Small Star Empires

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Small_Star_Empires_3:
José Aleixo Peralta da Cruz - up201403526
José Carlos Alves Vieira - up201404446

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

13 de Novembro de 2016

Resumo

O *Small Star Empires* é um jogo de tabuleiro bastante complexo. Não só tem vários tipos de peças, como diferentes categorias de casas e sistema de pontuação complicado. Para juntar toda esta informação relativa ao tabuleiro, foi necessário implementar 3 níveis de listas de Prolog: um para as linhas do tabuleiro, outro para cada casa do tabuleiro e outro para armazenar as naves daquela casa.

Além daquilo que o tabuleiro contém, a forma hexagonal das suas casas tornou ainda mais árdua a tarefa de alterar as estruturas de dados, pois o comum sistema de coordenadas matriciais não bastava para poder obter a informação necessária, pelo que desenvolvemos funções de *display* e de acesso segundo as coordenadas que uma determinada casa tinha na lista de listas que representava o tabuleiro, e que também foram utilizadas para as funções de movimento.

Certas regras de jogo incluíam a inclusão de ainda mais informação. Por exemplo, verificar a cor das casas do tabuleiro. Assim, algumas regras que não alteravam significativamente o padrão de jogo foram alteradas para a representação ser mais simplificada.

Por possuir tantas condições declaradas, o código do programa desenvolvido é bastante extenso.

1. Introdução

No âmbito da unidade curricular de Programação Lógica, foi atribuída a tarefa de simular um jogo de tabuleiro em computador, recorrendo à linguagem de programação Prolog. Dos jogos propostos, o *Small Star Empires* sobressaiu-se como o mais interessante, pelo que foi o escolhido para desenvolvimento.

Neste relatório, estão explícitas as principais estratégias invocadas para tratar da parte lógica do jogo. Primeiramente é referida a forma de representação do estado do jogo e como ele é visualizado. Aborda-se também a forma de verificar as jogadas válidas e de as executar, tanto para restringir os movimentos do utilizador como para gerar os da inteligência artificial. Por fim, será declarada a condição que deteta o final de jogo e como é decidido o vencedor.

2. O Jogo

História

A empresa que criou o jogo (Archona Games) recorreu ao Kickstarter para tentar fazer com que o mesmo fosse possível, e conseguiu, adquirindo fundos suficientes para levar o projecto avante.

4 das maiores civilizações da galáxia adquiriram a habilidade de viajar à velocidade da luz, e estão prontas para impôr o seu domínio pelo vasto oceano de estrelas!

A civilização dos jogadores é uma dessas 4, e é o seu dever fazer com que a mesma seja a mais poderosa e com mais domínios.

Esse domínio será adquirido pelas viagens por entre planetas, sistemas e nebulosas com as naves da civilização.

Será preciso poder estratégico, portanto prepara-te para seres o império mais poderoso da galáxia!

Objetivo

Este é um jogo para 2 a 4 jogadores. O objetivo é colonizar a galáxia usando as naves, que serão movidas num tabuleiro hexagonal, sendo cada hexágono um sistema.

A colonização é realizada através da colocação de colónias ou de estações de comércio, que equivale ao controlo dentro daquele sistema.

No final do jogo, os jogadores calculam os pontos de cada um, dependendo esses da quantidade de sistemas que cada um controla.

O jogador com mais pontos é anunciado vencedor!

Regras

<u>Tipos de Sistemas:</u>

1. Sistemas Mãe: cada jogador começa no seu sistema mãe. Inicialmente, estes sistemas já estão ocupados pelo jogador correspondente, não sendo necessário colonizá-lo nem construir uma estação de comércio. No entanto, não dão pontos ao jogador que o controla no final da partida, mas contam como bónus de território e para os jogadores que tiverem algo adjacente a este sistema.
2. Sistemas de Estrelas: são colonizáveis. Cada sistema tem entre 1-3 planetas e oferecem 1-3 pontos no final da partida, dependendo do número de planetas.
3. Sistemas Nebulosos: quantos mais destes sistemas

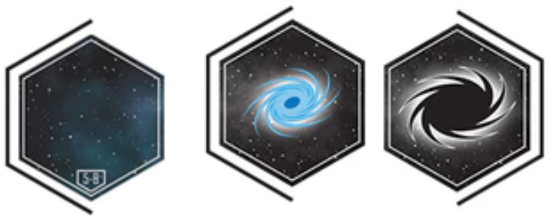
Figura 1 - Sistemas-mãe

Figura 2 - Sistemas de estrelas

da mesma cor forem controlados pelo mesmo jogador, mais pontos esse jogador recebe no final da partida. 1 sistema equivale a 2 pontos, 2 sistemas a 5 pontos e 3 sistemas a 8 pontos.

Figura 3 - Sistemas nebulosos

4. Sistemas Vazios: são colonizáveis. Contam como bónus de território caso estejam colonizados, mas não fornecem pontos no final da partida, exceto se tiver sido construída uma estação de comércio e houver sistemas controlados pelos oponentes adjacentes a este.
5. Buraco da Minhoca: podem ser usados para rapidamente viajar para diferentes cantos da galáxia.
6. Buracos Negros: são muitos perigosos, razão suficiente para que as naves não possa passar por cima nem mover-se para dentro de buracos negros.



Componentes dos jogadores:

1. Naves: cada jogador usa 2-4 naves. Podem sobrevoar sistemas controlados pelo jogador correspondente, mas buracos negros e sistemas controlados por outros jogadores impedem as naves de se deslocarem desse ponto adiante.
2. Colónias: usadas para marcar controlo por um jogador sobre um sistema.
3. Estações de Comércio: igual às colónias, mas no final da partida podem dar pontos bónus.

Flow da partida:

1. Escolha da cor do jogador (vermelho, verde, azul ou amarelo).
2. Posicionamento das suas naves no correspondente sistema mãe.

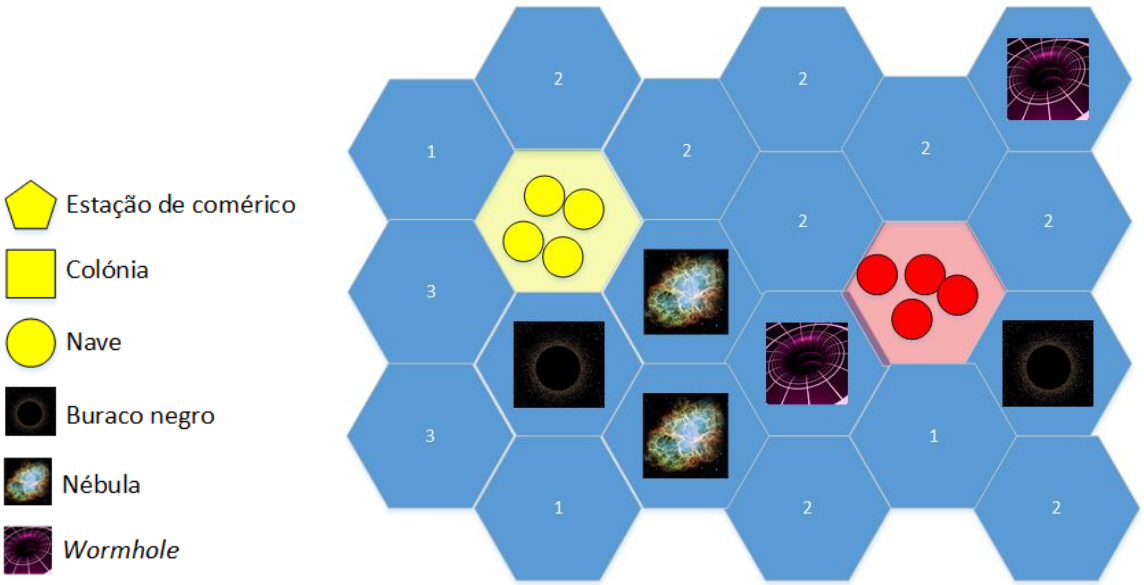
3. Se algum dos jogadores tiver vencido a ronda anterior, é esse mesmo que começa a partida. Se ainda não tiver sido feito nenhuma partida, o jogador que começa é o mais novo.
4. Sistema de funcionamento por turnos, avançando de jogador para jogador usando o método dos ponteiros do relógio (sentido horário).
5. O jogador move uma das suas naves, e onde pousar tem de construir uma colônia ou uma estação de comércio, ficando com o controlo do sistema. É proibido alterar o que já está construído numa fase posterior do jogo.
6. Se algum jogador não conseguir mover pelo menos uma das suas naves, o turno passa para o próximo jogador.
7. Realiza-se este ciclo até que mais nenhuma nave possa viajar ou todos os sistemas estiverem ocupados.
8. A partida acaba quando todos os sistemas estiverem ocupados ou todas as naves ficaram impossibilitadas de viajar.
9. É realizado o cálculo dos pontos:
    a. Sistemas de Estrelas: 1 ponto por cada planeta no sistema.
    b. Sistemas Nebulosos: 2 pontos por 1 nebulosa de uma cor, 5 pontos por 2 nebulosas da mesma cor, e 8 pontos por 3 nebulosas da mesma cor.
    c. Estações de comércio: 1 ponto por cada estação de comércio, colónia ou sistema mãe de outro jogador adjacente a esta estação.
    d. Pontos bônus: 3 pontos para o jogador com o maior território (o sistema mais conectado num único território).
10. No caso de empates (ordem de desempates):
    a. jogador com mais colónias por colonizar.
    b. jogador com mais estações de comércio por construir.
    c. jogador com mais planetas controlados.
    d. todos os jogadores empatados.

3. Lógica do Jogo

3.1. Representação do estado de jogo

O tabuleiro de jogo é representado internamente como uma lista de listas, denominada *board*. Cada elemento do *board*, será uma lista que representa uma linha do tabuleiro de jogo. Cada elemento de uma linha será outra lista, contendo toda a informação de uma casa do tabuleiro.

Um exemplo visual da composição do tabuleiro para o início de um jogo é a seguinte, em que todas as naves de cada jogador começam nos respetivos sistemas-mãe:



Para a representação em Prolog, a informação é representada segundo uma série de palavras-chave, cujo significado se traduz para algo relativo ao tabuleiro:

Tipos de sistemas
home - sistema-mãe
starX - sistema com X estrelas
nebula - sistema nebuloso
emptyS - sistema vazio
wormhole - buraco de minhoca
blackhole - buraco negro

Donos do sistema
player1 - jogador um
player2 - jogador dois
free - sistema livre

Naves
Jogador um:
-shipA, shipB, shipC, shipD
Jogador dois:
-shipW, shipX, shipY, shipZ

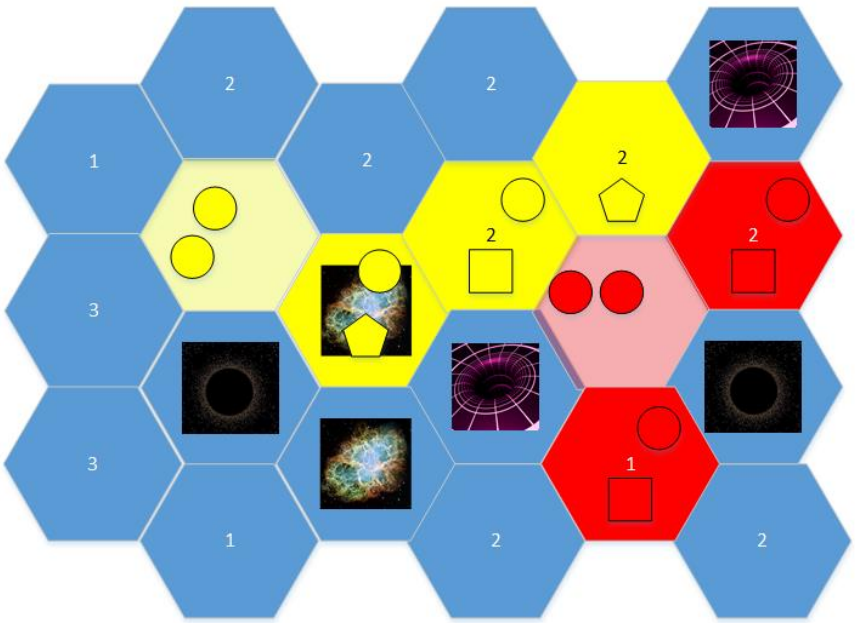
Construções
trade - estação de comércio
colony - colónia
none - nenhuma

A representação de uma casa do tabuleiro em prolog é uma lista de elementos: **[<tipo de sistema>, <dono>, <lista de naves>, <construções>]**.

O tabuleiro da imagem acima é representado em Prolog por:

```
board([
    [[star2, free, [], none], [star2, free, [], none], [wormhole]],
    [[star1, free, [], none], [star2, free, [], none], [star2, free, [], none]],
    [[home, player1, [shipA, shipB, shipC, shipD], none], [star2, free, [], none],
[star2, free, [], none]],
    [[star3, free, [], none], [nebula, free, [], none], [home, player2, [shipW,
shipX, shipY, shipZ], none]],
    [[blackhole], [wormhole], [blackhole]],
    [[star3, free, [], none], [nebula, free, [], none], [star1, free, [], none]],
    [[star1, free, [], none], [star2, free, [], none], [star2, free, [], none]]
    ]
    ).
```

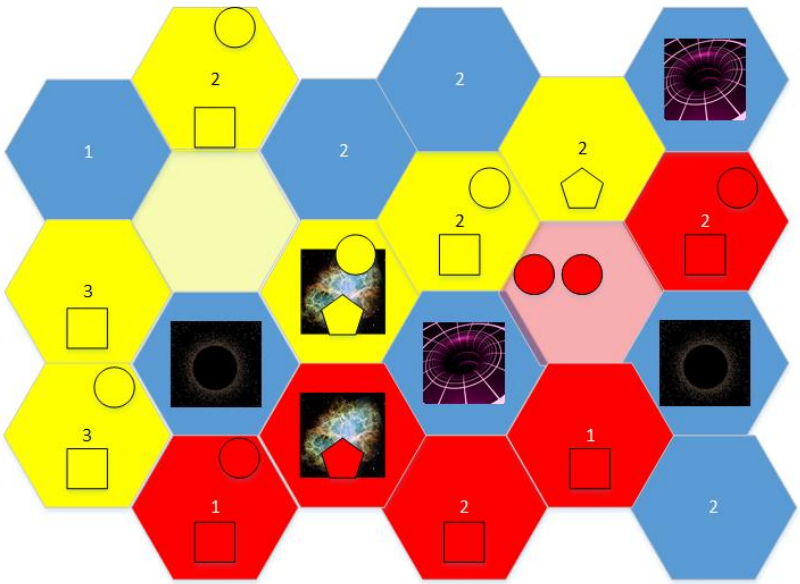
De seguida mostra-se uma situação para a qual o jogo poderá desenvolver-se, começando o amarelo primeiro. À medida que as naves avançam e conquistam sistemas e nébulas, constroem-se colónias ou estações de comércio.



E a representação em Prolog é:

```
board([
    [[star2, free, [], none], [star2, free, [], none], [wormhole]],
    [[star1, free, [], none], [star2, free, [], none], [star2, player1, [], trade]],
    [[home, player1, [shipC, shipD], none], [star2, player1, [shipA], colony], [star2, player2, [shipW], colony]],
    [[star3, free, [], none], [nebula, player1, [shipB], trade], [home, player2, [shipY, shipZ], none]],
    [[blackhole], [wormhole], [blackhole]],
    [[star3, free, [], none], [nebula, free, [], none], [star1, player2, [shipX], colony]],
    [[star1, free, [], none], [star2, free, [], none], [star2, free, [], none]]
    ]
    ).
```

Uma possível situação de fim de jogo resulta da falta de opções de jogada pelo jogador vermelho, que não consegue movimentar nenhuma das suas naves, enquanto o jogador amarelo consegue. Assim, perde o jogador vermelho. Na imagem de exemplo, as naves ainda estão presentes para mostrar as suas últimas posições:



Em Prolog:

```
board([
    [[star2, player1, [shipC], colony], [star2, free, [], none], [wormhole]],
    [[star1, free, [], none], [star2, free, [], none], [star2, player1, [], trade]],
    [[home, player1, [], none], [star2, player1, [shipA], colony], [star2, player2, [shipW], colony]],
    [[star3, player1, [], colony], [nebula, player1, [shipB], trade], [home, player2, [shipY, shipZ], none]],
    [[blackhole], [wormhole], [blackhole]],
    [[star3, player1, [shipD], trade], [nebula, player2, [], trade], [star1, player2, [], colony]],
    [[star1, player1, [shipX], colony], [star2, player2, [], colony], [star2, free, [], none]]
    ]
    ).
```

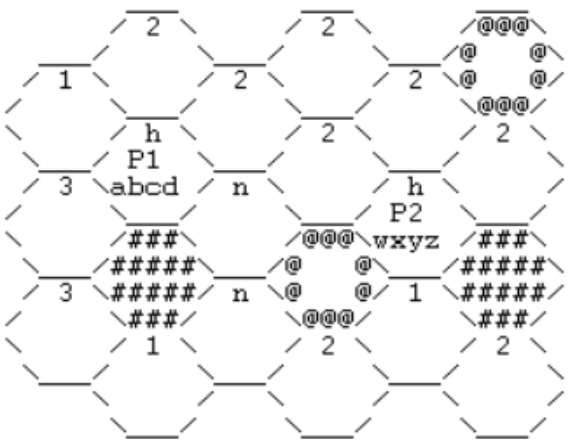
3.2. Visualização do tabuleiro

Para visualizar o tabuleiro, são geradas formas hexagonais recursivamente, que representam cada casa do tabuleiro. É possível determinar a dimensão do tabuleiro e ele gera-se automaticamente. Assim, a dimensão do nosso tabuleiro depende da dimensão da lista de listas que representa o jogo.

Se o número de elementos da lista *board* for 3, o tabuleiro terá 3 linhas. O número de colunas é determinado pelo número de elementos que uma linha tem.

A informação de cada casa é declarada na matriz de representação de jogo como uma lista dos diversos elementos que uma casa pode conter e é escrita recursiva e automaticamente a partir da matriz, preservando o espaçamento entre hexágonos.

A visualização do tabuleiro é feita recorrendo ao predicado `display_board(Board)`. Segue-se um exemplo do tabuleiro que representa o estado de jogo inicial descrito acima e respetiva visualização:



Os números que surgem em cada casa são o número de planetas que aquele sistema possui. Quando em vez de um número aparece “n” o sistema é nebuloso e quando aparece “h” é um sistema mãe.

Buracos de minhoca aparecem representados por símbolos “@” que formam um círculo na casa e buracos negros por “#” que preenchem toda a célula.

As letras na 3ª linha do hexágono (a, b, c, d, w, x, y, z) representam as naves que se encontram nessa casa.

“P1” indica que o sistema pertence ao jogador 1 e “P2” ao jogador 2.

Quando é construída uma estação de comércio, ela aparece na casa representado por “[T]”. Se em vez disso se construir uma colónia, é mostrado “(C)”.

3.3. Lista de jogadas válidas

No *Small Star Empires*, uma jogada é caracterizada pelo movimento de uma nave pertencente ao jogador. Para obter as jogadas válidas numa determinada altura do jogo, fizeram-se várias verificações por passos.

A primeira camada de verificações trabalha sobretudo a nível das coordenadas da matriz, isto é, vai buscar a casa de origem e a casa de destino da nave, comparando coordenadas para concluir se a mudança é válida ou não puramente em termos geométricos:

- Verificar se é possível obter uma casa a partir das suas coordenadas, isto é, se ela pertence ao tabuleiro.
    - Predicado: `getPiece(+Row, +Column, +Board, -Piece):- ...`

- Verificar se a casa está numa direção válida, considerando os lados do hexágono. Isto restringe o movimento a uma única direção.
    - Predicado: `verifyValidGeometricDirection(+Xi, +Yi, -Xf, -Yf):- ...`

A segunda camada lida com as condições impostas pelo próprio jogo. Inclui detetar se a trajetória que a nave faz está bloqueada e se a casa de destino é válida:

- Verificar se a trajetória da nave não interseta nenhuma casa inválida, ou seja, se apenas passa por sistemas vazios ou pertencentes ao próprio jogador.
    - Predicado: `unobstructedPath(+Board, +Player, +Xi, +Yi, +Direction, +NumberOfCells, -Xf, -Yf):- ...`

- Verificar se a casa destino é válida. Este predicado em particular é feito para falhar caso a casa não seja válida.
    - Predicado: `checkValidLandingCell(+Cell):- ...`

Existe um predicado que junta tudo isto e obtém a casa destino da nave segundo os parâmetros inseridos pelo jogador ou que falha, caso a casa não seja válida:

```
moveNCellsInDirection(+Xi, +Yi, +Direction, +NumberOfCells, -Xf, -Yf):-
....
```

3.4. Execução de jogadas

A execução de uma jogada é, a nível lógico, equivalente a retirar uma nave do seu sistema de origem e colocá-la na célula destino. Ao "aterrar" num sistema, o jogador passa a dominá-lo, pelo que é obrigado a colonizá-lo. Sendo assim, quando uma nave é movimentada, é necessário fazer as alterações necessárias para colonizar o sistema.

O jogador decide para onde a nave vai e como colonizar o sistema onde ela aterra. Para receber informação do jogador, é invocado o seguinte predicado:

```
readPlayerInput(Board, WhoIsPlaying, OldPiece, NewPiece,
PieceToMove, PieceToMoveRow, PieceToMoveColumn, DestinationPiece,
DestinationRow, DestinationColumn):- ...
```

Este predicado é equivalente a um ciclo de *input* de informações que o jogador deve inserir para fazer a sua jogada. É pedida a nave a deslocar, a direção da trajetória e o número de casas que se pretende viajar. Caso a jogada não seja válida, é pedida a inserção de uma nova jogada por parte do jogador, ou seja, este predicado também serve de **verificação** de jogada.

O predicado que executa a alteração do tabuleiro, depois de a alteração ter sido validada é o seguinte:

```
updateBoard(Board, OldPiece, NewPiece, PieceToMove, PieceToMoveRow,
PieceToMoveColumn, DestinationPiece, DestinationRow, DestinationColumn,
UpdatedBoard):- ...
```

3.5. Avaliação do tabuleiro

O valor de uma jogada do tabuleiro é influenciada por três fatores:

- Valor dos sistemas que se possuem;
- Número de sistemas nebulosos conquistados;
- Valor dos sistemas adjacentes aos colonizados.

Como tal, para decidir uma jogada, a inteligência artificial examina todo o tabuleiro, extrai as casas válidas para jogar a partir das funções de validação e depois verifica, segundo os critérios acima, qual é a melhor opção:

```
searchMaxScore(Board, [X|Xs], [Y|Ys], CurrentMaxScore, Building,
CellToPlayX, CellToPlayY, OriginCellX, OriginCellY, UpdatedBoard):- ...
```

3.6. Final do jogo

O jogo termina se os dois jogadores não tiverem opções para movimentar as naves e se já não existem mais colónias ou estações de comércio disponíveis.

Devido à forma como foi implementado o ciclo de jogo, o predicado de jogo é algo peculiar no seu funcionamento.

```
endGame(Board):- …
```

Este predicado está construído para ser bem-sucedido se o jogo **não** tiver acabado. Isto porque o ciclo de jogo foi implementado de forma recursiva e só pára quando há uma falha. Essa falha é ativada pelo endGame, que falha quando o jogo acaba.

Por sua vez, o ciclo de jogo é bem-sucedido se falhar, de forma a que as pontuações sejam mostradas no final.

3.7. Jogada do Computador

O computador pode jogar de dois modos: ou jogador VS computador, ou computador VS computador.

O computador escolhe uma das suas naves ao acaso, através do predicado chooseShipToMove(Board, OriginCellX, OriginCellY), que recebe um tabuleiro e retorna a posição X e Y da casa onde se situa uma das naves a que o computador tem acesso.

De seguida calcula o X e o Y de todas as casas para que esse navio pode mover-se, através do predicado getAllPossibleCellsToMove(OpponentPlayer, Board, X, Y, ListX, ListY), retornadasem ListX todas as posições X e em ListY todas as posições Y possíveis de viajar. Este predicado já restringe as posições retornadas, uma vez que só contabiliza casas válidas, ou seja, apenas casas que não são buracos negros, que não estão ocupadas, etc…

Depois é executado o predicado searchMaxScore(Board, [X|Xs], [Y|Ys], CurrentMaxScore, Building, CellToPlayX, CellToPlayY, OriginCellX, OriginCellY, UpdatedBoard), onde inicialmente se determina qual é que é a casa que, viajando para lá, permite obter uma maior pontuação. Quando essa casa é determinada, o movimento da peça é realizado e o tabuleiro é atualizado.

No modo computador VS computador, ambos os diferentes “lados” do computador usam este método para tentarem vencer.

4. Interface com o Utilizador

O programa inicia com o menu principal, onde é possível escolher de entre os 3 modos de jogo disponíveis, onde é possível aceder através do número mostrado.

Por exemplo, para iniciar um jogo jogador VS jogador, o utilizador introduziria “1.” (sem as aspas).

```
*********************************************************************************************
********************************** SMALL STAR EMPIRES ***************************************
*********************************************************************************************


***** Main Menu *****

Player VS Player --> Type 1
Player VS AI --> Type 2
AI VS AI --> Type 3

Select Game Mode
|:
```

Caso tenha sido escolhido o modo jogador VS jogador, é perguntado aos jogadores qual deles é o mais velho. Isto é importante, pois segundo as regras o jogador mais novo é o primeiro a começar. Está implícito que os jogadores têm de concordar quanto à numeração (quem é o jogador 1 e quem é o 2).

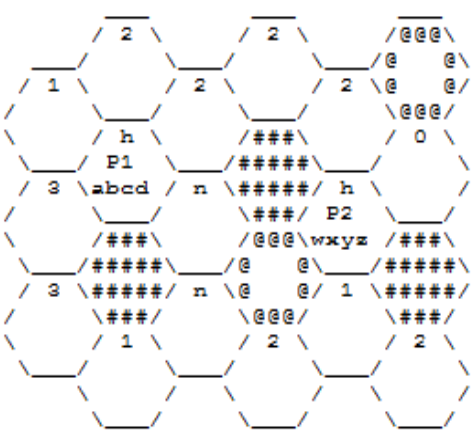
```
Who is the youngest player?
Player 1 --> 1
Player 2 --> 2
|:
```

Logo de seguida o jogo começa. É apresentada uma mensagem indicativa de que o jogo começou, a quem pertence o turno atual, o tabuleiro atual do jogo e informação de quantas colônias e estações de comércio esse jogador tem disponível.

De seguida são feitas perguntas ao jogador sobre a movimentação das suas peças, tais como a nave que este quer mover, em que direcção (norte, sul, noroeste, nordeste, sudoeste, sudeste), quantas casas quer viajar e o tipo de edifício que deseja construir.

```
********** THE BATTLE IS ON! **********

*************** Player 1 turn ***************
```



```
You have: 0 trade station(s) and 20 colony(ies)

Select ship
|: a.

Select direction to travel (n, s, nw, ne, sw, se)
|: n.

Select number of cells to travel
|: 1.

Player 1, what building would you like to construct?
   t --> Trade Station
   c --> Colony
|: c.
```

Depois de tudo isso, são impressas linhas brancas para tirar informação do turno anterior do ecrã.

```
**************** Player 2 turn ****************

       ___         ___         ___
      / 2 \       / 2 \       /@@@\
  ___/ P1  \___/       \___/@    @\
 / 1 \a     / 2 \       / 2 \@    @/
/     \(C)/      \___/       \@@@/
\     / h \      /###\       / O \
 \___/ P1  \___/#####\___/       \
 / 3 \bcd   / n \#####/ h \      /
/     \___/       \###/ P2  \___/
\     /###\       /@@@\wxyz /###\
 \___/#####\___/@    @\___/#####\
 / 3 \#####/ n \@    @/ 1 \#####/
/     \###/       \@@@/       \###/
\     / 1 \       / 2 \       / 2 \
 \___/       \___/       \___/       \
     \       /   \       /   \       /
      \___/       \___/       \___/

You have: 0 trade station(s) and 20 colony(ies)

Select ship
|: y.

Select direction to travel (n, s, nw, ne, sw, se)
|: ne.

Select number of cells to travel
|: 1.

Player 2, what building would you like to construct?
   t --> Trade Station
   c --> Colony
|: c.
```

A partir daqui o ciclo é sempre o mesmo até o jogo terminar (todas as casas estarem preenchidas ou todos os navios estarem cercados).

5. Conclusões

O jogo utilizado para este projeto requeria um elevado grau de planeamento, que não foi correspondido no desenvolvimento desta aplicação. Era necessário implementar, por exemplo, um melhor sistema de coordendas e predicados mais simples, que não aumentassem a complexidade do código à medida que ele se expandia.

Ainda assim, foi bem-sucedida a implementação de uma inteligência artificial capaz de decidir qual a melhor jogada. Verificaram-se as vantagens de uma linguagem com um paradigma lógico e declarativa como o Prolog, em relação às mais divulgadas linguagens orientadas por objetos no que toca a decisões face a um certo panorama.