University of Minho
School of Engineering

Agentes e Sistemas Multiagente

Caroline Rodrigues
PG39285

Hugo da Gião
PG41073

Rafaela de Pinho
PG41095

8 de Dezembro de 2019

Resumo

No âmbito da UC de Agentes Inteligentes, foi-nos pedido a conceção e implementação de um sistema multiagente de monitorização e resolução de catástrofes utilizando o ambiente de desenvolvimento Jade.

Palavras-chave: agentes, inteligência artificial, sistemas multiagentes, diagramas UML.

Conteúdo

Capítulo 1

Introdução

Inteligência artificial (IA) é a inteligência demonstrada pelas máquinas. É frequentemente usada para descrever máquinas que imitam funções "cognitivas" ligadas à mente humana, como "aprendizagem". A maior parte dos livros sobre IA define-a como o estudo de "agentes inteligentes".

O agente inteligente é um sistema autónomo e flexível que age num determinado ambiente, de modo a atingir objetivos. A flexibilidade do agente está relacionada com as suas capacidades de reação, iniciativa, aprendizagem e socialização.

Um sistema multiagente é um sistema composto por interações de múltiplos agentes. Estes sistemas podem resolver problemas complicados de serem resolvidos por um só agente.

O sistema que desenvolvemos consiste num sistema de alocação de recursos para um quartel de bombeiros. Dentro desse universo, criamos alguns agentes com comportamentos específicos que, juntos, atuam em busca dos mesmos objetivos. Hoje em dia, existem diversas soluções no mercado para este problema. Estas utilizam técnicas variadas para resolver este problema, nomeadamente, sistemas de suporte à decisão.

Neste relatório, apresentamos o estado da arte sobre os agentes e a sua aplicação, a arquitetura, os protocolos de comunicação e os diagramas UML. Elucidamos aspetos importantes para a construção do sistema e é possível verificar os agentes criados, as suas funções, os comportamentos e as mensagens trocadas. Descrevemos a estrutura do projeto com os *behaviours*, *performatives*, variáveis, classes e funções utilizados para garantir o bom funcionamento do sistema.

Capítulo 2

Conceção do sistema e estado da arte

2.1 Estado da arte

Este capítulo apresenta pesquisas sobre o tema de inteligência artificial aplicada em simuladores de combate a incêndio.

2.1.1 Diferentes tipos de Sistemas de suporte à decisão na área de prevenção e combate de incêndios

Hoje em dia, existem vários sistemas de suporte à decisão que auxiliam profissionais na área de combate a incêndios.

SSD's baseados em gestão de dados e algoritmos matemáticos e económicos

Este sistema incorpora na sua base de dados um conjunto de cenários pré-processados correspondentes a incêndios específicos. O output deste sistema é um modelo probabilístico que permite às autoridades competentes fazer um uso mais eficiente dos recursos.

SSD's baseados no comportamento dos fogos

Este sistema utiliza o principio de "elliptical growth model of forest fire fronts" para simular o comportamento de fogos.

SSD's baseados no algoritmo de Dijkstra's

De acordo com leis naturais, um fogo vai seguir a rota que demore menos tempo. Este sistema utiliza o algoritmo de Dijkstra para prever a propagação dos fogos.

SSD'S para determinação de risco de incêndio Estes sistemas são constituídos por serviços de localização de zonas com perigo de incêndio.

2.1.2 Soluções existentes

Um Incêndio é uma ocorrência de fogo não controlado que se pode tornar extremamente perigoso para os seres vivos, ou, segundo o dicionário, "o fogo que avança fora de controlo". A exposição a um incêndio pode provocar até a morte, quer devido a queimaduras graves, quer pela inalação dos gases tóxicos libertados.

Atualmente, existem projetos de pesquisa e soluções a serem desenvolvidos no campo da inteligência artificial para combate a incêndios, como veremos a seguir.

2.1.3 Bee2FireDirection

A Bee2FireDirection é uma solução da empresa portuguesa Compta que, contando com o apoio da plataforma Watson da IBM, permite detetar incêndios praticamente em tempo real e prever o nível de risco de ocorrência.

A solução "Bee2FireDetection" opera 24 x 7 como um sistema de vigilância que deteta automaticamente incêndios em fases iniciais e a grandes distâncias, permitindo uma rápida resposta.

Figura 2.1: Site da solução Bee2FireDirection

2.1.4 Faedo - Indra

A Indra é uma empresa global de consultoria e tecnologia que desenvolveu o Faedo, uma solução que é capaz de detetar os menores focos de incêndio a muitos quilómetros de distância. O principal objetivo é fornecer uma ferramenta eficaz que permita tratar os dados cada vez mais rapidamente.

Isso é possível graças ao algoritmo do sistema, capaz de distinguir o fogo de outras fontes de calor, como o motor de um veículo ou um animal, por exemplo.

O Faedo:

-Deteta incêndios por análise de imagem térmica, mesmo quando estes são de pequenas dimensões.

-Integra ferramentas para auxiliar na tomada de decisões para um melhor planeamento das tarefas de extinção.

-Ajuda através de uma análise mais aprofundada da origem do incêndio e das ações tomadas ao extingui-lo.

2.1.5 Simuladores de incêndio como ferramenta auxiliar para o desenvolvimento de projetos de arquitetura e de prevenção de incêndios

O estudo foi realizado por Emerson Luiz Baranoski, mestrando do Programa de Pós-Graduação em Construção Civil da Universidade Federal do Paraná – UFPR, e tem como objetivo lançar algumas considerações a respeito da utilização de simuladores de incêndio como ferramenta auxiliar para o desenvolvimento de projetos de arquitetura e de prevenção de incêndios. Inicialmente, foi efetuado um estudo sobre a teoria da evolução do fogo, comparando com os resultados obtidos em alguns programas de simulação de incênd . Em seguida, relacionaram-se alguns dos principais simuladores de incêndio existentes, indicando as suas principais caraterísticas e, finalmente, foi realizada uma análise sobre a utilização dos simuladores de incêndio no desenvolvimento de projetos.

2.1.6 Modelagem de Sistemas Multiagentes com o RoboCup Rescue: uma Simulação da Propagação de Incêndios

Esta pesquisa apresenta uma análise de simulações feitas no Ambiente RoboCup Rescue para o fenómeno de propagação de incêndios, tendo-se destacado as caraterísticas da propagação de incêndios em centros urbanos e o modo como os agentes cooperantes atuam na sua contenção. Duas simulações foram executadas: uma representando a alta capacidade dos agentes em combater o incêndio, e a outra apresentando a falta de capacidade dos agentes no combate ao incêndio.

Figura 2.2: Site da solução Bee2FireDirection

2.2 Agentes

Neste sistema multiagente, vamos ter três tipos de agentes: agentes participativos (drones, aeronaves e camiões), o agente central (quartel dos bombeiros) e o agente incendiário (cria fogos).

2.2.1 Agentes Participativos

Os agentes participativos são os agentes que vão apagar os fogos. Estes comunicam a sua posição e disponibilidade para apagarem um incêndio à central, e são autónomos, pois são eles que decidem se necessitam de abastecer água e combustível.

1. Os camiões levam, no máximo, 10 unidades de água e 10 unidades de combustível; a sua velocidade é 1x velocidade (valor a definir na construção do sistema, na próxima fase).
2. As aeronaves levam, no máximo, 15 unidades de água e 20 unidades de combustível; a sua velocidade é 2x velocidade.
3. Os drones só conseguem levar, no máximo, 2 unidades de água e 5 unidades de combustível, todavia são mais rápidos; a sua velocidade é 4x velocidade.

Figura 2.3: Animação de um veículo de bombeiros

2.2.2 Agente Central

O agente central recebe o alerta que um fogo começou. Através das coordenadas dos agentes participativos, calcula o melhor percurso para ir apagar o fogo e comunica aos agentes. Caso esse não esteja disponível, volta a calcular.

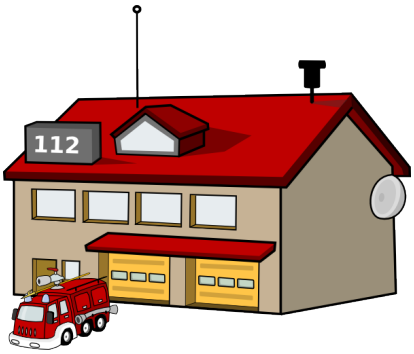


Figura 2.4: Animação de um quartel de bombeiros

2.2.3 Agente Incendiário

O agente incendiário cria fogos em vários pontos, aleatórios, no mapa.

Figura 2.5: Animação de um criminoso

2.2.4 Agente Interface

O agente interface permite a interação entre o sistema e os utilizadores. Tencionamos apresentar ao utilizador algo similar a um mapa de zoneamento com zonas coloridas de acordo com o seu tipo, e com as posições dos veículos assinaladas com um círculo colorido de cor vermelha.

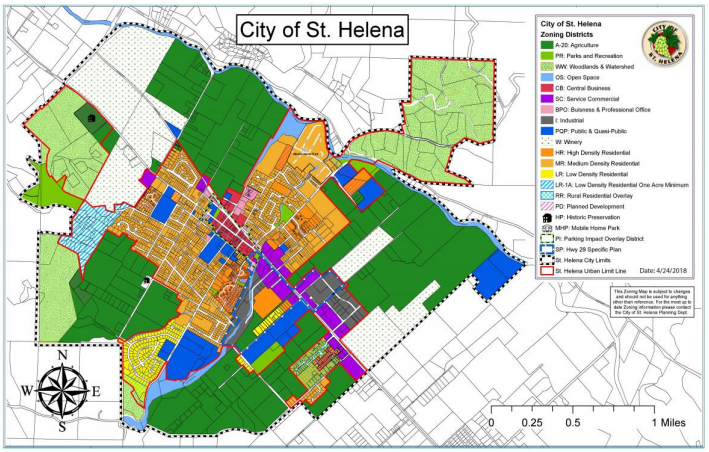


Figura 2.6: Exemplo de mapa de zoneamento

2.3 Protocolo Geral

2.3.1 Estados dos agentes

—

1. **Veículo**

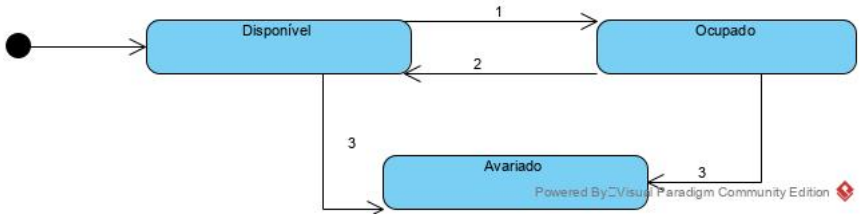


Figura 2.7: Diagrama de estados dos veículos

Um veículo está recetivo para novas tarefas, caso não esteja ocupado a apagar um incêndio.

2. **Outros agentes** Os restantes agentes mantêm-se, geralmente, num estado constante.

2.3.2 Protocolos de comunicação

1. **Comunicação de Incêndios**

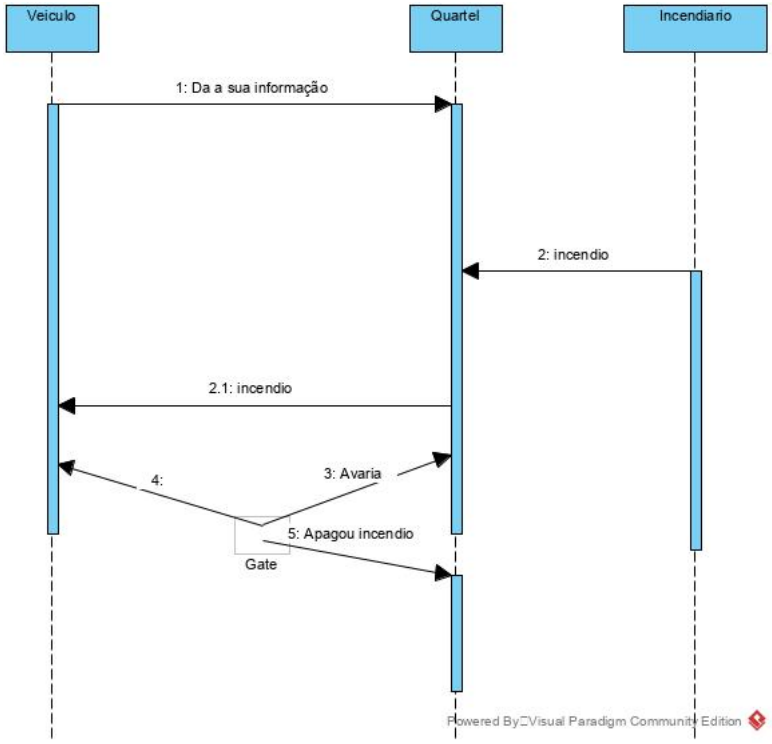


Figura 2.8: Diagrama de estados dos veículos

O veículo comunica ao quartel as suas coordenadas e o incendiário comunica a posição dos novos incêndios. O quartel, após escolher um veículo disponível, comunica-lhe as suas diretivas. Caso este possa realizar o trabalho, comunica ao quartel a sua intenção de o fazer; caso contrário, comunica ao quartel a sua recusa e este escolhe outro veículo para realizar o trabalho.

2.4 Interações entre agentes

2.4.1 Atividades dos Agentes

Nesta secção, serão explicadas diferentes interações entre os agentes no programa e o papel que estas realizam no sistema.

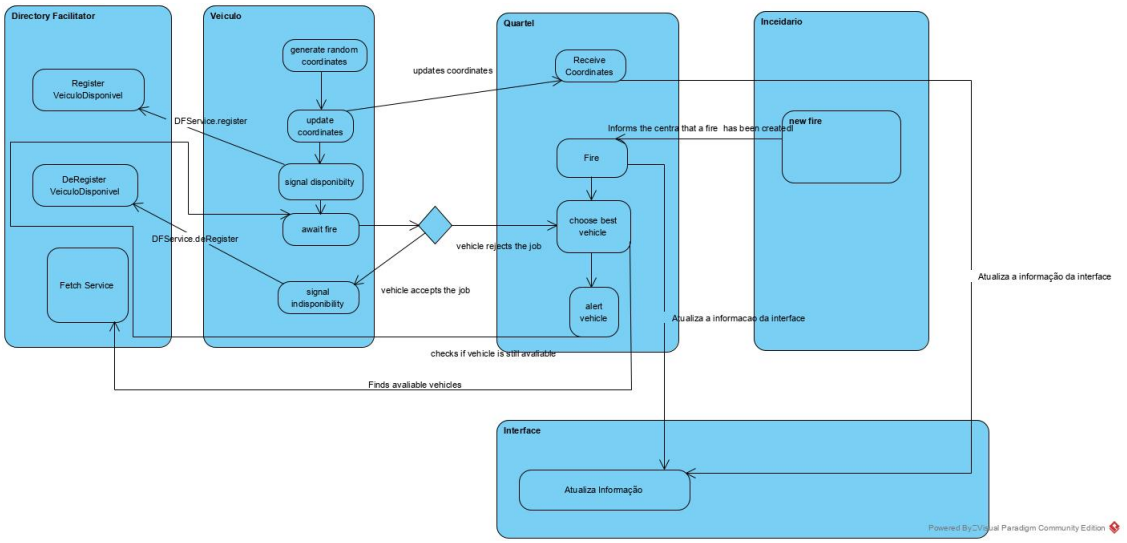


Figura 2.9: Diagrama de Atividade

O veículo comunica com o quartel, enviando periodicamente as suas posições. Quando é iniciado ou o veículo apaga um incêndio, ele regista a sua disponibilidade nas páginas amarelas. Caso o veículo fique imobilizado devido à falta de combustível, o processo termina depois de enviar uma mensagem ao quartel, informando-o da ocorrência e "deregister" o seu serviço das páginas amarelas.

O quartel utiliza as páginas amarelas para obter a lista de veículos disponíveis para apagar o incêndio. Após escolher o veículo para realizar o serviço, verifica se este aceita o serviço. Existem várias razões para que um veículo possa rejeitar um serviço, como, por exemplo, um veículo não ter combustível suficiente para chegar ao local do incêndio ou a uma bomba de combustível. Este acontecimento deverá ser raro, uma vez que o quartel deve possuir informação atualizada dos veículos, mas, caso isso aconteça, têm esta garantia.

O quartel comunica também com o incendiário, que lhe envia informação relativa à localização dos novos fogos. Ele também comunica com o agente interface, para atualizar a informação mostrada ao utilizador e utilizada nas estatísticas.

2.4.2 Diagramas de classes

Quartel

Este agente tem como função principal gerir os movimentos do veículo de modo a garantir a maior eficiência no processo de apagamento e contenção de fogos.

Este agente recebe a informação dos vários veículos e do incendiário, e dá diretivas aos veículos, elucidando sobre o modo de atuação.

O agente quartel guarda a informação relativa ao mapa num "array de arrays" de células, que é um objeto que indica a informação de uma célula e guarda as posições dos incêndios correntes, a informação relativa aos incêndios apagados e as informações dos veículos.

Este agente contém duas *CyclicBehaviours*, uma para a atualização da informação dos veículos e fogos apagados, e outra para receber informação relativa a novos incêndios.

Ele também contém uma *TickBehaviour* utilizada para atualizar a informação na interface.

Veículo

O veículo é um agente que interage com o quartel e que possui como função principal apagar fogos seguindo as diretivas do quartel.

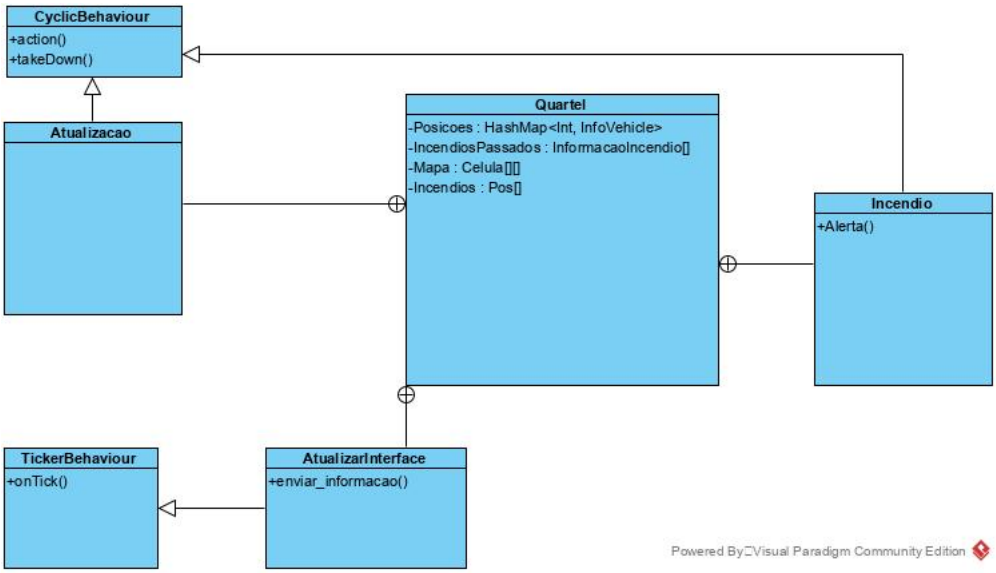


Figura 2.10: Diagrama de classe

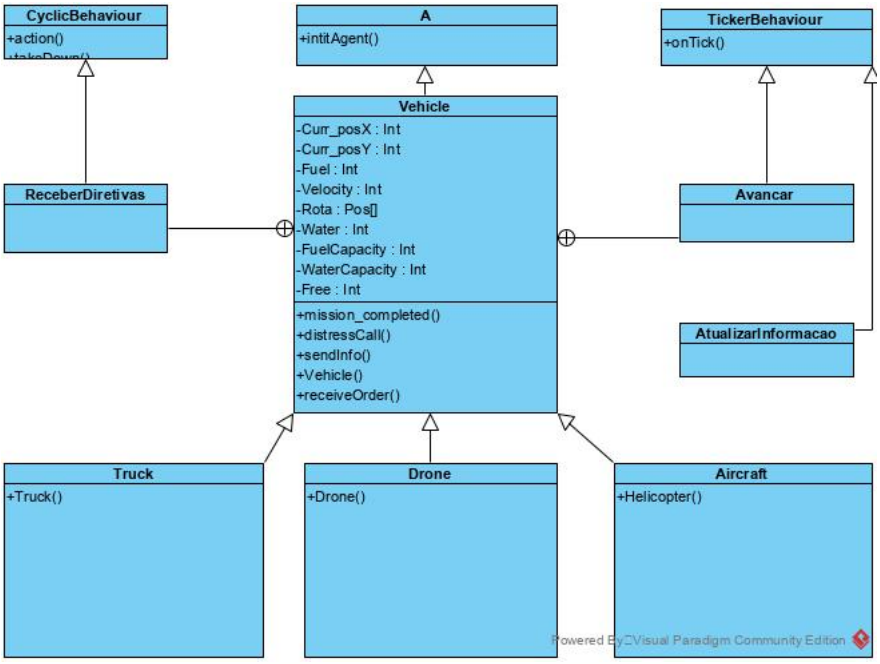


Figura 2.11: Diagrama de classe

Este agente tem uma *Cyclicbehaviour* chamada ReceberDiretivas, que é utilizada para receber diretivas relativas a incêndios a ser apagados pelo veículo. Se o veículo poder apagar o incêndio, então ele calcula o melhor caminho para chegar ao incêndio e poder apagá-lo, e guarda este caminho na variável rota.

Todas as informação relativamente à sua posição corrente no mapa e às quantidades de água e combustível, correntes e máximas, são guardadas. Possui duas *TickerBehaviours*. A 'Atualizar', que envia a informação relativa ao seu combustível, água, velocidade e posição ao quartel, e a 'Avançar', que avança o número de posições no array de rota igual ao valor de velocidade. Caso o array de rota esteja vazio, ele calcula uma nova rota de modo a otimizar o tempo, de acordo com as suas necessidades de água ou combustível.

1. **Aeronave**
   Este agente herda da classe veículo. Tem algumas das suas caraterísticas predefinidas, como

a capacidade de armazenamento de água e combustível, e velocidade. Possivelmente, terá caraterísticas diferentes no que se refere à sua capacidade de movimento, podendo, eventualmente, passar por células em que outras classes de veículos não possam.

2. **Camião**
   Este agente herda da classe Veículo. Tem algumas das suas caraterísticas predefinidas, como a capacidade de armazenamento de água e combustível, e velocidade.

3. **Drone**
   Similar a aeronave, com diferenças nas suas caraterísticas predefinidas.

2.4.3 Incendiário

A função deste agente é criar incêndios.

O incendiário é um agente que, a cada período de tempo determinado, escolhe uma célula do mapa ao caso e informa o quartel que um incêndio foi criado.

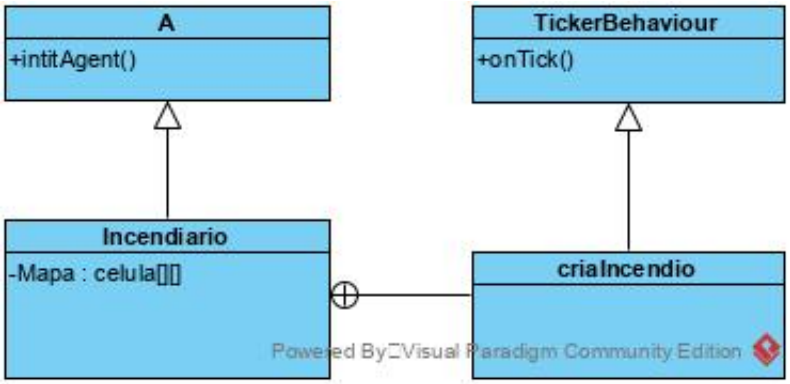


Figura 2.12: Diagrama de classe

Este agente contém informação relativa ao mapa similar à do quartel e do veículo.

Contém uma *TickerBehaviour*, criaIncendio, que escolhe uma célula aleatória e envia ao quartel a informação de que um incêndio foi criado nessa célula.

2.4.4 Interface

Este agente é utilizado para mostrar de uma forma conveniente ao utilizador a execução do sistema. Também é utilizado para apresentar estatísticas sobre o sistema.

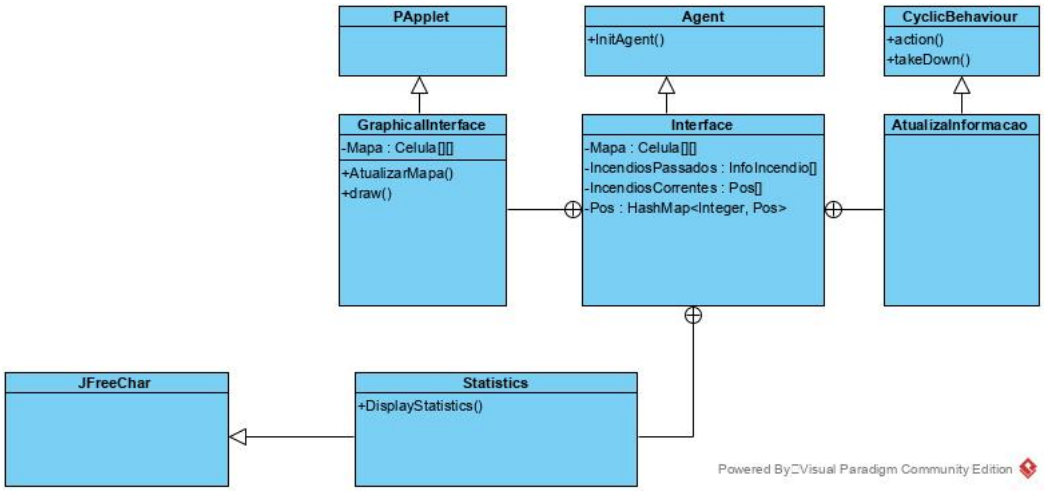


Figura 2.13: Diagrama de classe

Este agente contém um objeto para a interface gráfica do programa. Esperamos fazer esta parte utilizando *processing* ou, por ventura, outra biblioteca gráfica de Java. O mesmo para estatísticas. Pretendemos utilizar JFreeChart ou, eventualmente, outra biblioteca de Java.

Este agente possui uma *CyclicBehaviour* que recebe a informação relativa ao funcionamento do sistema do quartel.

2.5 Processos Internos dos Agentes

2.5.1 Processo de escolha do veículo a realizar a tarefa

Para escolher o veículo que realiza a tarefa, o quartel consulta a lista dos veículos disponíveis nas páginas amarelas, escolhe o mais adequado para a realização da tarefa, contacta esse veículo e, se ele aceitar, acaba este processo, senão escolhe outro veículo.

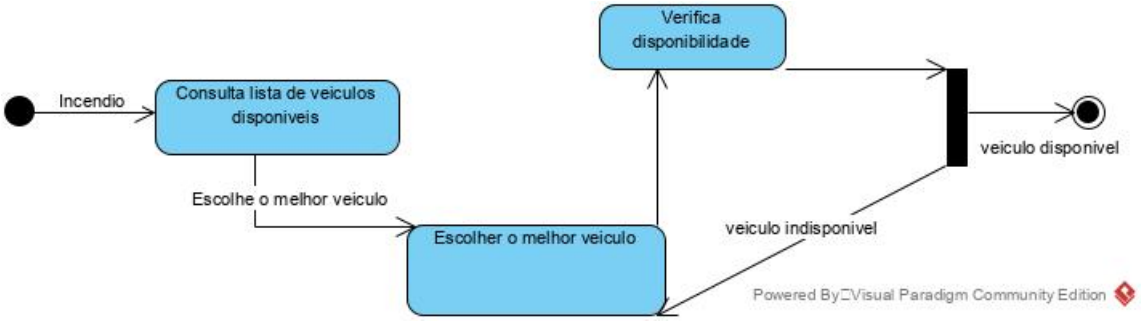


Figura 2.14: Diagrama de classe

2.5.2 Processo de escolha de tarefa do veículo

Existem duas maneiras de um veículo escolher uma nova tarefa. Uma delas é receber diretivas do quartel, outra é, caso não as tenha, tomar iniciativa e abastecer água ou combustível.

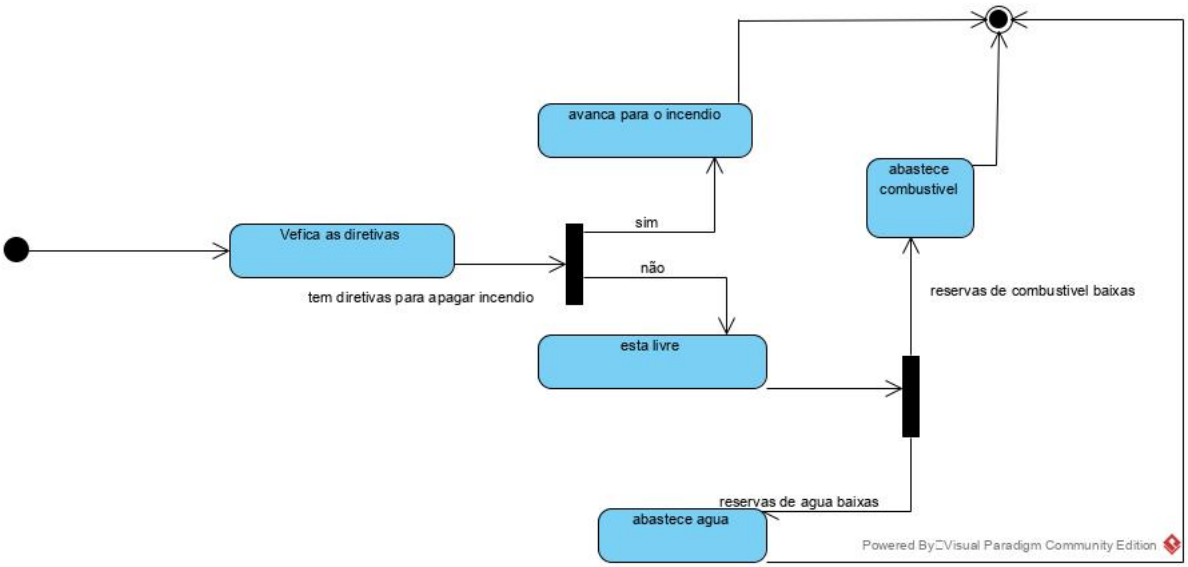


Figura 2.15: Diagrama de classe

2.5.3 Processo de criação de incêndio pelo incendiário

O incendiário escolhe uma célula aleatoriamente e, se esta célula puder ser incendiada, comunica a informação ao quartel.

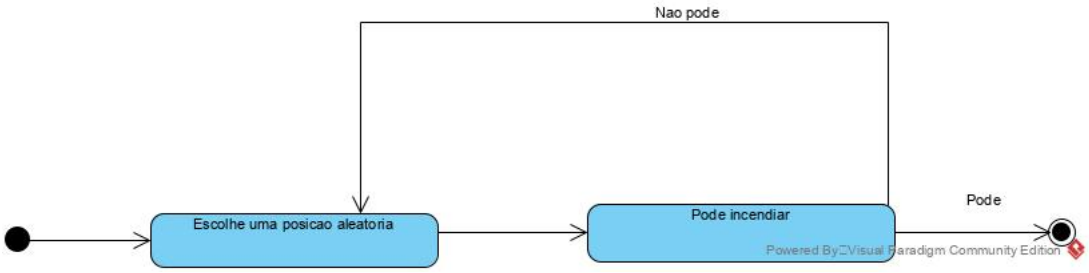


Figura 2.16: Diagrama de classe

Capítulo 3

Mudanças realizadas no esquema do trabalho

Este capítulo relata algumas das mudanças realizadas no esquema do trabalho decorrentes dos problemas e desafios encontrados na sua implementação.

3.0.1 Mudanças a Nível de Comunicação entre os Agentes

A nível de comunicação ocorreram algumas mudanças, nomeadamente: o quartel pode enviar diferentes tipos de mensagens a veículos, utilizadas para as diferentes tarefas a ser realizadas por estes, como reparar outros veículos ou apagar incêndios; o quartel pode também receber mensagens de rejeição diferentes, consoante a tarefa realizada pelo veículo, no entanto no caso da tarefa ser aceite recebe uma mensagem apenas aquando da completude da mesma.

Optou-se por evitar o envio de mensagens de aceitação, sendo a ausência de mensagens indicativa da aceitação do trabalho.

Ao finalizar a tarefa de reparação ou abastecimento de um veículo, o veículo responsável envia uma mensagem ao veículo imobilizado a indicar o serviço fornecido.

São também enviados diferentes tipos de mensagens à interface para actualizar as diferentes componentes da mesma.

Caso fiquem imobilizados, os veículos podem também realizar pedidos de ajuda.

3.1 Mudanças a nível de agentes

3.1.1 Veículos

Ao nível de *behaviours*, o agente manteve-se o mesmo. Porém, ocorreram algumas mudanças em termos de funções utilizadas. Criámos várias funções auxiliares não descritas na fase 1, no entanto, estas são de auxílio ao funcionamento do agente e da sua comunicação com o exterior. Também criamos mais estados do agente veículo do que previamente previsto, motivado pelo facto de os veículos terem mais estados e poderem, assim, exibir comportamentos mais complexos. Guardamos também mais informação neste agente do que foi modelado na parte 1 nomeadamente o engine do jess utilizado para tomar algumas decisões, onde se situa na rota que vai percorrer para executar atarefa corrente, o incˆendio que vai apagar ou o veículo que vai salvar. Guardamos também mais informação neste agente do que o que foi modelado na parte 1, nomeadamente guardamos uma instância do motor jess que o veículo utiliza de modo a tomar decisões que lhe permitam melhor alocar o seu tempo.

3.1.2 Incendiário

O comportamento e *behaviours* deste agente mantêm-se similares ao que foi modelado na fase 1. Este continua apenas com uma *TickerBehaviour* que gera os incêndios para enviar à estação.

3.1.3 Quartel

Em vez de guardarmos a lista de incêndios passados e correntes, optamos por guardar a lista de fogos não apagados e as dos não alocados. Guardamos também uma instância do jess engine e um *HashMap* com as localizações e AID's dos veículos que precisam de auxílio.

Optamos por criar uma *TickerBehaviour* para a alocação de recursos, pois faz mais sentido para podermos privilegiar incêndios com maior gravidade e pelo facto de poderem ocorrer momentos em que nenhum veículo esteja disponível para apagar os fogos.

3.1.4 Interface

Implementamos algumas mudanças no esquema do agente interface na segunda fase. Na sua maioria, elas devem-se ao funcionamento das bibliotecas utilizadas. Por exemplo, as classes Papplet e as restantes funções necessárias para o funcionamento da interface gráfica e para guardar as estatísticas relativas ao funcionamento do sistema estão localizadas na classe MainContainer. O agente interface fica então com a responsabilidade de recolher informação do sistema a partir dos outros agentes. Optamos, também, por guardar a informação do sistema numa classe criada para o efeito. Assim, apenas guardamos um mapa e um objeto de informação do sistema nesta classe. O objeto de informação do sistema é recebido no momento da sua criação e é partilhado com o MainContainer.

Capítulo 4

Implementação do sistema

4.1 Funções e capacidades dos agentes

Esta secção detalha as diferentes capacidades e funções dos agentes no âmbito do sistema desenvolvido, bem o modo como as mesmas foram implementadas.

4.1.1 Interface

Interface Gráfica

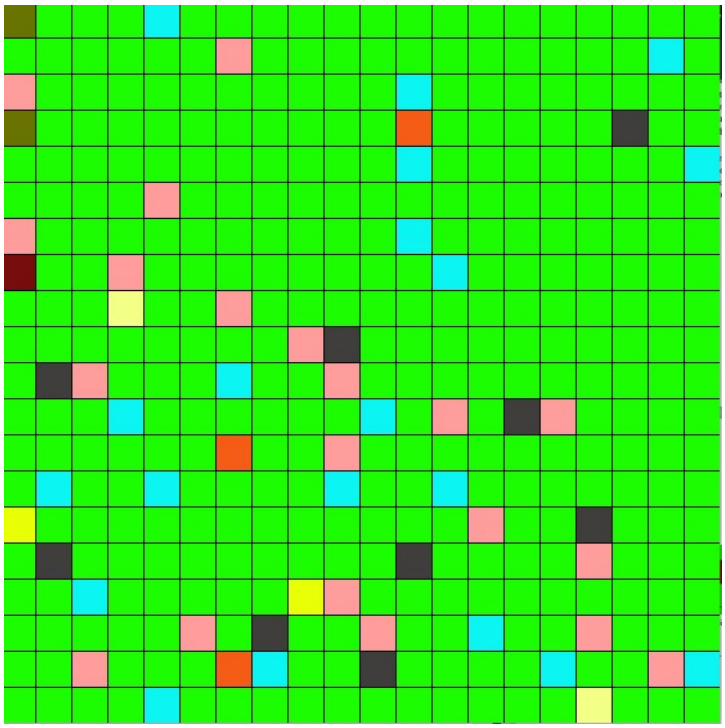


Figura 4.1: Interface Gráfica

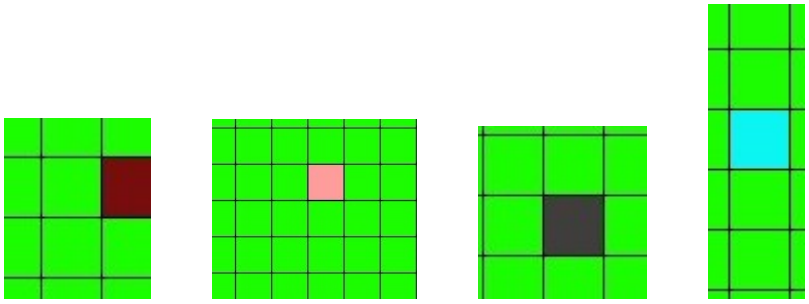
Este trabalho possui uma interface gráfica implementada, utilizando a biblioteca *processing*, que como referido no capítulo 1 é semelhante a um mapa de zoneamento.

O uso desta interface consiste em monitorizar o comportamento dos diferentes agentes no sistema.

Para implementar esta funcionalidade, o agente interface recolhe informação relativa ao estado do sistema a partir das mensagens recebidas do agente quartel e guarda-as num objeto partilhado com o MainContainer. Aquando da invocação da função de atualização da interface gráfica, são

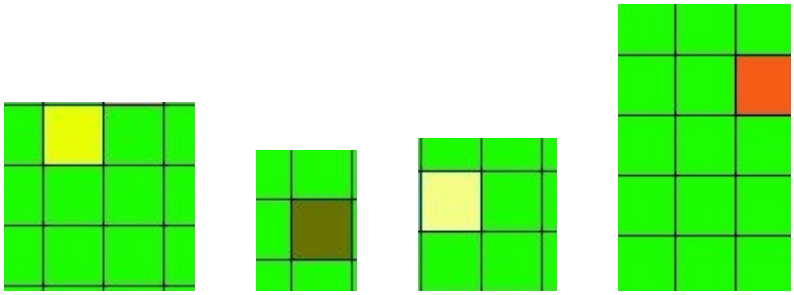
lidos os valores guardados no objeto partilhado e são desenhados no ecrã os incêndios e veículos nas suas respetivas posições.

A interface gráfica consiste num conjunto de retângulos que representam o mapa, sendo que cada quadrícula tem uma cor diferente, dependendo do tipo de área que representa. A estas, são sobrepostas quadrículas de cor laranja, representativas dos incêndios e, posteriormente, quadrículas de cores diferentes, indicativas das posições dos veículos.



(a) fire station (b) residential zone (c) gas station (d) water reservoir

Figura 4.2: Representações de diferentes zonas na interface gráfica



(a) Truck (b) Drone (c) airship (d) water reservoir

Figura 4.3: Representações dos diferentes veículos e dos incêndios na interface gráfica

Estatísticas

Utilizamos a biblioteca JFreeChart para criar gráficos com informação relevante para a analise da performance do sistema.

Esta funcionalidade pode ser usada pelo utilizador fazendo duplo clique na interface gráfica e, posteriormente, um ficheiro com o gráfico relativo às estatísticas do estado do sistema será guardado numa directoria predefinida.

O gráfico criado consiste num BarPlot com os diferentes indicadores estatísticos.

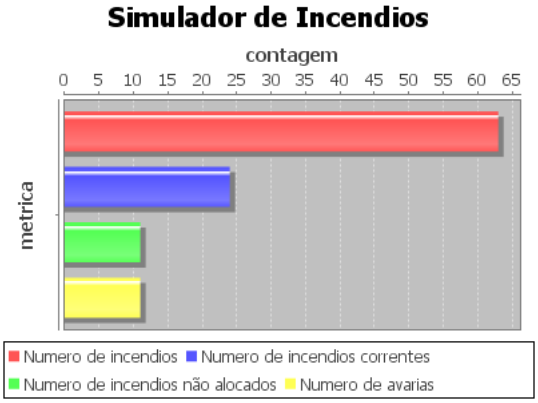


Figura 4.4: Ficheiro com as estatísticas do sistema

4.1.2 Veículos

Como referido na primeira fase, existem diferentes veículos. Estes são utilizados para criar objetos do tipo 'veículo' com características diferentes.

Receber e executar pedidos para apagar incêndios

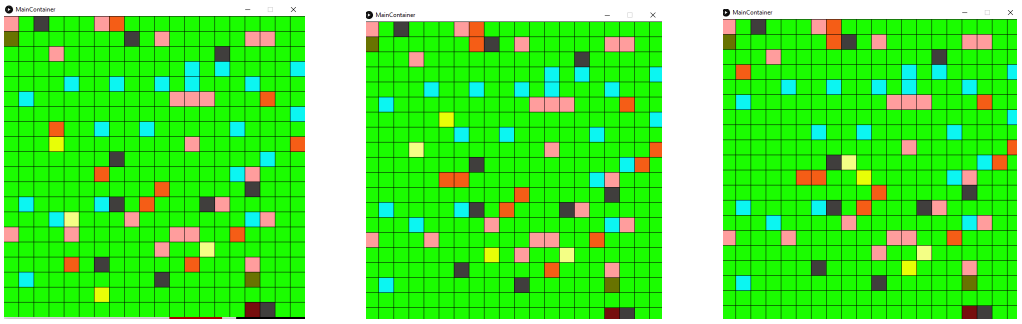
Provavelmente, a funcionalidade mais importante deste agente apresenta-se como a sua capacidade de receber pedidos do agente quartel para apagar incêndios. O agente, quando se encontra em estado livre, recebe pedidos da estação relativos a novos fogos para os quais este foi escolhido para executar a tarefa de os extinguir.

O veículo recebe da estação uma mensagem com a performative ACLMessage.PROPOSE e ontologia "incêndio".

Ao receber um pedido, o agente veículo verifica se lhe é possível apagar o incêndio. Caso seja possível, o veículo entra no estado incêndio e guarda o caminho até à posição do incêndio. O caminho é calculado utilizando uma função criada para o propósito, que utiliza o algoritmo de djikstra para encontrar um caminho, direto ou não, que o veículo possa tomar para chegar ao local do incêndio. No caso de o veículo não poder apagar o incêndio, uma mensagem de rejeição é enviada ao quartel e verifica-se se é possível chegar a uma estação de gasolina. Se o veículo conseguir, este entra em estado de emergência, retira-se do diretório de busca e vai abastecer. Se não conseguir abastecer, retira-se do diretório de busca e manda uma mensagem de socorro à estação.

Quando está em modo incêndio e é acionada a sua *behaviour* de movimento, o veículo, se não tiver chegado ao local do incêndio, avança uma posição para a frente. De acordo com o tipo da célula correspondente a esta posição, abastece ou enche o reservatório de água e perde 1 unidade de combustível.

Quando chega ao local do incêndio, o veículo perde 1 unidade de água, avisa a estação do seu sucesso na tarefa executada e verifica se tem água. Se tiver, regista-se no diretório de busca, senão, tenta encontrar um caminho para encher o reservatório de água. Se conseguir, entra no estado de emergência e enche o reservatório de água, senão, manda uma mensagem de socorro à estação.



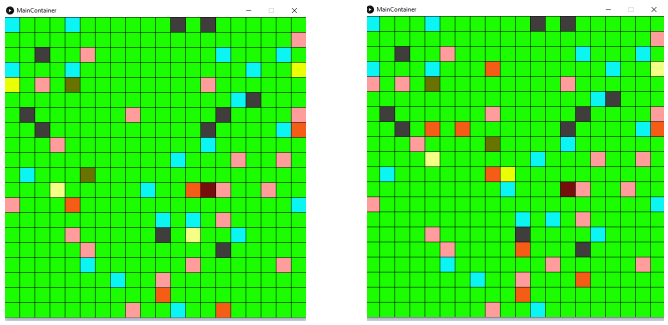
(a) apagar incêndios fase inicial (b) apagar incêndios fase posterior (c) incêndio apagado

Figura 4.5: Representações de um incêndio a ser apagado

Receber e executar pedidos para socorrer veículos

Os veículos pedem indicações para auxiliar outros veículos que estejam imobilizados. Estas mensagens são enviadas com a performative ACLMessage.PROPOSE e ontologia "veículo". O veículo deverá estar em estado livre.

Os processos de decisão e de execução da tarefa são similares aos de apagar incêndios. As únicas diferenças ocorrem na conclusão das tarefas, em que não é necessário verificar as reservas de água, é enviada uma mensagem de informação ao veículo imobilizado em vez da estação, e difere a informação guardada para executar a tarefa.



(a) veiculo a ser socorrido (b) veiculo socorrido

Figura 4.6: Representações de um veiculo a ser socorrido

Aproveitar melhor o tempo livre

Uma das capacidades deste agente é a de, caso esteja em estado livre e consoante os recursos disponíveis, repor estes.

Para tal, utilizamos a biblioteca jess para o processo de tomada de decisão.

A primeira parte deste processo consiste em declarar uma lista representatitava dos recursos do agente num motor jess. Posteriormente, acionamos o motor jess e este toma uma decisão de acordo com os factos inseridos e com as regras previamente definidas. A partir destes, ele cria um novo facto "allocation". Após correr o motor jess, recolhemos o facto "allocation"criado e agimos de acordo com o mesmo.

Pedir auxílio à estação

Como já foi referido anteriormente, os veículos possuem a capacidade de pedir auxílio à estação quando necessário. Isto acontece quando os mesmos estão imobilizados por falta de recursos.

Quando isto acontece, os veículos enviam uma mensagem com performative ACLMessage.REQUEST à estação com a sua localização corrente e mantêm-se parados até ser-lhes prestado auxílio.

Notar e resolver emergências

Os veículos podem entrar em estado de emergência quando acabarem de apagar um incêndio, caso tenham as suas reservas de água vazias ou caso não consigam chegar ao local do incêndio depois de receber uma diretiva para tal, mesmo que consigam abastecer.

4.1.3 Quartel

Alocar Incêndios

O agente quartel possui um *TickerBehaviour* para alocar recursos. Quando esta *behaviour* é acionada, os incêndios e pedidos de auxílio são alocados pelos veículos disponíveis. O processo de alocação de incêndios consiste em, primeiramente, recolher a lista dos veículos disponíveis, a partir do DFService. Através desta, ordenamos a lista dos incêndios não alocados, de acordo com a sua prioridade e, depois, por ordem de receção. Para cada incêndio, colocamos a lista veículos e o seu *score* na dos incêndios. O *score* é calculado de acordo com os fatores velocidade, distância ao incêndio, combustível e capacidade de combustível que são inseridos como factos no motor jess.

Estando os factos inseridos no motor jess, corremo-lo e obtemos o veículo disponível. Este passa a ser o veículo escolhido. Assim, removemos este veículo da lista dos disponíveis e enviamos-lhe uma mensagem com performative ACLMessage.PROPOSE e ontologia "incêndio".

Este agente possui também uma*CyclicBehaviour* para receber mensagens. Alguns usos desta *behaviour* passam pela receção de mensagens relativas à rejeição de trabalho propostos, um deles incêndios. Ao receber uma mensagem com Performative de ACLMessage.REFUSE e ontologia "incêndio", o quartel adiciona o incêndio rejeitado à lista dos incêndios não alocados.

A *Cyclicbehaviour* descrita anteriormente é também utilizada para receber novos incêndios do agente incendiário. Quando é recebida uma mensagem com performative ACLMessage.PROPOSE, o agente guarda o incêndio na lista de incêndios não alocados.

Alocar pedidos de ajuda

O processo de alocação de pedidos de ajuda é bastante similar aos de incêndios. Realça-se que a alocação de pedidos de ajuda é prioritária em relação à alocação de fogos, ou seja, a cada instância da *TickerBehaviour* de alocação de recursos, são primeiramente alocados os pedidos de ajuda e, só depois, os incêndios. Dito isso, o processo de alocação de pedidos é exatamente igual ao dos incêndios, sendo apenas utilizados objetos e ontologias diferentes. O mesmo acontece no processo de gestão de rejeições.

Por outro lado, os processos de receção de pedidos de auxílio e de confirmação de finalização dos mesmos são diferentes. O processo de receção de pedidos é realizado a partir de mensagens com performative ACLMessage.REQUEST e o processo de confirmação da realização do pedido é inexistente, pois as mensagens são enviadas para o veículo afetado.

Receber posições dos veículos

Quando o agente recebe uma mensagem com performetive ACLMessage.INFORM, este coloca a posição e o AID do veículo no HashMap de localizações.

4.1.4 Incendiário

Criar incêndios

A única função deste agente é criar incêndios para realizar esta tarefa. Este escolhe duas coordenadas aleatoriamente, verifica se o tipo da célula lhe permite criar um fogo nesta e, se lhe for possível criar um incêndio, envia uma mensagem com performative ACLMessage.PROPOSE com o objeto de incêndio ao quartel.

4.2 Jess

Para este projeto, criamos dois ficheiros jess para usos distintos.

O primeiro é utilizado pelo agente estação para a alocação de recursos entre os veículos disponíveis.

```
(deftemplate vehicle
    (slot score)
    (slot name)
    )

 (deftemplate task
        (slot name)
 )

 (deftemplate allocation
        (slot taskname)
        (slot vehiclename)
        )

(defrule allocate_vehicles
        ?f <- (task (name ?n))
        ?v <- (vehicle (name ?n1) (score ?s1))
        (not (vehicle (score ?s2&:(< ?s2 ?s1))))
        =>
        (assert (allocation (taskname ?n) (vehiclename ?n1)))
        )
```

Neste ficheiro, criamos 3 *templates* de factos.O primeiro para veículos e o seu *score*, o segundo para as tarefas , e o terceiro para alocação dos recursos.

Criamos também uma regra de modo a estabelecer que, para cada tarefa recebida, esta encontra um veículo para o qual não existe outro com o *score* mais baixo e cria um facto do tipo *allocation* com o seu nome e o da tarefa.

O segundo é utilizado pelo agente veículo para obter um melhor proveito do tempo livre.

```
deftemplate vehicle
    (slot water)
    (slot fuel)
    (slot maxFuel)
    (slot maxWater)
    )

 (deftemplate decision
        (slot value)
 )



(defrule allocate_time
        ?v <- (vehicle (fuel ?f) (water ?w) (maxFuel ?f1) (maxWater ?w1) )
        =>
        (if (and (<= ?w  2) (> ?w1 2 )) then  (assert (decision (value 0) ))
        else (if (<= ?f (- ?f1 1 ) ) then (assert (decision (value 1))) else
        (if (< ?w ?w1) then (assert (decision (value 0))) else
        (assert (decision (value 3))))))
        )
```

Neste ficheiro, criamos 2 *templates*, um para o veículo e outro para a decisão.

Criamos uma regra de modo a estabelecer que, por cada veículo inserido, e de acordo com os seus recursos, se cria uma regra de chamada de decisão com um valor inteiro correspondente a uma ação no sistema. Neste caso, 0 corresponde a procurar água, 1 a procurar combustível e 3 a ficar parado

4.3 Estrutura do projeto

Esta secção detalha a função de algumas classes, variáveis e funções do sistema concebido.

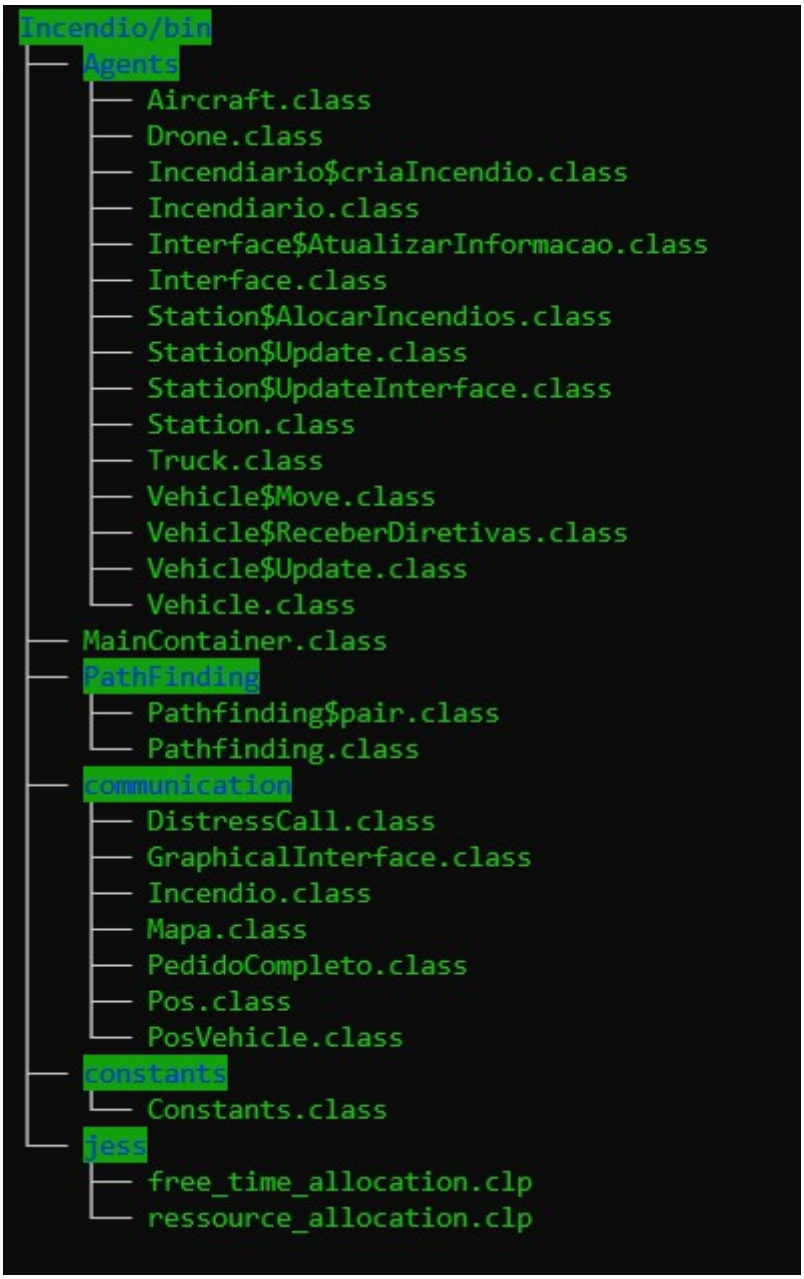


Figura 4.7: Estrutura do programa

4.3.1 Station.java

Esta classes corresponde ao agente Quartel,a sua funcionalidade consiste em gerir os diferentes recursos e tarefas existentes no sistema tal como enviar a informação pertinente a interface gráfica.

Variáveis

- Station manager - Guarda o objecto do agente para uso em classes contidas no mesmo.
- HashMap <AID,PosVehicle> localizações - Utilizado para guardar a informação relativa às posições dos veículos.
- HashMap<AID,PosVehicle> DistressCalls - Utilizado para armazenar os AID's e posições dos veículos com pedidos de ajuda por alocar.
- ArrayList< *Incendio* > incendios - Utilizado para guardar a lista dos incêndios existentes no momento presente.
- ArrayList< *Incendio* > NAincendios - Guarda a lista dos incêndios não alocados.
- Rete engine - Guarda uma instância do engine Jess utilizado para tomar decisões relativas a alocação de recursos para as diferentes tarefas.
- Mapa mapa - Mapa do sistema.
- int nr_incendios - Guarda o número de incêndios que ocorreram no sistema.

Funções

- setup() - Função que inicia o agente.

Classes

Alocar Recursos

Funções

- AlocarRecursos(Agent a, long period) - Cria a behaviour
- int getScore(PosVehicle v,int x,int y) - Calcula o score que um determinado veículo tem para realizar uma tarefa.
- void addVehicleFact(String name,PosVehicle v,int x,int y,Rete engine) - Adiciona um facto do tipo veículo
- addTaskFact(String name,Rete engine) - Adiciona um facto do tipo task.
- AID allocate_ressource(int x,int y,ArrayList<DFAgentDescription> results) - Dada a lista dos agentes disponíveis devolve o agente mais adequado para realizar uma tarefa.
- void allocateFire(ArrayList<incendio> New_NAincendios, int i, ArrayList<DFAgentDescription> results) - Devolve o melhor veículo para apagar um incêndio esta função utiliza a função allocate_ressource.
- allocateDistressCall(HashMap<AID,PosVehicle > New_DistressCalls ,AID aid, ArrayList < DFAgentDescription > results) - Devolve o melhor veículo para auxiliar outro esta função utiliza a função allocate_ressource.
- onTick()

Update

Funções

- Update()
- action()
- void new_fire(ACLMessage msg) - Guarda a informação relativa ao incêndio contido na mensagem dentro das variáveis globais desta classe.
- atualizarCordenadas(ACLMessage msg) - Dada uma mensagem com as coordenadas no veículo esta função guarda a informação no estado global.

UpdateInterface

Funções

- UpdateInterface(Agent agent,int time) - Envia a informação contida no sistema pertinente ao funcionamento da interface gráfica.
- AtualizarInterface()
- onTick()

4.3.2 Vehicle.java

Variáveis

- int Work_progress - Posição do array destination onde o veículo se encontra.
- int Curr_posX - Posição X corrente no mapa.
- int Curr_posY - Posição Y corrente no mapa.
- public int Fuel - Combustível do veículo.
- public int FuelCapacity - Capacidade de combustível do veiuclo.
- public int Water - Reservas de água do veículo.
- public int WaterCapacity - Capacidade máxima de água do veículo.
- int Velocity - Velocidade do veículo.
- int State - Estado do veículo.
- Pos destination[] - Destino correntemente a ser atingido.
- Mapa mapa - Mapa do sistema.
- Incendio incendio_corrente - Incêndio correntemente a ser apagado.
- PosVehicle veiculo_avariado - Posição do veículo avariado a ser auxiliado pelo agente.
- AID aid_veiculo_avariado - AID do veículo avariado a ser auxiliado pelo agente.
- Rete engine - Instância de jess utilizada para tomar decisões.

Funções

- setup()
- AtualizaInformacao() - Envia o seu posicionamento à Estação.
- sendDistressCall() - Envia um pedido de ajuda à Estação.

Classes

Update

Funções

- Update(Agent a, long period)
- void onTick()

Move

Funções

- Move(this,time) - Corresponde a uma acção de movimento do veículo.
- PedidoCompleto() - Sinaliza a estação que um pedido foi completo.
- addFact(int Fuel,int FuelMax,int Water,int WaterMax,Rete engine) - Adiciona um facto ao motor jess.
- AllocateStateTime() - Função invoca o motor jess para escolher uma tarefa a ser realizada em tempo livre.
- void avanca()
- onTick()

4.3.3 Incendiario.java

variáveis

- Mapa mapa

Classes

CriaIncendio

Funções

- onTick()

4.3.4 Interface.java

Variáveis

- Interface manager
- Mapa mapa - mapa do sistema
- GraphicalInterface graphicalinterface

Classes

AtualizarInformacao

Funções

- action()

4.3.5 MainContainer.java

Variáveis

- Runtime rt;
- ContainerController container
- static GraphicalInterface GI
- public int nr_statistics

Funções

- populateCells(Mapa mapa,int type,int x,int y,int nr_cells) Cria células do tipo definido no mapa.
- Mapa createNewMap(int x,int y)- Cria um novo objecto do tipo Mapa. Cria um novo mapa deste tamanho com as características predefinidas no sistema.
- drawGraph() - Cria um ficheiro na directoria predefinida com um gráfico representante do estado do sistema.

4.3.6 Pathfinding.java

4.3.7 Funções

- ArrayList<pair> neighbours(int size_x,int size_y,int x,int y) - Devolve a lista dos vizinho duma célula.
- Pos[] djikstra(Mapa mapa,int orig_x,int orig_y,int dest_x,int dest_y) - Devolve o caminho mais curto entre dois pontos.
- Pos[] concatArrays(Pos[] arr1,Pos[] arr2)
- Pos[] find_path_aux(Mapa mapa,int FuelCapacity,int Fuel, int Curr_posX,int Curr_posY,int incendioX,int incendioY,ArrayList¡Pos¿ gas_stations)
- Pos[] find_path(Mapa mapa,int FuelCapacity,int Fuel, int incendioX,int incendioY,int Curr_posX,int Curr_posY)
- Pos[] path_nearest_gas_station(Mapa mapa,int Curr_posX,int Curr_posY,int Fuel) - Encontra um caminho entre a posição de um veículo e um incêndio.
- Pos[] path_nearest_water_reservoir(Mapa mapa,int Curr_posX,int Curr_posY,int Fuel,int FuelCapacity) - Encontra o caminho para a reserva de água mais próxima.
- int distancia(int pointaX,int pointaY,int pointbX,int pointbY) - Calcula a distância entre dois pontos.

4.3.8 Constants.java

Variáveis

- public static final int TruckFuelCapacity - Capacidade de combustível do agente camião.
- public static final int TruckWaterCapacity - Capacidade de água do agente camião.
- public static final int TruckVelocity - Velocidade ao agente camião.
- public static final int TruckFuelCapacity - Capacidade de combustível do agente camião.
- public static final int TruckWaterCapacity- Capacidade de água do agente camião.
- public static final int TruckVelocity - Velocidade do agente camião.

- public static final int DroneFuelCapacity - Capacidade de combustivel do agente camião.
- public static final int DroneWaterCapacity - Capacidade de água do agente camião.
- public static final int DroneVelocity - Velocidade ao agente camião.
- public static final int AircraftFuelCapacity - Capacidade de combustível do agente camião.
- public static final int AircraftWaterCapacity - Capacidade de água do agente camião.
- public static final int AircraftVelocity - Velocidade ao agente camião.
- public static final int RuralZone
- public static final int ResidentialZone
- public static final int GasStation
- public static final int FireStation
- public static final int WaterReservoir
- public static final int Nr_Trucks
- public static final int Nr_Drones
- public static final int Nr_Aircrafts
- public static final int Nr_GasStationCells
- public static final int Nr_ResidentialCells
- public static final int Nr_FireStationCells
- public static final int Nr_WaterReservoirs
- public static final int SizeX - Tamanho x do mapa.
- public static final int SizeY - Tamanho y do mapa.
- public static final int StateFree
- public static final int StateFire
- public static final int StateBroken
- public static final int StateRepair
- public static final int StateEmergency
- public static final int distanceWeight - Peso da distancia ao local no processo de alocação de recursos.
- public static final int fuelWeight - Peso do combustível no processo de alocação de recursos.
- public static final int fuelCapacityWeight - Peso da capacidade de combustível no processo de alocação de recursos.
- public static final int speedWeight - Peso da velocidade na decisão de alocação de recursos.
- public static String statistics_path - Path onde são guardadas as estatísticas.

4.4 Bibliotecas

4.4.1 Processing

`https://processing.org/`

4.4.2 Jess

`https://jessrules.com/jess/docs/71/library.html`

4.4.3 JfreeChart

`http://www.jfree.org/jfreechart`

Por motivos de conveniência utilizamos uma versão antiga desta biblioteca(jcommon-1.0.8,jfreechart-1.0.13).Estes ficheiros estão disponibilizados juntamente com o nosso projeto.

Capítulo 5

Conclusão

Na primeira fase do trabalho, elaboramos parte do relatório que contém o estado da arte sobre os agentes e a sua aplicação em domínios concretos. Idealizamos também a estrutura do sistema, juntamente com os diagramas UML. Dessa forma, modelamos uma arquitetura distribuída, baseada em agentes para combate a incêndios. No âmbito de planeamento, tivemos alguns problemas com a ferramenta de criação dos diagramas, mas conseguimos ultrapassá-los.

Na segunda fase, desenvolvemos o sistema multiagente de combate a incêndios. Implementamos o código e construímos os agentes no ambiente de desenvolvimento JADE e JESS, mas, para isso, tivemos que realizar alguns ajustes e alterações que não estavam contempladas na primeira fase. Para apresentação dos resultados estatísticos, utilizamos uma API gráfica JFreeChart.

Com a realização deste trabalho, conseguimos abordar vários temas leccionados nas aulas, como, por exemplo, agentes, sistemas multiagente e diagramas UML. Com a utilização de *behaviours*, *performatives* e a FIPA-ACL garantimos o bom funcionamento do sistema com agentes que enviam e recebem mensagens, trocam informações e atuam em conjunto, com o fim de alcançar os mesmos objetivos definidos previamente, de que são em concreto neste sistema apagar fogos,auxiliar veículos imobilizados e outras tarefas relacionadas com a extinção de incêndios.

Este trabalho permitiu-nos também verificar algumas das dificuldades em implementar um sistema, de computação em agentes, sofisticado e complexo, nomeadamente enfrentamos algumas dificuldades em termos do mecanismo de mensagens, sincronização das tarefas dos agentes e diagnóstico de erros no funcionamento do sistema.

Com este trabalho, também aprendemos um pouco mais sobre as soluções que já existem em torno deste tema.

Capítulo 6

Referencias

https://hal.inria.fr/hal-01431006/document
https://link.springer.com/article/10.1007/s11676-017-0452-1
https://encurtador.com.br/xCER8
http://www.bee2firedetection.com/
https://www.indracompany.com/en/faedo-protect-forests-forest-fires
https://encurtador.com.br/bcfgm