - 1,
676     DNext is D - 1,
677     check_movement(XNext, Y, DNext, 7, P, NewT, T).
678 check_movement(X, Y, D, 8, P, OldT, T):-
679     check_field(X, Y, P, OldT, NewT),
680     XNext is X - 1,
```

```
681     YNext is Y - 1,
682     DNext is D - 1,
683     check_movement(XNext, YNext, DNext, 8, P, NewT, T).
684
685 check_field(X, Y, Player, T):-
686     board([X, Y], [_, 0, M]),
687     condition((M == Player),
688     (
689         T = rush
690     ),
691     condition((M == 0),
692     (
693         T = move
694     ),
695     (
696         fail
697     ))).
698 check_field(X, Y, Player, OldT, NewT):-
699     board([X, Y], [_, P, M]),
700     condition(((OldT == rush; OldT == drop), P \== 0),
701     (
702         NewT = jump
703     ),
704     condition((OldT == rush, M \== Player, M \== 0),
705     (
706         NewT = drop
707     ),
708     condition((OldT == move),
709     (
710         P == 0,
711         NewT = OldT
712     ),
713     (
714         NewT = OldT
715     ))))).
716
```