U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Relatório Intercalar

Professores:

Rui Carlos Camacho de Sousa Ferreira da Silva
Henrique Daniel de Avelar Lopes Cardoso
Daniel Augusto Gama de Castro Silva

Autores:

Catarina Alexandra Teixeira Ramos, up201406219
Inês Isabel Correia Gomes, up201405778
Turma 5, Small_Star_Empires_2

16 de outubro 2016

1. Descrição do jogo

Small Star Empire é um jogo estratégico de tabuleiro com peças hexagonais que tem como objetivo tomar posse do máximo de peças possíveis. Cada peça do tabuleiro é um sistema para onde o jogador se pode deslocar com as suas naves, conforme as regras, e no fim, por cada peça controlada pelo mesmo o jogador recebe pontos. Quem tiver a pontuação mais alta ganha.

1.1. Sistemas

Vazio	Estes sistemas funcionam como os sistemas estrela mas o jogador que o controla não receberá pontos por colônias no final do jogo.
Estrela	Sistemas constituídos por 1 a 3 planetas. No fim do jogo, o jogador recebe 1 ponto por cada planeta num sistema estrela dominado por ele.
Nébula	Quanto maior for o número de sistemas nebula de uma cor dominados pelo jogador, maior será a pontuação.
Wormhole	Sistemas que permitem ao jogador viajar rapidamente entre sistemas do mesmo género. Não podem ser dominados por nenhum jogador nem alojar nenhuma nave.
Blackhole	Sistemas altamente evitados devido à sua grande atracção gravitacional.O jogador não pode mover nenhuma das suas naves para este sistema.

Figura 1 - representação dos vários sistemas por ordem de descrição

1.2. Peças

Nave	Meio de deslocamento do jogador que permite colonizar novos sistemas. Cada jogador terá em sua posse 4 naves.
Colónia	Peças usadas na colonização e proclamação dum sistema. Cada jogador possui 16 colónias.
Estação de trocas	Peças com funcionalidades idênticas às colónia com a exceção de darem pontos extra dependendo se os vizinhos são ou não dominados por inimigos. Cada jogador possui 4 estações de troca.

1.3. Regras

O objetivo principal deste jogo é obter a maior pontuação possível. Para tal os jogadores devem seguir uma série de regras. Em primeiro lugar, os jogadores movem-se por turnos, e em cada turno devem mover um barco e estabelecer controlo.

Para estabelecer controlo o jogador deve posicionar uma colónia ou uma estação de troca.

O jogo termina quando ou nenhum dos jogadores tem jogadas possíveis ou não existem mais células livres.

2. Representação do estado do jogo

O tabuleiro deste jogo é modular, ou seja, pode ser encaixado por setores (conjunto de 7 peças) permitindo o utilizador escolher vários formatos para o seu tabuleiro. Neste caso usaremos a representação mais simples do tabuleiro para o estado inicial. No estado final poderão existir duas opções: um dos jogadores não encontra nenhuma posição possível para cada uma das suas naves ou todas as células estão ocupadas.

O estado do jogo pode ser representado internamente e externamente. Internamente corresponde aos dados interpretados pelo computador. Externamente corresponde à visualização pelo utilizador.

Na representação interna:

- 0 : corresponde aos espaços vazios (não são células do tabuleiro);
- [x, y, z] : corresponde a três id's
    - x : existem 7 id's correspondentes a cada sistema da seção 1.1
    - y : id do player conjugado com a respectiva colónia/sistema de troca.
        - 1 e 2 : jogador 1;
        - 2 e 3 : jogador 2;
        - 1 e 3 : colónias;
        - 2 e 4 : sistema de trocas.
    - z : número de naves nessa célula.

Na representação externa: [tipo de sistema, player, nº de naves]

- Tipo de sistema:
    - Sx : sistema de x planetas com x entre 0 e 3;
    - Nx : nebulosa de cor x com x igual a r, g ou b;
    - B : blackhole;
    - H : homeworld;
- Player : 1C/T ou 2C/T consoante é o jogador 1 ou 2 e se colocou uma colónia ou sistema de troca.

2.1. Estado Inicial

```
[ [0, [1,-1,0], 0, [5,-1,0], 0, [2,-1,0], 0, [0,-1,0], 0 ],
  [ [4,-1,0], 0, [6,0,4], 0, [3,-1,0], 0, [7,-1,0], 0, [3,-1,0] ],
  [0, [2,-1,0], 0, [0,-1,0], 0, [4,-1,0], 0, [1,-1,0], 0 ],
  [ [7,-1,0], 0, [2,-1,0], 0, [5,-1,0], 0, [6,2,4], 0, [4,-1,0] ],
  [0, [3,-1,0], 0, [5,-1,0], 0, [0,-1,0], 0, [4,-1,0], 0] ]
```

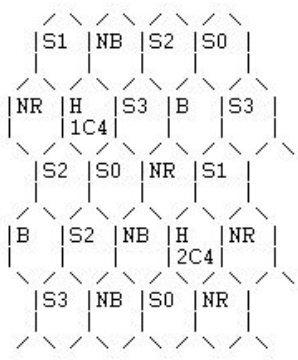

Figura 2 - Representação do estado inicial

2.2. Estado Final

```
[ [0, [1,1,0], 0, [5,2,1], 0, [2,0,1], 0, [0,3,0], 0 ],
  [ [4,2,1], 0, [6,0,0], 0, [3,2,0], 0, [7,-1,0], 0, [3,3,2] ],
  [ 0, [2,3,0], 0, [0,1,0], 0, [4,0,1], 0, [1,1,0], 0 ],
  [ [7,-1,0], 0, [2,0,1], 0, [5,2,0], 0, [6,2,0], 0, [4,2,0] ],
  [ 0, [3,0,0], 0, [5,1,1], 0, [0,0,0], 0, [4,3,0], 0 ] ]
```

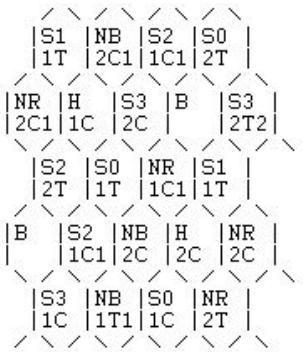

Figura 3 - Representação do tabuleiro numa fase final onde todas as células estão ocupadas.

```
[ [0, [1,2,0], 0, [5,3,0], 0, [2,0,1], 0, [0,2,1], 0 ],
  [ [4,2,1], 0, [6,0,0], 0, [3,0,0], 0, [7,-1,0], 0, [3,2,1] ],
  [ 0, [2,1,0], 0, [0,0,0], 0, [4,0,1], 0, [1,0,1], 0 ],
  [ [7,-1,0], 0, [2,2,0], 0, [5,2,0], 0, [6,2,0], 0, [4,3,1] ],
  [ 0, [3,-1,0], 0, [5,-1,0], 0, [0,-1,0], 0, [4,-1,0], 0 ] ]
```

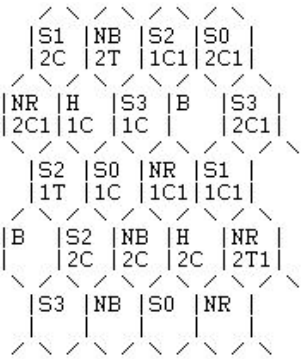

Figura 4 - Representação do tabuleiro quando um dos jogadores não consegue fazer mais nenhuma jogada possível.

3. Visualização do tabuleiro em modo de texto

Em Prolog são usados dois predicados para representar as células do tabuleiro:

- **player**(ID, Numero_do_jogador, Colonia_Trade).
- **systemType**(ID, Nome_sistema, Propriedade).

O **player** tem um id associado bem como o número do jogador (1 ou 2) e se essa célula está ocupada por uma colónia ou estação de troca. O **systemType** é constituído por um ID, o nome da célula e a propriedade associada (por exemplo, na nebulosa tem a sua cor). Além destes dois predicados ainda é representado por um número 'N' que representa o número de naves dessa célula.

Para imprimir o tabuleiro são usados os seguintes predicados:

- **displayBoard**(I).
  - **boardInfo**(I).
  - **board**(I, [L1 | L2]).
  - **displayTopLine**(L1).
  - **displayMatrix2D**([L1 | L2]).
    - **displayLine**(L1).
      - **displayLine1**([L1 | L2]).
        - **displayInfo1**(E1)
          - **systemType**(ID, _A, _B).
          - **displaySystem**(ID).
      - **displayLine2**([L1 | L2]).
        - **displayInfo2**(E2).
          - **player**(ID, _A, _B).
          - **displayPlayer**(ID).
      - **displayBottomLine**([L1 | L2]).

O **displayBoard** chama recursivamente cada predicado abaixo com o número do tabuleiro escolhido. **boardInfo** apenas disponibiliza um texto de ajuda à interpretação do tabuleiro. b**oard** recebe o número do tabuleiro e procura-o na lista de factos do predicado **board**. De seguida, **displayTopLine** começa por desenhar a parte superior da célula, e o **displayMatrix2D** passa à interpretação do tabuleiro, fazendo display linha a linha através do predicado **displayLine**. O displayLine é dividido em 3 displays: **displayLine1** que representa a informação sobre o sistema; **displayLine2** que representa a informação sobre o player; **displayBottomLine** que desenha a parte de baixo da célula.

```
=============== BOARD INFO ===============
```


```
Cell(Type,Prop,Team,Aloc,NShips)

Type:  (H) - HomeBase  (S) - Star System  (N) - Nebula System  (B) - Blackhole  (W) - Whormhole
Properties:  0-3 - Planets  (R)/(B) - Color
Team:  (1)- Red team  (2) - Blue team
Alocated:  (C) - Colony  (T) Trade Center
NShips: 1-4
```

Figura 5 - Representação da informação do tabuleiro

4. Movimentos

Um jogador deve se mover em qualquer uma das 6 direções possíveis, em linha reta. O jogador pode ainda mover o número de casas que escolher, desde que respeite as seguintes regras:

- O jogador pode:
  - mover-se para qualquer sistema não ocupado;
  - passar por sistemas sob o seu controlo;
  - passar por wormholes;
- O jogador não pode:
  - mover para um sistema sob o seu controlo;
  - passar por sistemas inimigos;
  - mover ou passar por blackholes;
  - mover para um wormhole;

Para processar o movimento iremos utilizar três predicados. **ValidMove** verifica se o movimento é válido numa determinada direcção para n unidades. Este predicado irá-se chamar recursivamente de forma a validar cada unidade percorrida. **FreeCell** que verifica se há algum movimento possível para uma determinada nave. Este predicado chama recursivamente **ValidMove** até encontrar uma célula livre ou até percorrer todas as direcções e distância (em células) possível.

Temos ainda o predicado **wormholeTeleport** que verifica se a partir de uma célula tipo wormhole é possível ir para uma outra do mesmo tipo.

Para ser mais eficiente, podemos também utilizar o predicado **ValidShips** para obter uma lista de coordenadas (outra lista) da posição das naves de um jogador com possíveis movimentos válidos. Desta forma, podemos verificar onde se encontram as naves, para facilitar a escolha de qual destas deslocar, e também, controlar o final do jogo caso a lista esteja vazia.

```
    | 2 |   | 3 |
| 1 |   | A |   | 4 |
    | 6 |   | 5 |
```

A partir da célula A é possível deslocar uma nave em direção às células 1, 2, 3, 4, 5 e 6. Estes números irão representar as 6 direcções possíveis a partir da célula central.

validMove(Jogador,Linha,Coluna,Direcção,Resultado**)**.
validMove(Jogador,Linha,Coluna,Direcção,Unidades,Resultado**).**
freeCell(Jogador,Linha,Coluna,Resultado**).**
wormholeTeleport(Jogador,Linha_Inicial,Coluna_Inicial,Linha_Final,Coluna_Final,Direcção,Resultado**).**
validShips(Jogador,Posição_Naves**).**

Para uma boa implementação dos predicados é necessário que a célula em 'Linha' e 'Coluna' pertença a 'Jogador', onde se encontre pelo menos uma nave e que respeite todas as regras acima mencionadas. No caso específico do **wormholeTeleport**, temos que verificar se as coordenadas iniciais e finais são válidas e testar se existe movimento possível para uma determinada 'Direcção' a partir do wormhole. Temos ainda que ter em conta que, ao deslocar uma nave temos que evitar os blackholes e processar os wormholes.