

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 de tabuleiro formado por sistemas hexagonais onde o jogador os pode controlar ao colocar uma estação de trocas ou colónia no sistema. Para se deslocar para um sistema o jogador utiliza as suas naves. O objetivo do jogo é obter a melhor pontuação possível.

* 1. Sistemas
     1. Estrela

Estes sistemas podem ser constituídos por 1 a 3 planetas em que o jogador pode colocar um colónia ou estação de trocas, desta forma, passa a ter controle sobre este sistema. No fim do jogo, o jogador recebe 1 ponto por cada planeta num sistema estrela dominado por ele.

* + 1. Wormhole

Estes sistemas permitem ao jogador viajar rapidamente entre sistemas do mesmo gênero. Não podem ser dominados por nenhum jogador nem alojar nenhuma nave.

* + 1. Nebula

Estes sistemas podem ser controlados da mesma maneira que os sistemas estrela e no fim, quanto maior for o número de sistemas nebula dominados pelo jogador maior será a pontuação.

* + 1. Buraco Negro

Estes sistemas são altamente evitados devido à sua grande atracção gravitacional que representa um grande perigo para as naves. Desta forma um jogador não pode mover nenhum dos seus navios para este sistema.

* + 1. Vazio

Estes sistemas funcionam como os sistemas estrela mas o jogador que o controla não receberá pontos por colónias no final do jogo.

* 1. Peças
     1. Nave

Este tipo de peças são o meio de deslocamento do jogador o que os permite colonizar novos sistemas. Cada jogador terá em sua posse 4 naves.

* + 1. Colónia

Este tipo de peças são usadas para colonizar um sistema e proclamar o domínio sobre um sistema. Cada jogador possui 16 colónias.

* + 1. Estação de trocas

Este tipo de peças funcionam como as colónias com a exceção de que podem dar uns bónus extras no final do jogo. Cada jogador possui 4 estações de troca.

* 1. 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 disponíveis.

1. Representação do estado do jogo

O tabuleiro deste jogo é modular, ou seja, pode ser encaixado por setores 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;
* [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
  + Nx : nebulosa de cor x
  + 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.
  1. Estado Inicial

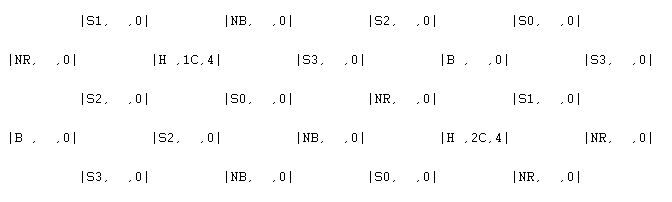
[ [0, [1,-1,0], 0, [5,-1,0], 0, [2,-1,0], 0, [0,-1,0], 0 ],

[ [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] ]



* 1. Estado Final

Todas as células do tabuleiro estão ocupadas:

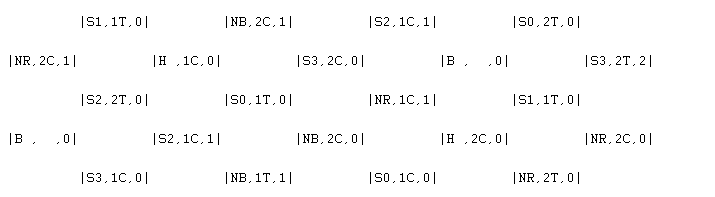
[ [0, [1,1,0], 0, [5,2,1], 0, [2,0,1], 0, [0,3,0], 0 ],

[ [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 ] ]



Um dos jogadores não consegue fazer mais nenhuma jogada possível.

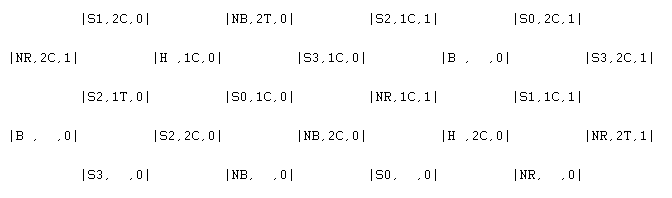
[ [0, [1,2,0], 0, [5,3,0], 0, [2,0,1], 0, [0,2,1], 0 ],

[ [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 ] ]



1. 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).

Para imprimir o tabuleiro são usados os seguintes predicados:

* **displayBoard**(I).
  + **boardInfo**(I).
  + **board**(I,S).
  + **displayMatrix2D**(S).
    - **displayLine**(L1).
      * **displayCell**( [ IDs | [ IDp | [ N | [ ]]]] ).
        + **displaySystem**(IDs).
        + **displayPlayer**(IDp).

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 o **displayMatrix2D** passa à interpretação do tabuleiro fazendo display linha a linha através do predicado **displayCell**. Por sua vez é feito o **displaySystem** e **displayPlayer** dos predicado **systemType** e **player**, respetivamente.

//Deve ser incluída pelo menos uma imagem correspondente ao output produzido ou pretendido

1. 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. **X** (predicado) que verifica se o movimento é válido para uma das células adjacentes à nave numa determinada direcção. **Y** que verifica se o movimento é válido numa determinada direcção para n unidades. **Y** irá chamar recursivamente o predicado **X** de forma a validar cada unidade percorrida. E por fim, **Z** que verifica se há algum movimento possível para uma determinada nave. **Z** chama recursivamente **Y** até encontrar uma célula livre (yes) ou até percorrer todas as direcções e distância (em células) possível.

No caso específico da célula wormhole, temos o predicado **W** que recebe as coordenadas de outra célula wormhole para onde se irá deslocar. Será depois necessário testar se existe movimento possível a partir do wormhole final visto que não se pode mover para o mesmo.

validMoves(Player, Board) -> vê se existem movimentos válidos para este jogador. Devolve uma lista de células

usa

shipsFromPlayers(Player, Board) -> que vê quais as células que têm barcos deste jogador

validMovesFromCell() -> igual à validMoves só que recebe uma célula especifica (o validMoves chama esta para cada célula do shipsFromPlayers)

moveShipToCell() -> recebe 2 células, onde está e para onde vai. Verifica se a célula para onde vai está na lista de células válidas para essa célula

//preciso dar nomes aos predicados