Faculdade de Engenharia da Universidade do Porto

Programação Funcional e em Lógica

Turma 4 - Flügelrad 7

António Santos, up201705558@fe.up.pt **Contribuição: 50%**

Luís Alves, up202108727@fe.up.pt **Contribuição: 50%**

Porto, 6 de novembro de 2023

Instalação e Execução

Para utilizar o jogo é necessário fazer *download* dos arquivos presentes na pasta PFL_TP1_T04_Flugelrad_7.zip e descompactá-los. Através do programa Sicstus executa-se o ficheiro main.pl que se encontra na pasta src no diretório principal. Depois só é necessário escrever o predicado play/0.

Descrição do Jogo

Flugelrad é um jogo de tabuleiro que consiste em sete hexágonos, cada um com uma pequena cavidade no centro, que partilham parte dos seus vértices. O jogo é jogado com dezoito pequenas bolas de duas cores, azuis e verdes. As bolas são movidas através de uma roda que se coloca no centro do hexágono e o objetivo é ter no tabuleiro seis bolas seguidas da cor que pertence ao jogador independentemente da forma que apresentam.

As bolas colocadas num hexágono não podem ser movidas em duas jogadas consecutivas e um jogador não pode mover o hexágono de modo a ficar na mesma posição que antes de ele jogar.

Lógica do Jogo

O *GameState* é representado pelo *Board* e pelo *Player*:

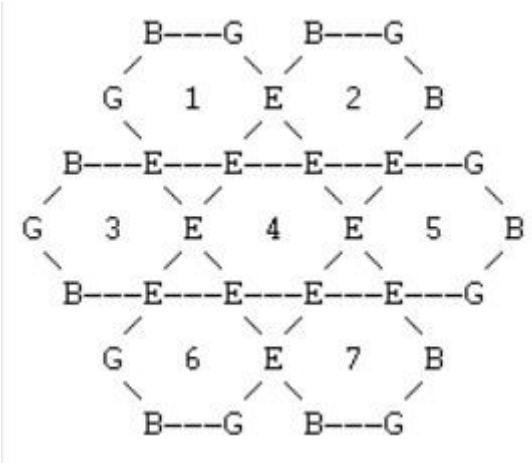
- **Board**: uma matriz que representa os sete hexágonos do jogo. Contém elementos que representam a cor das bolas, se algum buraco está vazio, o número do hexágono.
- **Player**: o jogador que pode ser *Player_1* ou *Player_2*.

Na implementação usamos estruturas auxiliares de modo a não alterar os valores diretamente do *Board*. Por isso, visualmente, o *Board* não é alterado pelo que permanece igual ao do estado inicial.

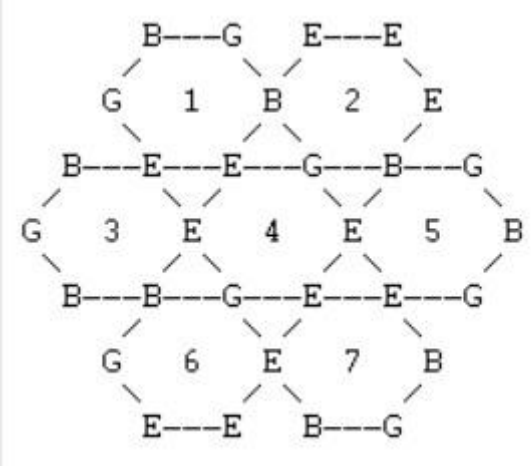
```
vertice(2, g, [1, 6], false).
vertice(3, b, [4, 6], false).
vertice(4, g, [3, 7], false).
vertice(5, g, [1, 9], false).
```

Estado do jogo inicial

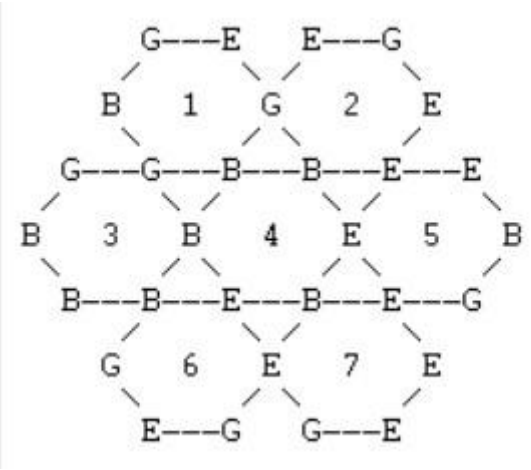
```
GameState([[
  [n, n, n, n, n,   n, 1, d, d,  d, 2, n, n,  n, 3, d, d,   d, 4],
  [n, n, n, n, n,   r, n, n, n,  n, n, l, n,  r, n, n, n,   n, n, l],
  [n, n, n, n, 5,   n, n, n, fi, n, n, n, 6,  n, n, n, se,  n, n, n, 7],
  [n, n, n, n, n,   l, n, n, n,  n, n, r, n,  l, n, n, n,   n, n, r],
  [n, n, 8, d, d,   d, 9, d, d,  d, 10, d, d, d, 11, d, d,  d, 12, d, d, d, 13],
  [n, r, n, n, n,   n, n, l, n,  r, n, n, n,  n, n, l, n,   r, n, n, n, n, n, l],
  [14, n, n, n, th, n, n, n, 15, n, n, n, fo, n, n, n, 16,  n, n, n, fif, n, n,  n, 17],
  [n, l, n, n, n,   n, n, r, n,  l, n, n, n,  n, n, r, n,   l, n, n, n, n, n, r],
  [n, n, 18, d, d,  d, 19, d, d, d, 20, d, d, d, 21, d, d,  d, 22, d, d, d, 23],
  [n, n, n, n, n,   r, n, n, n,  n, n, l, n,  r, n, n, n,   n, n, l],
  [n, n, n, n, 24,  n, n, n, si, n, n, n, 25, n, n, n, sev, n, n, n, 26],
  [n, n, n, n, n,   l, n, n, n,  n, n, r, n,  l, n, n, n,   n, n, r],
  [n, n, n, n, n,   n, 27, d, d, d, 28, n, n, n, 29, d, d,  d, 30]
], player_1]).
```



Estado do jogo intermédio



Estado do jogo final



Visualização do estado do jogo

No início do jogo, cada utilizador precisa de “configurar” o jogo segundo os seguintes parâmetros:

- Modo (Humano vs Humano, Humano vs Computador, Computador vs Computador).
- Nome do(s) jogador(es).
- Escolha da cor pelo *Player_1*.

Em todos os casos existe validação de input.

O predicado **display_game(+GameState)** tem como função principal “limpar” a consola e executar o predicado **display_board(+Board)** e é chamado no predicado **game_cycle(+GameState)**.

```
% display_game(+GameState)
% Displays the game
display_game([Board|_]) :-
  clear_console,
  display_board(Board).
```

Validação e Execução das Jogadas

É pedido ao jogador para indicar qual o hexágono e quantas vezes o pretende rodar. Consideram-se as informações dadas pelo jogador válidas quando estas estão dentro dos limites. É necessário escolher um hexágono do 1 ao 7 e um número de rotações entre 1 a 5.

Contudo, ainda existe mais uma condição, um hexágono não pode ser escolhido duas vezes consecutivas. Para isso guardamos num predicado qual foi o último hexágono escolhido de modo a conseguir validar uma jogada.

```
% move(+GameState, +Move, -NewGameState)
% Makes a move (rotates a hexagon)
move([_, Player], [Hexagon, Rotation], NewGameState) :-
  hexagon(Hexagon, Vertices),
  rotate_vertices(Vertices, Rotation),
  other_player(Player, NewPlayer),
  board(NewBoard),
  NewGameState = [NewBoard, NewPlayer].
```

Fim do Jogo

Para verificar o final do jogo, utilizamos o predicado **game_over(+GameState, -Winner)**. Como neste jogo é necessário saber que buracos/vértices estão à volta, decidimos criar uma espécie de grafo de modo a facilitar a nossa verificação.

Depois de termos o grafo implementado, desenvolvemos um algoritmo de DFS para verificar a condição de ter um agrupamento de 6 bolas “ligadas” entre si.

```
% dfs_vertices(+N, +Player, -Winner)
% Depth-first search algorithm for the vertices (Main algorithm)
dfs_vertices(0, Player, Winner) :-
  counter(Player, Counter),
  (Counter >= 6 -> Winner = Player, asserta(game_over_bool(1)), !;
  retract(counter(Player, _)),
  asserta(counter(Player, 0))
  ), !.
dfs_vertices(N, Player, Winner) :-
  counter(Player, Counter),
  (Counter >= 6 -> Winner = Player, asserta(game_over_bool(1)), !;
    retract(counter(Player, _)),
    asserta(counter(Player, 0)),
    setNotVisited(30),
    vertice(N, V, _, _),
    (V = e -> !;
      dfs(N, Player)
    ),
    N1 is N - 1,
    dfs_vertices(N1, Player, Winner)
  ),
  !.
```

```
% dfs(+N, +Player)
% Depth-first search algorithm
dfs(N, _) :-
  vertice(N, _, _, Visited),
  Visited = true, !.
dfs(N, Player) :-
  vertice(N, Value, Adjacent, _),
  retract((vertice(N, _, _, _))),
  asserta((vertice(N, Value, Adjacent, true))),
  color_of(Player, Color),
  (Value = Color -> counter(Player, Counter),
    NewCounter is Counter + 1,
    retract(counter(Player, Counter)),
    asserta(counter(Player, NewCounter)), dfs_adjacent(Adjacent, Player), !; !
  ).

% dfs_adjacent(+Adjacent, +Player)
% Depth-first search algorithm for the adjacent vertices
dfs_adjacent([], _) :- !.
dfs_adjacent([H|T], Player) :-
  dfs(H, Player),
  dfs_adjacent(T, Player).
```

Jogadas do Computador

Para criarmos as jogadas do computador utilizamos duas listas, uma com as opções possíveis de hexágonos e outra com as opções possíveis de rotações. Depois através da biblioteca *Random* do Prolog escolhemos uma opção aleatória de cada lista representando assim uma movimentação.

```
choose_move([Hexagon, Rotation], 2) :-
  ValidHexagonMoves = [1, 2, 3, 4, 5, 6, 7],
  ValidRotationMoves = [1, 2, 3, 4, 5],
  random(0, 7, HexagonIndex),
  nth0(HexagonIndex, ValidHexagonMoves, Hexagon),
  random(0, 5, RotationIndex),
  nth0(RotationIndex, ValidRotationMoves, Rotation),
  last_move(LastHexagon),
  (LastHexagon = 0 -> write('\nChoose a hexagon to play: \n'); format('\nChoose a hexagon to play except ~d: \n', LastHexagon)),
  format('Hexagon between 1 and 7: ~d\n', Hexagon),
  write('How many times do you want to rotate it? \n'),
  format('Rotation between 1 and 5: ~d\n', Rotation),
  validate_move(Hexagon), !.
```

Conclusões

O jogo foi implementado com sucesso apesar de não termos completado todas as funcionalidades pedidas. Foi um desafio desenvolver um jogo desconhecido e com um *layout* um pouco “estranho” a partir de uma linguagem com um paradigma diferente a que não estamos acostumados. Com isto conseguimos aplicar o que aprendemos nas aulas e para nós foi um desafio cumprido.

Bibliografia

- boardgamegeek/flugelrad
- swi-prolog