Exploração de Marte

Agentes e Inteligência Artificial Distribuída

Universidade do Porto

Faculdade de Engenharia

FEUP

Dezembro 2016

Grupo T05_4
Marina Camilo - up201307722 - up201307722@fe.up.pt
Diogo Ferreira - up201502853 - diogoff@fe.up.pt
Ângela Cardoso - up200204375 - angela.cardoso@fe.up.pt

Conteúdo

Capítulo 1

Objetivo

1.1 Descrição do cenário

No âmbito da unidade curricular de Agentes e Inteligência Artificial Distribuída, o nosso grupo propôs-se a implementar um Sistema Multi-Agente para simulação de um cenário de extração de minérios em Marte. Para tal, é necessário descobrir os minérios, extraí-los e transportá-los para a base. Sendo assim, no sistema pretendido existem três tipos de Agentes:

- *Spotter* – Procura fontes de minérios e inspeciona-as para determinar se podem ser exploradas.
- *Producer* – É chamado a uma fonte de minério por um *spotter* para extrair o máximo de minério possível nessa fonte.
- *Transporter* – É alocado pelo *producer* para carregar o minério obtido para a base.

De forma a facilitar a procura, todos os agentes podem localizar fontes de minérios e enviar a sua localização para os *spotters* que os analisarão. A escolha do *producer* por parte do *spotter* segue um protocolo de negociação. A alocação dos *transporters* a uma determinada fonte segue também um protocolo de negociação, iniciado pelo *producer*. Esta alocação, terá em conta a quantidade de minério a transportar, de modo a determinar mais corretamente o número necessário de *transporters*.

1.2 Objetivos do trabalho

Um dos objetivos deste trabalho é implementar os agentes de forma a que a simulação da exploração seja tão eficiente quanto possível.

O nosso principal objetivo é utilizar este projeto como forma de melhor interiorizar os conceitos dos Sistemas Multi-Agente, nomeadamente ganhando uma maior

familiaridade com as plataformas que permitem implementar e simular estes sistemas.

Capítulo 2

Especificação

2.1 Identificação e caracterização dos agentes

arquitetura, comportamento, estratégias

processos de raciocínio - novo no relatório final

Tal como descrito acima, o nosso sistema tem três tipos de agentes. De seguida descrevem-se mais detalhadamente estes agentes e a forma como os implementámos.

2.1.1 *Spotter*

Cada *spotter* tem a sua região conexa do espaço para explorar. Uma vez definida essa região, a estratégia do *Spotter* é simples, encaminha-se para o inicio da região e inspeciona cada célula consecutivamente, determinando se contém minério extraível ou não e guarda registo desta informação. Sempre que encontra minério extraível, entra em contacto com todos os *producers* para negociar e decidir qual deles será chamado para extrair o minério.

2.1.2 *Producer*

Os *producers* entram em ação assim que é encontrado minério extraível por um *spotter*. Nesse momento cada um deles terá que determinar o custo necessário para se deslocar ao local e extrair o minério. Esse esforço corresponde à distância (em passos) a que o *producer* se encontra do local no final de completar todas as tarefas em mãos, mais o tempo (em passos) que ele demorará a terminar a sua tarefa atual, caso esteja a extrair minério, mais o tempo que demorarão todas as outras tarefas com as quais já se tenha comprometido. Para esse efeito, cada *producer* deve guardar uma fila contendo informação sobre as suas extrações, em particular o tempo que cada uma delas demora, incluindo a deslocação da sua posição anterior até ao local da extração.

2.1.3 *Transporter*

Os *transporters* iniciam a sua atividade quando são contactados para transportar minério. À semelhança do que acontece com os *producers*, nesse momento terão de calcular o esforço que necessitam para se deslocarem ao local tendo em conta a sua capacidade final. Quando informarem os *producers* do esforço que necessita para chegar até um local terá que considerar qual a capacidade que terá disponível quando lá se deslocar. O *producer* escolhe os primeiros *transporters* que totalizam a capacidade disponível da carga que se pretende transportar, regressando à base apenas quando atingem a carga máxima.

2.2 Protocolos de interação

Existem 3 fases que se destacam no decorrer da simulação: repartição do espaço entre os *spotters* existentes, requerimento de um *producer* no momento de descoberta de um minério e, finalmente, o escalonamento dos *transporters* para a recolha dos fragmentos resultantes de um minério.

incluir diagramas de interação UML, se aplicável

2.2.1 Divisão de espaços

No inicio da simulação a nave-mãe divide o espaço disponível em linhas de acordo com o número de *spotters* existentes e entrega cada um dos dados dos sub-espaços a um *spotter* distinto de modo a que este o possa depois confirmar com os restantes *spotters* que pode realmente ficar responsável pela área. A Figura 2.1 demonstra como a nave-mãe faz esta divisão.

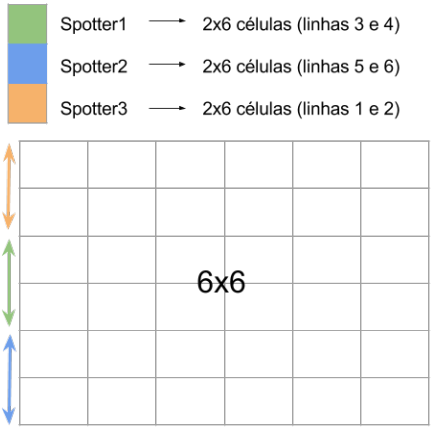

Figura 2.1: Repartição de espaços

Uma vez recebida a sugestão pela nave-mãe o *spotter* inicia então o protocolo de negociação **fipa-propose** (Figura 2.2) com os restantes *spotters*. Este protocolo define como propôr algo a outros agentes e tratar das suas respostas.

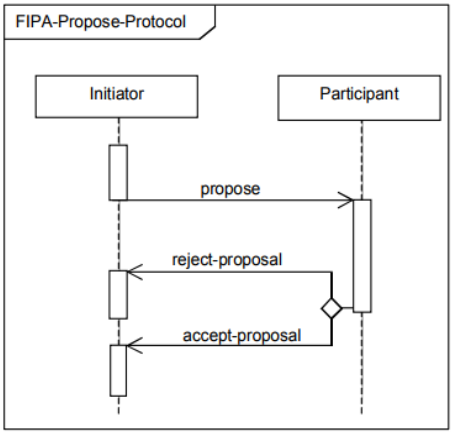

Figura 2.2: Repartição de espaços

Após a receção de todas as respostas é enviada uma última mensagem de volta a informar que a área ficou então afeta ao *spotter* que iniciou a proposta. Esta última fase é complementar ao protocolo. O protocolo **fipa-propose** é suportado pelas interfaces *ProposeInitiator* e *ProposeResponder* que um *spotter* implementa através de *behaviours* e pelas classes internas *RequestAreaBehaviour* e *AnswerCallBehaviour*, respetivamente. Finalmente, a receção da última mensagem de notificação é feita também através de um *behaviour* na classe interna *AcknowledgeAreaBehaviour*. A mensagem tem a performativa **inform** e no seu conteúdo uma string no formato **offset-height** em que *offset* indica a coordenada no eixo Y em que a área começa e *height* a altura da mesma. Os receptores desta mensagem guardam então internamente num mapa esta afetação para possíveis futuras propostas.

2.2.2 Afetação de *producers*

Uma vez encontrado minério, o *spotter* primeiro verifica se é exploravél e, se o for, é necessário chamar um *producer* para o extrair. O *spotter* envia então a posição do minério a todos os *producers*. Estes respondem com o valor do esforço de que necessitarão para se deslocar até ao local. O *spotter* escolhe o *producer* com o menor esforço e comunica com ele, pedindo para confirmar a afetação. Caso seja recusado, porque o *producer* foi afeto a outro minério entretanto, o *spotter* repete o processo desde o início. O esforço é calculado pela distância da última posição planeada do *producer* ao minério recém descoberto. Toda esta comunição segue o protocolo **fipa-contractnet** (Figura suportado pelas interfaces ContractNetInitiator e ContractNetResponder que os *spotters* e *producers* implementam, respetivamente.

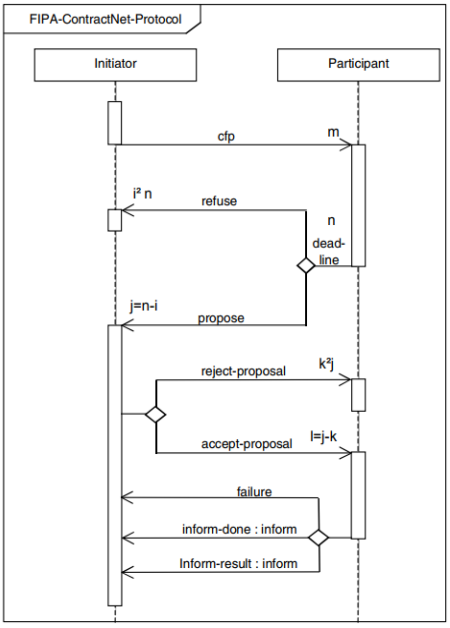

Figura 2.3: FIPA-ContractNet

A Figura 2.4 exemplifica o processo de escolha de *producers* para os minérios encontrados pelos *spotters* 1 e 2. Neste caso, ficou o *producer* 2 afeto ao minério do *spotter* 1 e o *producer* 1 afeto ao minério do *spotter* 2.

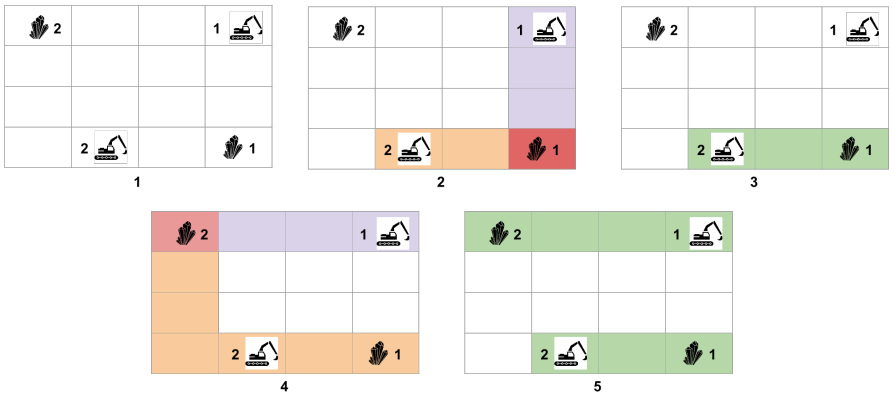

Figura 2.4: Exemplo de afetação de *producers*

2.2.3 Afetação de *transporters*

Após a extração do minério é necessário transportá-lo para a nave-mãe. O *producer* que acabou de extrair o minério tem que selecionar um ou mais *transporters*, do mesmo modo que o *spotter* seleciona um *producer*. Cada *transporter* comunica o valor do esforço e o minério que consegue transportar possibilitando o *producer*

de escalonar os diferentes agentes. Este processo segue também o processo **fipa-contractnet** (Figura 2.3) suportado pelas interfaces ContractNetInitiator e ContractNetResponder implementados nos *producers* e *transporters*, respetivamente.

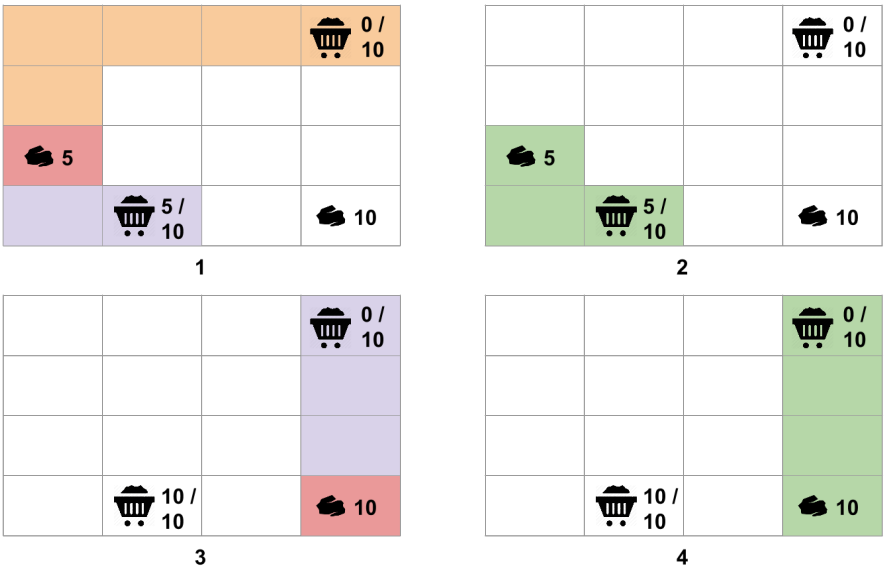

Figura 2.5: Exemplo de afetação de *transporters*

A Figura 2.5 exemplifica o processo de escolha de *transporters* para os minérios extraídos por dois *producers*. Consideramos ainda as quantidades de minério extraídas, a capacidade e carga corrente dos *transporters*. Estas restrições irão influenciar a forma como se negoceia a afetação dos *transporters* numa fase mais avançada da implementação.

Capítulo 3

Desenvolvimento

capítulo novo mas as secções não são completamente novas

3.1 Plataformas utilizadas

brevíssima descrição o que é - é capaz de ser melhor reduzir isto

3.1.1 JADE

O JADE, *Java Agent DEvelopment Framework*, é um software que permite desenvolver sistemas baseados em agentes. Os agentes são distribuídos por *containers* que podem estar em máquinas diferentes, cada um utiliza uma *thread*.

O JADE é uma ferramenta totalmente escrita em JAVA, que suporta troca de mensagens ACL, seguindo a especificação FIPA. Além disso, possui um sistema de gestão de agentes e um sistema de páginas amarelas. Como os agentes estão distribuídos por contentores que podem estar em máquinas diferentes, permite ter agentes remotos. Os agentes podem migrar entre contentores e ser clonados. Esta plataforma possui ainda uma série de ferramentas que simplificam a administração e o desenvolvimento de aplicações, tais como:

- agente de monitorização remota - interface gráfico para monitorizar a atividade dos agentes;
- agente pateta - permite trocar mensagens com outros agentes;
- agente inspetor - permite inspecionar outros agentes;
- agente introspetivo - permite monitorizar o ciclo de vida de um agente.

A implementação de agentes em JADE é feita recorrendo a comportamentos, que determinam as tarefas a executar pelos agentes consoante o contexto.

3.1.2 Repast 3

O Repast 3, *Recursive Porous Agent Simulation Toolkit*, é uma ferramenta de simulação baseada em agentes, que permite construir simulações locais à máquina com diversos agentes. O processamento de cada agente é distribuído por *threads*.

O Repast 3 suporta simulação de espaços físicos, representação 2D e 3D e análise em tempo real. Possui uma barra de ferramentas para controlar as simulações, um interface gráfico para manipular os parâmetros. Permite recolher dados em vários formatos, incluindo vários tipos de gráficos. A interação entre os agentes pode ser visualizada graficamente. Os espaços físicos simulados podem ser de variados tipos, como por exemplo, grelhas hexagonais ou retangulares, espaços contínuos ou redes. Além de poderem ser corridas com recurso ao interface gráfico, as simulações podem também ser lançadas em *batch*.

O simulador discreto considera unidades de tempo - *ticks* ou passos. Os eventos são planeados para ocorrer em *ticks* específicos e assim é respeitada a ordem dos acontecimentos. Uma simulação considera um conjunto de agentes, cujos comportamentos são controlados recorrendo ao plano.

3.1.3 SAJaS

O SAJaS, *Simple API for JADE-based Simulations*, é uma ferramenta que se propõe servir de ponte entre o desenvolvimento e a simulação de Sistemas Multi-Agente. Desta forma, através do SAJaS, podemos tomar vantagem dos benefícios do JADE e do Repast 3.

O SAJaS permite construir um Sistema Multi-Agente tal como é feito em JADE, existindo até uma ferramenta (MASSim2Dev) que traduz código JADE para SAJaS (e vice-versa), alterando as classes importadas. Ao contrário do JADE, permite ainda, recorrendo para isso ao Repast 3, a simulação deste tipo de sistemas. Além disso, o SAJaS permite obter melhores performances do que numa simulação construída em JADE, que teria que considerar algum tipo de agente representado o 'mundo'.

3.1.4 Realce das funcionalidades relevantes para o trabalho

Com o suporte do JADE são feitos os protocolos de comunicação entre os diferentes agentes, utilizando mensagens ACL. Usando o Repast 3 torna-se fácil simular um espaço físico, popular o espaço com os agentes construídos em JADE, desenhá-los e finalmente vê-los em ação, obtendo gráficos relativos à sua performance. O SAJaS permite-nos recorrer a estas duas ferramentas simultaneamente, construindo e simulando o nosso sistema de forma eficiente.

3.1.5 Ambiente de desenvolvimento

Os elementos do grupo utilizam Sistemas Operativos (SO) e Ambientes de Desenvolvimento Integrado (IDE) diferentes, de acordo com as suas preferências. Para tal, é especialmente útil as plataformas utilizadas serem baseadas em Java. Assim como,

o facto de apenas ser necessário adicionar os respetivos módulos `jar` ao projeto, para que qualquer IDE compile e execute a aplicação devidamente. Desta forma, a combinação SO - IDE utilizada por cada elemento, foi:

- Ângela Cardoso - macOS Sierra - IntelliJ IDEA;
- Diogo Ferreira - Manjaro Linux - NetBeans;
- Marina Camilo - Windows7 - Eclipse Neon;

3.2 Estrutura da aplicação

A aplicação está dividida em 2 packages: **main** e **agents**.

3.2.1 main

Nesta *package* encontram-se as classes gerais da simulação como o modelo e as variáveis de ambiente.

Environment

Aqui estão definidas um conjunto de variáveis estáticas que definem o ambiente de simulação.

MarsModel

O modelo da simulação. Inclui uma matriz bi-dimensional que representa o espaço físico e cada célula contém uma lista de agentes naquela posição.

MarsNode

Representa um nó no modelo. Inclui uma referência ao agente que representa.

Simulation

O ponto de entrada da simulação.

3.2.2 agents

Nesta *package* encontram-se as classes dos agentes a Figura 3.1 é demonstra a estrutura das mesmas.

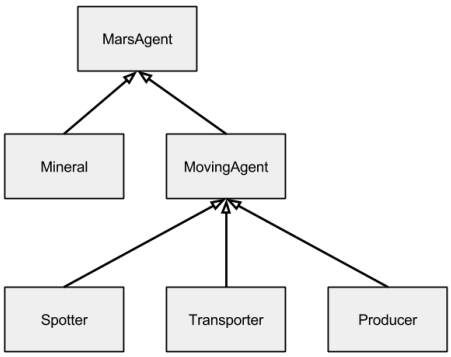

Figura 3.1: UML dos agentes

MarsAgent

Define a base de cada agente da simulação. Inclui métodos reutilizáveis como **move** e **translate** que permitem mover o agente.

MovingAgent

Define a base para cada agente que planeia os seus movimentos. Permite acumular movimentos que são executados sequencialmente.

Mineral

Representa um mineral. Não define qualquer *behaviour*.

Spotter

Representa um *spotter* no modelo. Implementa 6 *behaviours*:

- RequestAreaBehaviour - Inicia o protocolo **fipa-propose** com os restantes *spotters*.
- AnswerAreaRequestBehaviour - Responde às propostas iniciados pelos restantes *spotters*.
- AcknowledgeAreaBehaviour - Recebe as mensagens finais dos *spotters* quando os mesmos ficam afetos a uma área e guarda essa mesma informação.
- MoveBehaviour - Executa o plano de movimento do agente.
- ScanBehaviour - Analisa os minerais quando estes estão próximos o suficiente.

- RequestProducerBehaviour - Inicia o protocolo **fipa-contractnet** com os *producers*.

Producer

Representa um *producer* no modelo. Implementa 3 *behaviours*:

- RoutineBehaviour - Executa o plano de movimento do agente.
- AnswerCallBehaviour - Responde às mensagens do protocolo **fipa-contractnet** provenientes dos *spotters*.
- RequestTransporterBehaviour - Inicia o protocolo **fipa-contractnet** com os *transporters*.

Transporter

Representa um *transporter* no modelo. Implementa 2 *behaviours*:

- RoutineBehaviour - Executa o plano de movimento do agente.
- AnswerCallBehaviour - Responde às mensagens do protocolo **fipa-contractnet** provenientes dos *producers*.

módulos, diagrama de classes...

3.3 Detalhes relevantes da implementação

Capítulo 4

Experiências

Para examinar a qualidade da nossa implementação, desenhamos algumas experiências, que são apresentadas em seguida. Tivemos alguma dificuldade em realizar estas experiências, porque não fomos capazes de as automatizar completamente, devido à natureza gráfica do `Repast 3`.

4.1 Objetivos de cada experiência

4.1.1 Complexidade da implementação

Estas primeiras experiências têm o objetivo de determinar a qualidade geral da implementação, nomeadamente investigando a sua complexidade temporal. Uma vez que há vários parâmetros no nosso modelo, para determinar a sua complexidade, é necessário fixa-los e variar apenas um deles de cada vez. Os parâmetros que consideramos significativos são:

- *size* - tamanho do lado do quadrado que representa a área a explorar;
- *minerals* - número de minérios que serão gerados na área;
- *spotters* - número de *spotters* que serão disponibilizados;
- *producers* - número de *producers* que serão disponibilizados;
- *transporters* - número de *transporters* que serão disponibilizados.

Outros parâmetros, tal como a probabilidade de um dado minério ser extraível, também são relevantes. No entanto, não nos parecem tão importantes como os mencionados acima. De facto, em todas as experiências, decidimos que os minérios seriam sempre extraíveis, por forma a facilitar a análise dos resultados.

Em relação aos parâmetros que destacamos, para que as experiências não se prolongassem demasiado, no que toca a complexidade, estudamos apenas *size* e *minerals*, determinando o número de *ticks* que o modelo demora a correr, conforme varia um, e apenas um, destes parâmetros.

4.1.2 Percentagem de utilização dos agentes

Por um lado, parece claro que ter mais agentes leve a uma exploração mais eficiente, especialmente quando *size* e *minerals* são elevados. Por outro lado, num contexto de simulação de acordo com a realidade, ter mais agentes é mais dispendioso. Sendo assim, consideramos relevante determinar, para *size* e *minerals* fixos, qual o número ideal de cada tipo de agente. Para isso, fizemos variar *spotters*, *producers* e *transporters* cada um por sua vez, avaliando o número total de passos dados por cada agente, assim como o número de *ticks* em que cada um deles está inativo.

4.1.3 Comparação de diferentes técnicas

Como não chegamos a implementar diferentes técnicas para resolver o problema, não realizamos experiências que permitissem compará-las. No entanto, decidimos ainda assim expor aqui a forma como o faríamos.

A primeira forma de comparação, seria comparar o número de *ticks* que cada técnica usa fixados os vários parâmetros. A experiência mais rápida, seria considerada melhor nestas experiências.

Outra forma de comparação, seria considerar a quantidade total de passos que os vários agentes dão para completar a exploração em cada técnica. Neste caso, seria melhor a técnica que utilizasse menos passos.

Finalmente, também consideramos que seria adequada para este efeito a determinação do tempo decorrido entre deteção, extração e transporte de um minério. Neste caso, seria melhor a técnica que fosse mais rápida.

4.2 Resultados

4.2.1 Complexidade da implementação

Começamos por variar *size* mantendo fixos os restantes parâmetros, nomeadamente, utilizamos 1 minério e 5 agentes de cada tipo. Com o aumento do tamanho do quadrado a explorar, os *spotters* são os últimos agentes a terminar a sua tarefa. Isto porque têm que percorrer toda a área, enquanto que os restantes agentes apenas têm que se deslocar até ao minério, assim que este for encontrado.

Os resultados obtidos podem ser observados no gráfico da Figura 4.1, que sugere complexidade quadrática do nosso modelo em função do tamanho. Ora, isto faz imenso sentido uma vez que os *spotters* têm que percorrer todo o tabuleiro e a sua área é igual ao quadrado do tamanho do lado.

Quando variamos o número de minérios, mantendo o tamanho constante a 30 e 5 agentes de cada tipo, deixam de ser os *spotters* os últimos a terminar a exploração e passam a ser os *transporters*. Isto é razoável, se pensarmos que com mais minérios para extrair, os *producers* terão de continuar a extração mesmo depois de todos os *spotters* terminarem de percorrer toda a área, pois terão acumulado os vários minérios a extrair nas suas filas. Já os *transporters*, nunca podem terminar antes

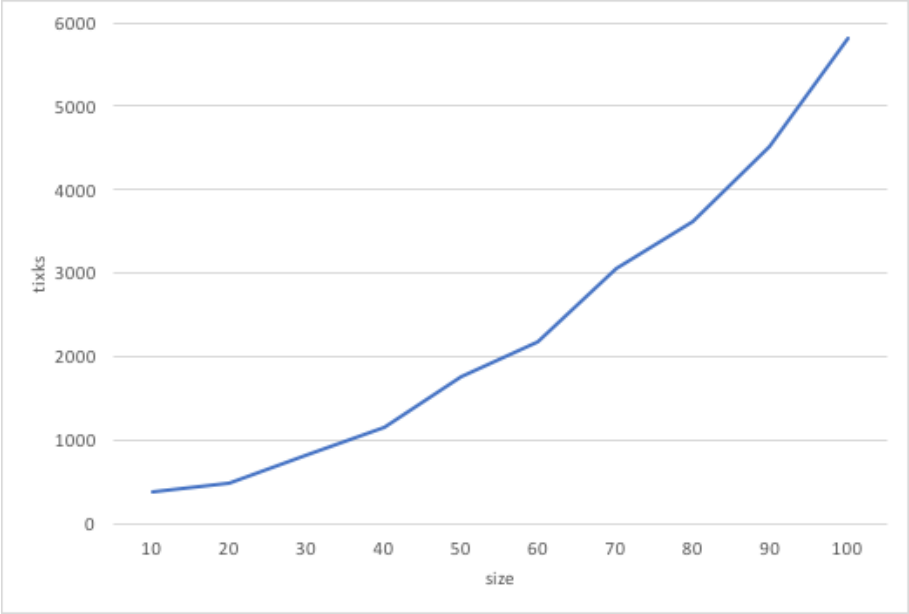

Figura 4.1: Variação do tempo de exploração em função do tamanho do quadrado a explorar

dos *producers*, dado que terão de transportar o minério apenas depois de este ser extraído.

Os resultados desta segunda experiência podem ser observados no gráfico da Figura 4.2, que sugere complexidade linear do nosso modelo em função do número de minérios. Faz sentido que assim seja, dado que para extrair e transportar um mineral é gasto no máximo o tempo de ida e volta até lá mais o tempo de extração, que é igual à quantidade de minério, que por sua vez, apesar de não ser sempre igual, varia sempre entre os mesmos valores que mantivemos ao longo das experiências.

4.2.2 Percentagem de utilização dos agentes

Começamos por variar o número de *spotters*, mantendo constantes o tamanho a 30, os minérios a 20, os *producers* a 5 e os *transporters* a 5. Os resultados obtidos estão disponíveis na tabela 4.1.

spotters	passos	inatividade	total
1	958	0	958
2	478	219	697
3	345	442	787
5	224	650	874
10	130	709	839

Tabela 4.1: Média de passos e inatividade dos *spotters* consoante o seu número

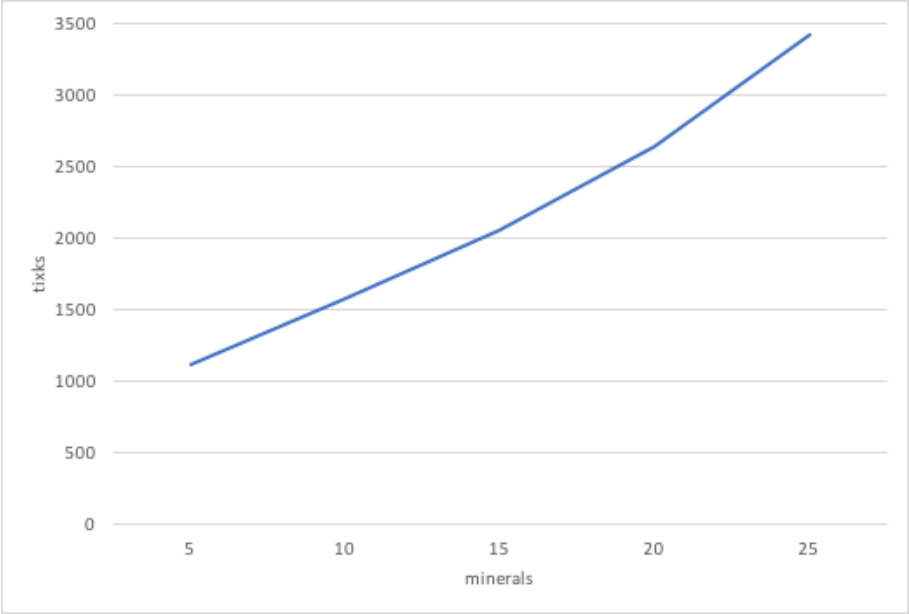

Figura 4.2: Variação do tempo de exploração em função do número total de minérios

É curioso observar que o número médio total de passos mais inatividade apenas baixa de 1 para 2 *spotters*, a partir daí vai subindo e baixa muito ligeiramente de 5 para 10 *spotters*. Mas isso deve-se essencialmente à inatividade, dado que quando os *spotters* terminam a exploração de toda a área, ficam à espera que os restantes agentes terminem as suas tarefas. De qualquer forma, não é útil utilizar um número muito elevado de *spotters*, porque a partir do momento em que são detetados os primeiros minérios, os restantes agentes começam a estar ocupados e a rápida deteção de minérios não se traduz em maior velocidade do modelo como um todo. Para esta configuração específica o número ideal de *spotters* parece ser 2, dado que apresenta o total mais baixo.

Variando o número de *producers* e mantendo constantes o tamanho a 30, os minérios a 20, os *spotters* a 5 e os *transporters* a 5, obtivemos os resultados que se apresentam na tabela 4.2.

producers	passos	inatividade	total
1	320	651	971
2	186	1294	1480
3	101	1661	1762
5	58	2262	2320
10	27	3680	3707

Tabela 4.2: Média de passos e inatividade dos *producers* consoante o seu número

Além dos dados apresentados, nesta experiência observamos que, com 10 *producers*, alguns que permanecem completamente inutilizados. Isto indica claramente

que para tamanho 30 e 20 minérios não faz sentido ter tantos *producers*. Outra observação interessante, é que o total de passos mais inatividade aumenta consistentemente com o número de *producers*, apesar de o número de passos baixar. Tudo indica que isto acontece devido às negociações entre os *spotters* e os *producers* para escolher aquele que irá realizar a extração. De facto, essas negociações são especialmente relevantes neste cenário, note-se que de 1 para 2 *producers* a inatividade quase duplica.

Finalmente, fizemos variar o número de *transporters*, mantendo constantes o tamanho a 30, os minérios a 20, os *spotters* a 5 e os *producers* a 5, obtivemos os resultados que se apresentam na tabela 4.3.

producers	passos	inatividade	total
1	920	1160	2080
2	395	1161	2056
3	245	1929	2175
5	174	2311	2485
10	85	3770	3855

Tabela 4.3: Média de passos e inatividade dos *transporters* consoante o seu número

Mais uma vez, 10 *transporters* é claramente demais, pois alguns não chegaram a realizar nenhuma tarefa. No entanto, ao contrário do que aconteceu com os *producers*, neste caso o total não aumenta sempre, tendo reduzido ligeiramente entre 1 e 2 *transporters*. Claro que se a área a explorar fosse maior e tivéssemos mais minérios, este efeito seria certamente mais notório, indicando que faz sentido ir aumentando o número de *transporters*, embora de forma conservadora. Mais uma vez, as negociações têm um peso grande, impedindo melhorias significativas quando há mais *transporters*.

Capítulo 5

Conclusões

5.1 Análise dos resultados das experiências

5.2 Desenvolvimento do trabalho

Capítulo 6

Melhoramentos

sugestões para melhoramentos a introduzir no programa

Alguns possíveis melhoramentos ao nosso projeto são:

- *Spotters* decidirem entre si a divisão do espaço a explorar;
- Ao ser escolhido um *producer* ou *transporter* ter em conta o número de tarefas que estão a realizar de forma a maximizar o esforço dos mesmos;
- Ser possível de alterar o percurso dos agentes. Exemplo: Se ao atribuir a um *transporter* uma tarefa e ficar em caminho ir a nave mãe descarregar, ele deveria descarregar e assim teria mais capacidade no final da tarefa;
- Atribuir os *transporters* de forma eficiente tendo em conta a capacidade dos mesmos. Exemplo: Se um existe um *transporter* que demora mais tempo mas tem capacidade para recolher o minério todo, deve ser selecionado em vez de atribuir pedaços de minério a vários *transporters* que no final demorem mais tempo a recolher o minério todo;

Capítulo 7

Recursos

7.1 Bibliografia

fazer os links aparecerem no texto

- Enunciado do trabalho
- Slides das aulas teóricas
- Slides sobre `JADE`
- Slides sobre `Repast 3`
- Página do `SAJaS`
- Página da ferramenta `MASSim2Dev`

7.2 Software

fazer os links aparecerem no texto

- `JADE`
- `Repast 3`
- `SAJaS`

7.3 Elementos do grupo

indicar percentagem aproximada de trabalho efetivo de cada elementos do grupo

- Ângela Cardoso - %
- Diogo Ferreira - %
- Marina Camilo - %

Capítulo 8

Apêndice

manual do utilizador (sucinto)

Após a criação de um projeto `JAVA` com o código entregue e adicionadas as bibliotecas de `SAJaS`, `JADE` e `Repast 3`, é necessário correr o programa com a `Main class: main.Simulation`.

Ao correr, vai ser apresentado a consola do `Repast 3`. Começando a simulação será apresentada uma janela onde decorrerá a simulação idêntica a esta:

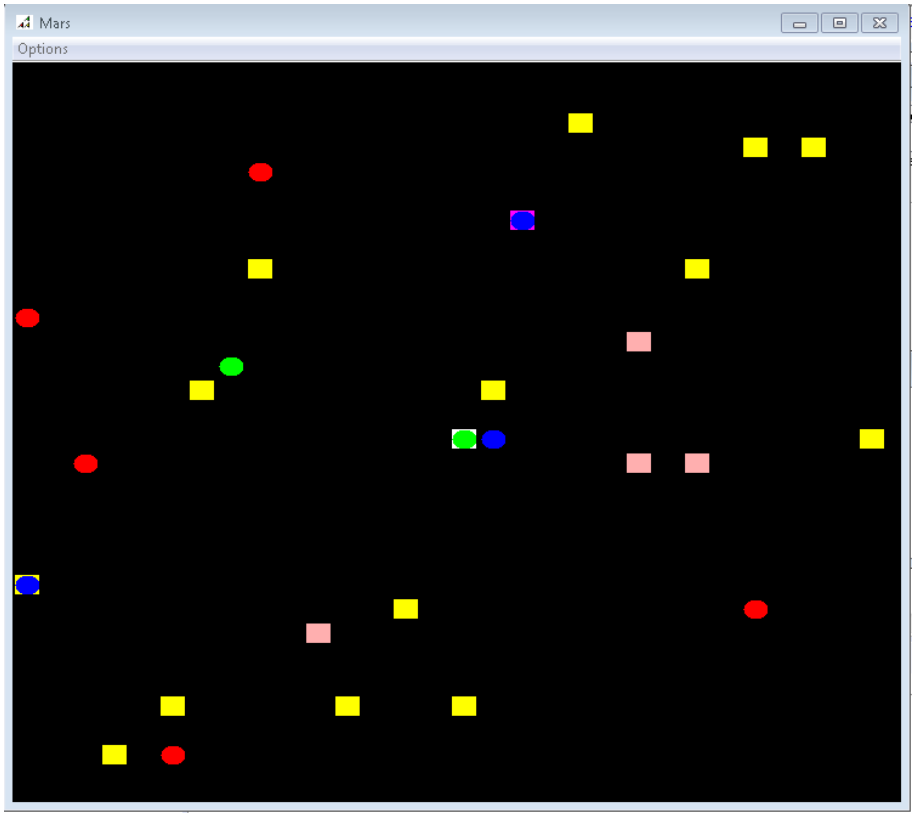

Figura 8.1: Exemplo de uma simulação

A simulação pode correr com diferentes tamanhos, números de minérios, *spotters*, *producers*, *transporters* e a capacidade dos *transporters*. Desta forma, também a percentagem do mineral e o número de fragmentos encontrados num mineral podem ser alterados. Assim, para alterar os parâmetros da simulação descritos acima é necessário aceder a classe `JAVA main.Environment`:

```java
package main;

import java.awt.*;

/**
 * @author diogo
 */
public class Environment {

    public static final int SIZE = 30;
    static final int MINERALS = 20;

    static final int SPOTTERS = 5;
    static final int PRODUCERS = 5;
    static final int TRANSPORTERS = 10;

    public static final int PROB_EXTRACTABLE_MINERAL = 100;
    public static final int MIN_FRAGMENTS_PER_MINERAL = 5;
    public static final int MAX_FRAGMENTS_PER_MINERAL = 15;

    public static final int TRANSPORTER_CAPACITY = 10;
```

Figura 8.2: Valores da Simulação