Universidade do Minho

Agentes Inteligentes

Trabalho Prático
Agentes e Sistemas Multiagente

Entrega Final

Mestrado de Engenharia Informática

Grupo 1

Ângela Barros - PG38407

Rui Fontes - PG39296

Braga, Portugal

8 de Dezembro de 2019

Conteúdo

Lista de Figuras

Listings

1 Contextualização

O cenário escolhido para este projeto é um simulador de combate a incêndios florestais com o uso de veículo autónomos e inteligentes (voadores e terrestres), através do uso de aeronaves, drones e camiões de bombeiros (agentes participativos). Estes agentes serão monitorizados por um quartel de bombeiros (agente central), responsável pela coordenação e distribuição de tarefas entre os agentes participativos.

A tomada de decisão por parte do quartel de bombeiros será influenciada por um conjunto de fatores existentes no meio, tais como: localizações de locais de abastecimento disponíveis de água (rios, mares, lagoas, entre outros) e combustível. Em contrapartida, como forma de gerar os incêndios em momentos/posições aleatórias, deverá ser criado um agente secundário (ateador de fogos) responsável por esta tarefa.

1.1 Estruturação do Documento

Relativamente à estrutura deste documento, no capítulo 1, é realizada uma pequena introdução do projeto. Por último, é abordada a estrutura do mesmo documento.

No capítulo 2 será apresentado o estudo do Estado da Arte. No capítulo 3 serão apresentados os diagramas UML desenvolvidos na fase de conceptualização do problema. A arquitetura encontra-se no capítulo 3.1 juntamente com o *Workflow* do projeto. No capítulo 7 é feita uma conclusão do trabalho desenvolvido até à data, e é ainda fornecida informação sobre trabalho futuro no capítulo 7.1.

2 Estado da Arte

Um Sistema Multiagente (SMA) é um sistema computacional em que dois ou mais agentes interagem ou trabalham em conjunto de forma a desempenhar determinadas tarefas ou satisfazer um conjunto de objetivos.

Um dos pontos essenciais para permitir a construção de sociedades de agentes, consiste em conseguir gerir as interações e as dependências das atividades dos diferentes agentes no contexto do Sistema Multiagente, ou seja, coordenar esses agentes. Desta forma, a coordenação desempenha um papel essencial nos SMA porque estes sistemas são inerentemente distribuídos.

Diversas metodologias de coordenação foram propostas por diferentes autores, dividindo-se em dois grupos principais:

- metodologias aplicáveis em domínios contendo **agentes competitivos** (*"self-interested"*), ou seja, agentes preocupados com o seu bem próprio;
- metodologias aplicáveis a domínios contendo **agentes cooperativos**, ou seja, agentes que incluem uma noção de preocupação pelo bem do conjunto.

Os Sistemas Multiagente incluem diversos agentes que interagem ou trabalham em conjunto, podendo estes serem homogéneos ou heterogéneos. Cada agente é basicamente um elemento capaz de resolução autónoma de problemas e opera assincronamente, com respeito aos outros agentes.

2.1 Distribuição de Tarefas

A alocação de tarefas pode ser realizada de duas formas:

1. **Alocação Centralizada de Tarefas** - um planeador central é utilizado para alocar as tarefas para os restantes agentes.
2. **Alocação Distribuída de Tarefas** - A tarefa pode ser recebida por todos os agentes e estes comunicam entre si para completar a tarefa.

2.2 Ferramentas de desenvolvimento

- **Aglet** - Sistema para desenvolvimento de agentes móveis desenvolvido pela IBM. Um *aglet* é um agente Java que pode mover-se autonomamente e espontaneamente de um host para outro, carregando consigo um pedaço de código.
- **JADE** - O Java Agent Development Framework, mais conhecido pelo acrónimo JADE, é um framework de software para o desenvolvimento de agentes inteligentes, implementado em Java.
- **Jack** - É uma framework em JAVA para o desenvolvimento de sistemas multiagente. JACK foi desenvolvido por *Agent Oriented Software Pty.* (AOS)

2.3 Exemplos de Sistemas Multiagentes

Existem diversos exemplos reais de Sistemas Multiagentes cujo foco é a coordenação de distribuição de tarefas. A título de exemplo, apresenta-se, de seguida, dois exemplos:

- **Sistema multiagente para coordenação de ambulâncias para serviços médicos de emergência** - Para coordenar ambulâncias em serviços de urgência, um sistema multiagente usa um mecanismo de leilão baseado em confiança. Resultados de testes utilizando dados reais mostram que este sistema pode eficazmente atribuir ambulâncias a pacientes e, consequentemente, reduzir o tempo de transporte. [1]

- ***RoboCup Rescue League* - Competição de simulação de agentes** - A competição de simulação de agentes envolve maioritariamente a avaliação da performance de equipas de agentes em diferentes mapas na plataforma do RoboCup Rescue Agent Simulation (RCRS), um ambiente de simulação para pesquisar/analisar a gestão de respostas em casos de catástrofe. [2]

2.4 Protocolos de Comunicação

2.4.1 Arquiteturas do Sub-Sistema de Comunicação

- **Comunicação Direta** - Os agentes tratam da sua própria comunicação sem intervenção de qualquer outro agente. Para tal, partilham especificações, enviando aos outros agentes as suas capacidades e/ou necessidades de forma a cada agente poder tomar, individualmente, as suas decisões relativas à comunicação.

  Neste tipo de arquitetura cada agente comunica diretamente com qualquer outro agente, sem qualquer intermediário. Um dos principais problemas que se coloca nesta arquitetura está relacionado com a inexistência de um elemento coordenador da comunicação, o que pode originar o bloqueio do sistema se, por exemplo, todos os agentes decidirem enviar mensagens ao mesmo tempo.

- **Comunicação Assistida** - Os agentes apoiam-se em agentes especiais designados "agentes facilitadores", de forma a efetuarem a comunicação com os outros agentes. Nesta arquitectura, a organização de agentes é do tipo sistema federado. Nestes casos, se um dado agente i desejar enviar uma mensagem a um outro agente j, terá primeiro de a enviar para o "agente facilitador", que se encarregará de a reencaminhar ao seu destinatário.

  Esta arquitetura resolve parcialmente o problema da coordenação da comunicação e diminui consideravelmente a complexidade necessária aos agentes individuais na realização de comunicação. Os agentes não necessitam de armazenar informações detalhadas sobre todos os outros agentes e nem sequer necessitam de saber o seu endereço de forma a comunicarem com eles. Basta comunicar com o "agente facilitador". No entanto, a existência do " agente facilitador" pode introduzir uma certa centralização no sistema e um estrangulamento (*bottleneck*) no sistema de comunicações. Se este agente deixar de funcionar, o sistema de comunicações deixa também de funcionar.

A comunicação tem dois fins principais: partilha do conhecimento/informação com outros agentes, e coordenação de atividades entre agentes. No entanto a realização de comunicação que permita atingir estas duas metas, requer a definição de uma linguagem comum, caracterizada por:

- **Sintaxe** - A parte da estrutura gramatical da linguagem que contém as regras relativas à combinação das palavras;

- **Semântica** - Significado dos símbolos e das suas combinações;

- **Vocabulário** - Conjunto de símbolos usados;
- **Pragmática** - Conjunto de regras de ação e fórmulas de interpretação dos símbolos utilizados na comunicação;
- **Modelo de domínio do discurso** - Significado que um conjunto de símbolos assume quando interpretado num determinado contexto de conversação.

3 Diagramas UML - Propostos

Nesta secção irão ser apresentados os diagramas propostos pelo grupo no início do desenvolvimento do presente projeto. Estes diagramas propostos foram o resultado da conceptualização inicial advinda do grupo, contudo, estes mesmos diagramas sofreram várias modificações que serão documentadas extensivamente na secção 4.

3.1 Classes do Sistema Multiagente

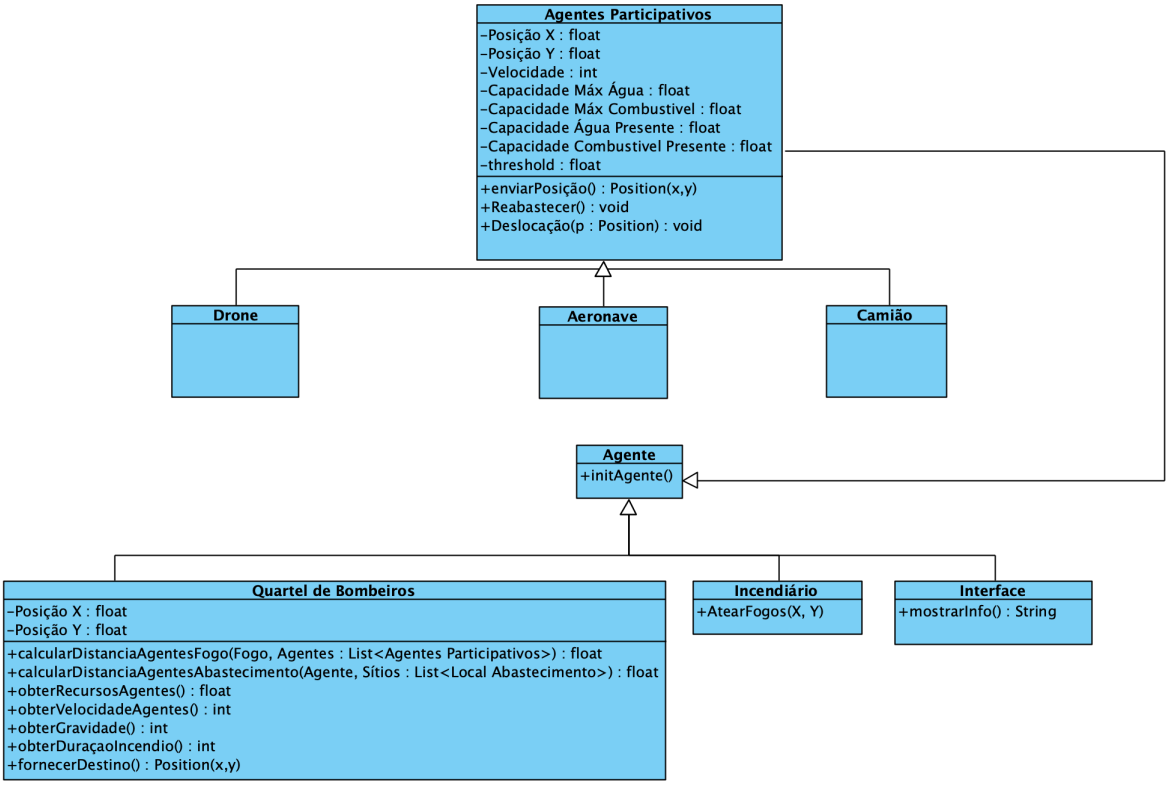

Figura 1: *Diagrama de Classes - Proposto*

Para o desenvolvimento do presente trabalho foi necessário criar várias classes de forma a dar resposta ao problema em questão. Como tal, a arquitetura do projeto tem como base:

- **Agentes Participativos** - Serão os agentes que irão apagar os fogos, ou seja, os Drones, Aeronaves e Camiões. Todos eles terão associados a si uma posição X e Y no mapa. Também terão associados a si uma capacidade máxima de combustível/água tal como a presente capacidade de combustível/água, isto permitirá ao agente *Quartel de Bombeiros* fazer uma gestão dos recursos de cada agente no terreno.
- **Quartel de Bombeiros** - Este é o agente central que irá controlar todos os agentes participativos e verificar a existência de fogos ativos no mapa. Este será responsável por avaliar a distância de todos os agentes participativos até ao incêndio ativo. Após ter todas as distâncias, o Quartel dos Bombeiros deverá escolher o agente participativo que se encontre mais perto do

local do incêndio e ordenar que este apague o incêndio, fornecendo-lhe, para tal, as coordenadas do incêndio. Para tal, o agente central irá verificar se cada agente participativo possui recursos acima ou abaixo do *threshold* definido. Caso possuam poucos recursos, os agentes participativos em causa serão ordenados a irem abastecer.

Toda a gestão relacionada com o incêndio é uma responsabilidade do agente Central, tal como calcular a gravidade do incêndio ou obter a duração do incêndio.

- **Incendiário** - Agente responsável por iniciar os fogos no mapa. Estes serão gerados em coordenadas aleatórias.
- **Interface** - Agente responsável por demonstrar a informação relativa às posições dos agentes participativos e dos fogos ativos no mapa. Na interface também deverá ser possível ver a deslocação dos agentes ao longo do mapa.

3.2 *Workflow*

O sistema irá ser modelado em função do agente incendiário. Todas as ações dos outros agentes são feitas de acordo com este primeiro agente.

Mal os sensores detetem um incêndio, informam o quartel da sua posição e gravidade. Com esta informação, o quartel vai verificar quais são os agentes mais próximos do incêndio e a quantidade de combustível e água que tem disponível.

- Caso o agente tenha recursos suficientes para fazer a viagem, apagar o incêndio e ainda ficar acima do *threshold*, irá ser escolhido para se deslocar e apagar o incêndio (o quartel atualiza no mapa a posição do agente). Quando o incêndio é apagado, o quartel é informado e atualiza o mapa.
- Caso os recursos do agente sejam inferiores ao *threshold*, ele irá deslocar-se à estação de abastecimento mais próxima para reabastecer. Mal chegue ao seu destino, o quartel irá atualizar o mapa.

O *threshold* é uma medida de segurança criada pelos programadores, de modo a que o agente possa ter combustível suficiente para se deslocar para uma estação de abastecimento e poder reabastecer.

Com esta sequência, mesmo que nenhum agente tenha recursos suficientes para se deslocar para o incêndio e o apagar, uma vez reabastecidos, irão ser novamente considerados no cálculo de distância ao incêndio. Deste modo, impede-se que um agente se mova e depois não possa ir reabastecer.

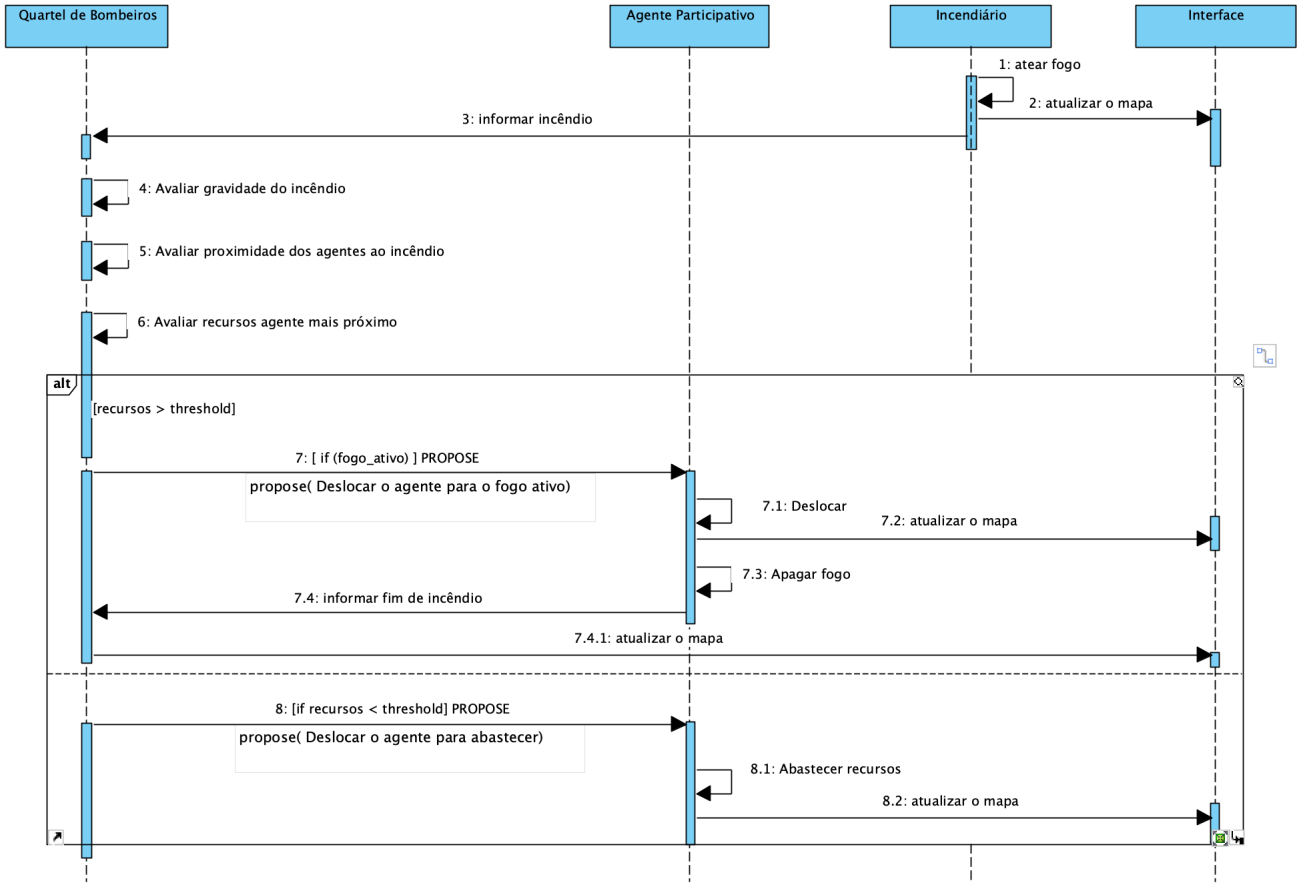

Figura 2: *Diagrama de Sequência - Proposto*

3.3 O Raciocínio dos Agentes Participativos

No diagrama de estados da figura 3, encontra-se o raciocínio dos agentes participativos.

Enquanto que não tem nenhuma ação para executar, o agente está em estado *idle*, ou seja, em estado inativo. Mal o quartel receba informação de que há necessidade de movimento, este faz os cálculos de distâncias no contexto do problema e, mal termine os cálculos, dá ordem de movimento ao agente participativo. Para a ação de movimento, há duas possíveis execuções. Ou um agente se move para apagar um incêndio, ou se move para reabastecer.

Quando o agente chega ao seu destino, seja ele qual for, irá executar a ação adequada (apagar o incêndio ou reabastecer). Uma vez terminada, o agente informa o Quartel do seu sucesso e, mais uma vez, o agente fica inativo, aguardando novas instruções.

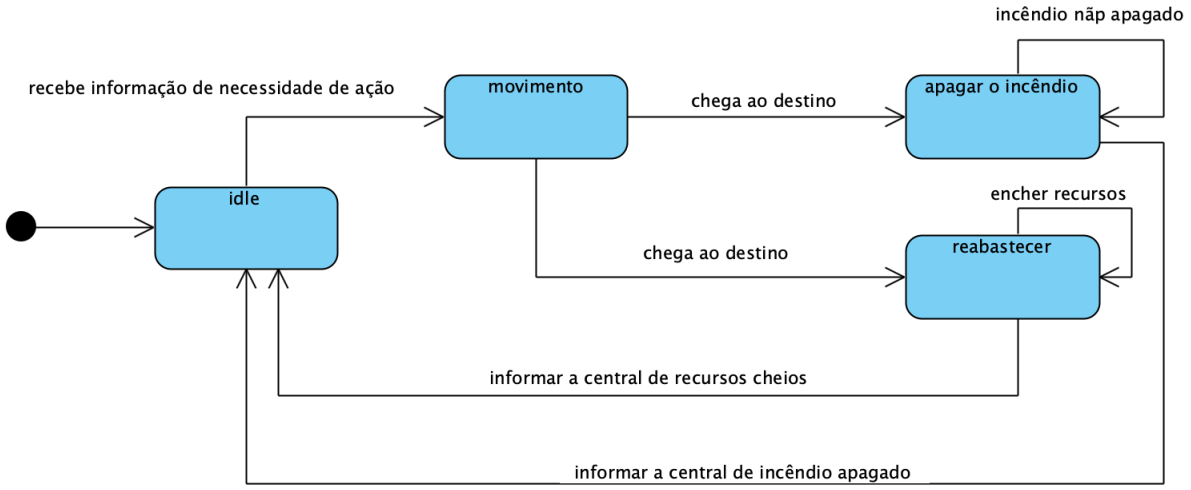

Figura 3: *Diagrama de Estados - Proposto*

4 Diagramas UML - Modificações

No decorrer do desenvolvimento do projeto foram realizadas várias modificações aos planos originais estabelecidos. Após análise do plano proposto pelo grupo, a principal modificação seria dar uma maior autonomia aos Agentes Participativos e diminuir o número de decisões do Agente Central (Quartel).

Ou seja, os Agentes Participativos é que estão responsáveis pela gestão dos seus recursos (água e combustível) e eles próprios é que irão decidir quando é que devem abastecer ou não. O objetivo era melhorar a performance do projeto e tornar cada Agente Participativo mais independente do Quartel para essas tomadas de decisão mais "pequenas". O Quartel continua responsável por designar os Agentes Participativos para apagar os fogos e por fazer os cálculos das distâncias, etc.

Contudo, isso é melhor ilustrado na nova versão do diagrama de classes presente na secção 4.1.

4.1 Classes do Sistema Multiagente

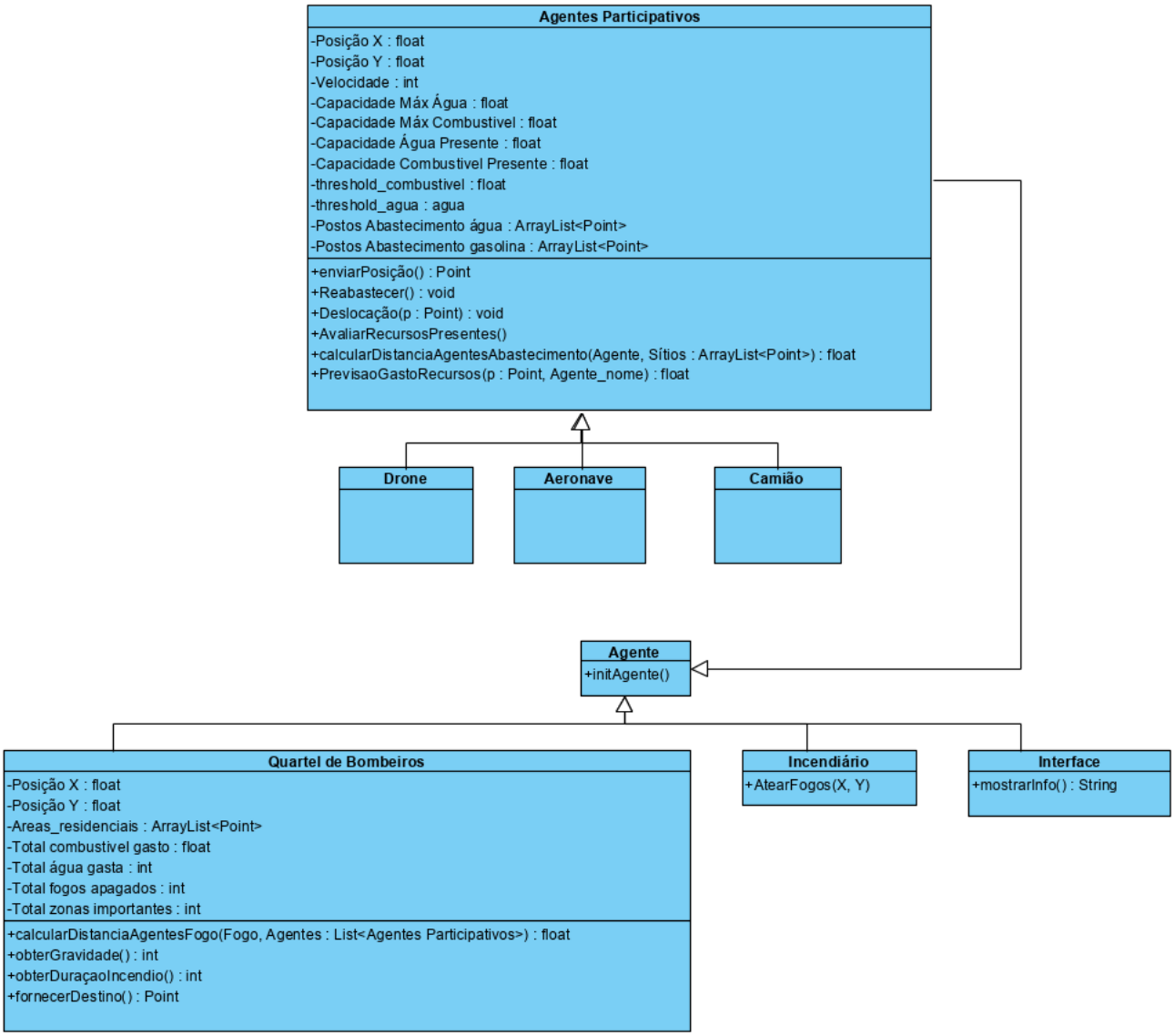

Figura 4: *Diagrama de Classes - Atual*

Como é possível verificar no novo diagrama de classes, foram efetuadas várias modificações durante o desenvolvimento do projeto, devido, maioritariamente, a querer dar uma maior autonomia aos agentes participativos durante o decorrer da simulação. Com a arquitetura desenhada previamente, os agentes participativos dependiam fortemente no agente central (Quartel) para todas as possíveis ações (apagar fogos, deslocação, abastecimento de recursos, etc). Contudo, nesta versão do sistema, é possível os agentes participativos decidirem qual a melhor ocasião para abastecerem os recursos necessários sem estar á espera do *input* do agente central. Ou seja, isto vem aliviar a "carga"de decisões que o agente central tem que tomar e vem distribuir mais uniformemente a tomada de decisão por todos os agentes. Este tópico já tinha sido identificado pelo grupo na apresentação da primeira fase do projeto e foi deixada em aberto esta possível mudança de forma a garantir uma melhor performance da solução.

Também foram feitas mudanças adicionais em relação aos atributos de cada agente, mas em especial o Quartel de Bombeiros. Foram adicionadas variáveis de forma a se poder efetuar as estatísticas dos fogos com sucesso, ou seja, poder contabilizar o tempo de cada fogo (assim como as médias totais), o número de fogos totais assim como o número de fogos em zonas residenciais (sendo estes considerados fogos mais graves).

As zonas/áreas de residência estão a ser geradas e gravadas no Quartel de Bombeiros assim como os Postos de Abastecimento nos Agentes Participativos. Inicialmente o grupo tinha pensado em meter a Interface a gerar os locais e a enviar as localizações por mensagem, contudo após análise essa ação não iria alterar em nada a complexidade da solução nem auferir em nenhuma vantagem pelo que foi decidido "cortar"esse passo extra e os locais estão a ser gerados localmente aos agentes que as necessitam.

4.2 *Workflow*

Mal o fogo seja iniciado é comunicado ao quartel a sua posição. Este depois avalia a gravidade do incêndio tendo em conta as coordenadas do mesmo (verificando se o incêndio não se encontra numa zona residencial).

Os agentes participativos são responsáveis de gerir os seus próprios recursos, portanto quando o Quartel envia uma ordem a um agente participativo para ir apagar um incêndio este não precisa de se preocupar se o agente tem recursos suficientes, pois o Quartel vai agir partindo do pressuposto que todos os agentes têm recursos suficientes. Esta verificação dos recursos, está dentro de um *CyclicBehaviour* pelo que os agentes participantes estarão constantemente a avaliar o seu estado dos recursos (a ver se não estão abaixo do *threshold* definido).

Portanto, a principal modificação da fase do trabalho observada na figura 4, é a autonomia a nível do Agente Participativo na medida em que agora os agentes podem tomar decisões a nível dos recursos sem precisar do *input* do Agente Central. Eles próprios vão regulando o estado das reservas dos seus recursos e vão encher os seus recursos. A informação relativa às localizações dos pontos de abastecimento estão guardadas na "classe-mãe"(Agente Participativo) num ArrayList de Points que são *hard-coded* instanciados no setup da classe. Ainda foi considerado colocar este instanciamento a surgir na classe Interface e depois ser enviado por mensagem para os agentes participativos, mas após conversações com o docente, o grupo chegou á conclusão que esse passo "extra"não iria auferir mais valor ao trabalho pelo que se deixou a instanciação local à classe.

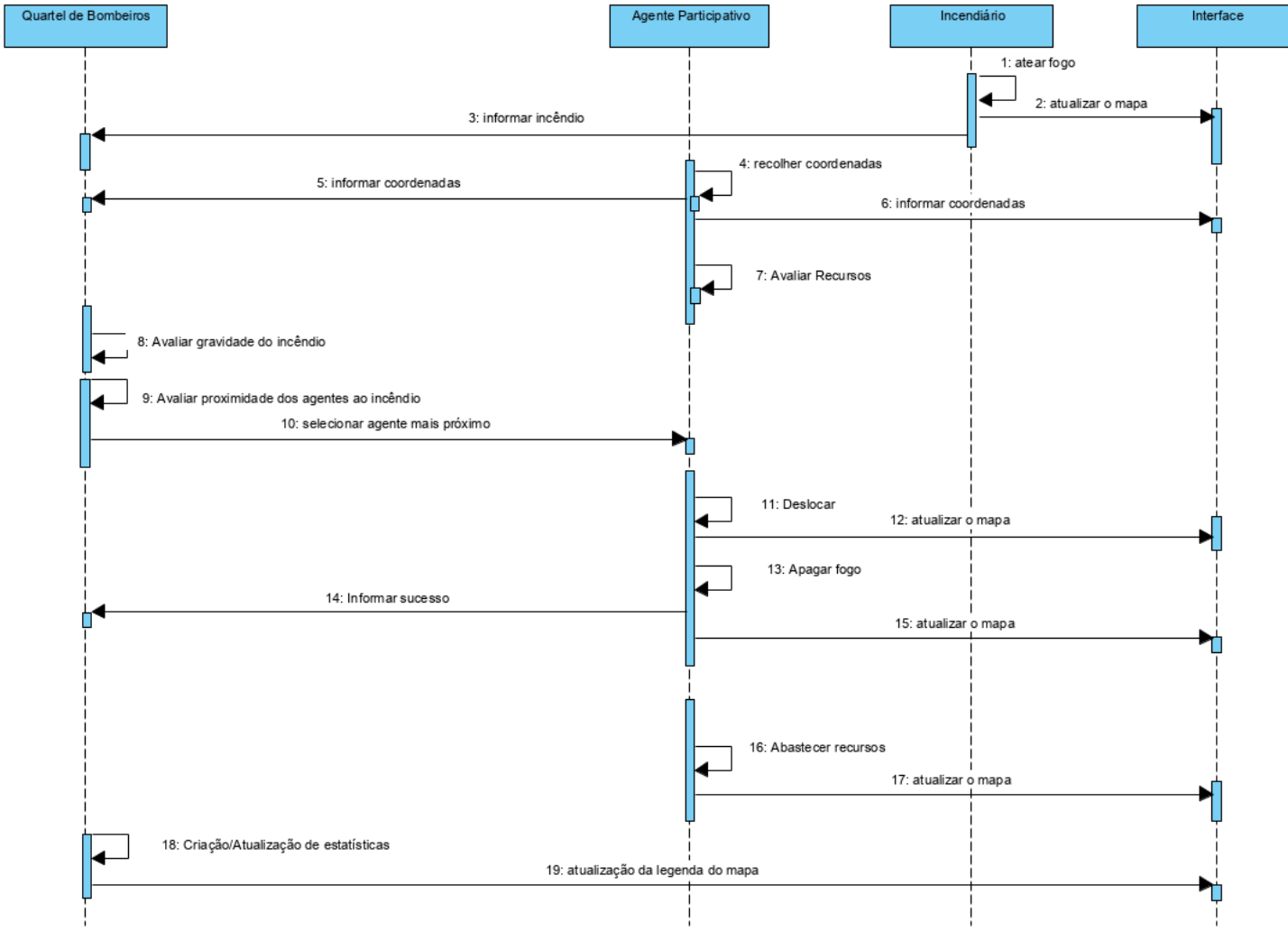

Figura 5: *Diagrama de Sequência - Atual*

4.3 O Raciocínio dos Agentes Participativos

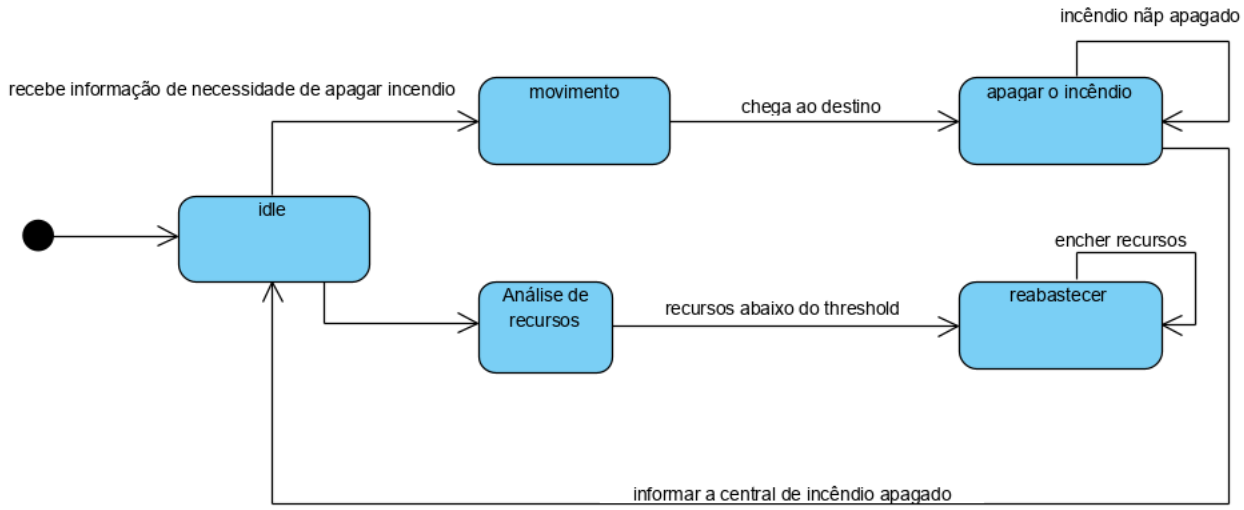

Figura 6: *Diagrama de Estados - Atual*

No diagrama de estados da figura 6, encontra-se o raciocínio dos agentes participativos.

A principal diferença é a de que o Agente Participativo já não recebe uma mensagem vinda do agente central para ir abastecer os recursos. Esta gestão é feita por si próprio, ou seja, ele próprio analisa os recursos existentes e avalia se é necessário ir reabastecer ou não.

5 Lista de *Behaviours* e *Performatives*

Para cada agente, temos vários tipos de *behaviours* e *performatives* usadas, alguns dos quais são comuns a todos.

Classe	*Behaviour*
Agente Participativo	*Ticker Behaviour* • O primeiro *Ticker Behaviour* trata da verificação dos recursos dos agentes. A cada 5 segundos verifica se o agente tem recursos suficientes, mas só o faz caso o agente não se encontre ocupado. Na verificação, ele analisa a informação sobre os recursos e vai tomar uma decisão de acordo com a quantidade de cada um em relação ao limite (*threshold*) pré-definido. Caso haja necessidade de abastecer, o agente vai procurar a estação mais próxima e irá deslocar-se para lá. • O segundo *Ticker Behaviour* serve para informar o quartel das posições dos agentes no mapa.
Aeronave, Camião e Drone	*Cyclic Behaviour* • Este *behaviour* é utilizado na classe *Receiver* dos agentes e vai, tal como o nome indica, receber informação. Esta informação é lhes enviada pelo Quartel, e trata-se de informação relevante à ação de apagar incêndios. Esta mensagem tem como objetivo enviar o agente mais próximo e que tenha recursos suficientes para apagar o fogo ativo e é obtida com a performative **CONFIRM**. No fim, o agente utiliza a performative **CONFIRM** para informar o quartel que o incêndio foi apagado com sucesso.
Incendiário	*Ticker Behaviour* • Este agente é o responsável pela criação dos fogos. Com a ajuda do *Ticker Behaviour*, ele vai criar um incêndio a cada 5 segundos e informa o quartel e a interface com a *performative* **INFORM**.
Interface	*Cyclic Behaviour* • A interface é a responsável pela representação de informação no ecrã. Ela utiliza o *Cyclic Behaviour* da mesma forma que os agentes, mas apenas guarda essa informação para a representar visualmente. Através da performative **INFORM**, recebe a posição do fogo diretamente do agente incendiário.

Quartel	*OneShot Behaviour* • Trata de enviar os agentes para os seus destinos. *Cyclic Behaviour* • Mais uma vez, este *behaviour* é usado para receber mensagens dos agentes. A performative **INFORM** é relativa à mensagem recebida do agente incendiário (criação de incêndio e respetiva posição). Quando recebe esta performative, o quartel trata de chamar os agentes participativos para decidir qual irá ser escolhido para ir apagar o fogo. Depois de escolhido, ele aciona o *OneShot Behavior*, e envia o agente mais próximo. • A *performative* **REQUEST** informa o quartel da localização dos agentes participativos. • A performative **CONFIRM** informa o quartel do sucesso a apagar o incêndio, dos recursos utilizados para o fazer e do tempo gasto.

6 Desenvolvimento

6.1 Código

Nesta secção iremos apresentar alguns excertos de código que sejam mais relevantes para cada classe de forma a poder transparecer de uma melhor forma a capacidade/papel de cada agente.

6.1.1 Classe Agente Participativo (Classe-mãe)

Como sendo a classe "mãe"dos agentes participativos é quase considerada como se fosse uma classe abstracta. Ou seja, possui todos os métodos e atributos em comum de todos os agentes participativos (camião, aeronave e drone) mas nunca é instanciada nem registada nas "páginas amarelas".

A lógica mais importante fica, portanto, inserida nesta classe.

```
//verificar o combustivel gasto
protected float calculaRecursos(float X, float Y) {

float dist = this.calculaDistancia(this.posicaoX, X, this.posicaoY, Y);
float aux = dist/10;

//considera que gasta 1 de combustivel por cada 10 distancia da matriz...
return (aux);

}
```

Listing 1: Verificação do combustível gasto pelo Agente Participativo

```
//calcular distancia
protected float calculaDistancia(float x_inicial, float x_dest,
float y_inicial, float y_dest) {

   float distance = (float) Math.sqrt(((Math.pow((x_dest - x_inicial), 2))
       + (Math.pow((y_dest - y_inicial), 2))));

   return distance;

}
```

Listing 2: Função de calcular a Distância

Como é possível ver no excerto de código em cima, a fórmula do cálculo da distância entre dois pontos é a distância Euclideana.

6.1.2 Classe Camião

```
//registo nas paginas amarelas
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("Camiao");
sd.setName(getLocalName());
dfd.addServices(sd);

try {
    DFService.register(this, dfd);
}catch (FIPAException e) {
    e.printStackTrace();
}
```

Listing 3: Registro do Camião nas páginas amarelas

6.1.3 Classe Quartel

O Quartel é o responsável por encontrar o agente participativo mais próximo do fogo e, consequentemente "chamá-lo"para ir apagar o fogo.

```
protected void calcularAgenteMaisProximo(Collection<Agente_Participativo>agentes_ativos, float
float distancia = 0;

for(Agente_Participativo agente : agentes_ativos) {
 distancia = (float) Math.sqrt((y - agente.getPosicaoY())
       * (y - agente.getPosicaoY()) + (x - agente.getPosicaoX())
       * (x - agente.getPosicaoX()));

 if(distancia < minDistancia) {
       minDistancia = distancia;
    agenteMaisProximo_nome = agente.getAgente_nome();
       agenteMaisProximo = agente.getAid_agente();

 }
}
}
```

Listing 4: Cálculo do agente mais próximo

6.1.4 Classe Incendiário

A única responsabilidade do agente incendiário é iniciar fogos de x em x segundos. Para tal, é utilizado um *TickerBehaviour* para o comportamento ciclico de criação de fogos.

Os fogos são criados com coordenadas aleatórias.

```
	private class Request extends TickerBehaviour{

		public Request(Agent a, long period) {
			super(a,period);
		}

		public void onTick() {

			//Posio aleatria para atear o fogo
			Random rand = new Random();
			int posX = rand.nextInt(100);
			int posY = rand.nextInt(100);

			//enviar ao quartel a posio do fogo
			AID receiver = new AID();
			AID receiver_interface = new AID();
			receiver.setLocalName("Quartel");
			receiver_interface.setLocalName("Interface");

			ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
			msg.setContent(posX + "," + posY );
			msg.addReceiver(receiver);
			msg.addReceiver(receiver_interface);
			myAgent.send(msg);

		}
	}
```

Listing 5: Criação de fogos

6.2 Interface - Camada de Apresentação

A interface é o agente responsável pela representação visual da informação. Para o desenvolvimento desta classe, foi utilizado o *Java Swing* para tratar de desenhar o mapa.

Ver secção 7 para mais informações relativamente à Interface.

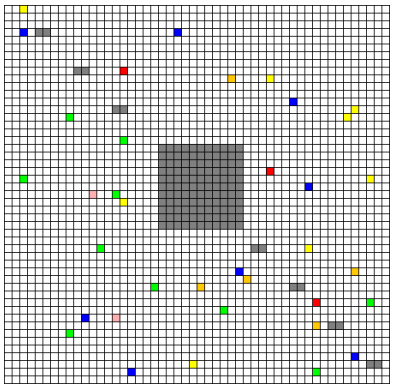

Figura 7: A interface com todos os elementos.

Esta é a interface com todos os elementos existentes desenhados:

- Amarelo - postos de abastecimento de combustível
- Azul - postos de abastecimento de água
- Cinzento - zonas residenciais
- Laranja - camiões
- Rosa - aeronaves
- Verde - drones
- Vermelho - fogos

Os agentes e os fogos são desenhados durante a execução do programa, uma vez que a sua posição varia, enquanto que os postos de abastecimento e as zonas residenciais são desenhadas quando o programa inicia.

6.3 Simulação

Prints do funcionamento da simulação.

```
NOME: Drone0001 ||| PosX: 21 & PosY: 0
NOME: Drone0002 ||| PosX: 39 & PosY: 62
NOME: Drone0003 ||| PosX: 70 & PosY: 58
NOME: Drone0004 ||| PosX: 47 & PosY: 26
NOME: Drone0005 ||| PosX: 8 & PosY: 14
NOME: Drone0006 ||| PosX: 27 & PosY: 77
NOME: Drone0007 ||| PosX: 63 & PosY: 12
NOME: Drone0008 ||| PosX: 51 & PosY: 63
NOME: Drone0009 ||| PosX: 17 & PosY: 79
NOME: Camiao002 ||| PosX: 34 & PosY: 3
NOME: Camiao001 ||| PosX: 7 & PosY: 5
NOME: Drone0010 ||| PosX: 5 & PosY: 38
NOME: Camiao003 ||| PosX: 35 & PosY: 32
NOME: Camiao004 ||| PosX: 60 & PosY: 48
NOME: Camiao005 ||| PosX: 88 & PosY: 63
NOME: Aeronave1 ||| PosX: 8 & PosY: 18
NOME: Aeronave2 ||| PosX: 30 & PosY: 22
```

Figura 8: *Coordenadas dos agentes participativos*

Na figura 8 é possível ver o "arranque"da simulação. Onde todos os agentes Participativos enviam para o quartel a informação relativa às suas coordenadas. As coordenadas são geradas aleatóriamente.

Posteriormente, estas coordenadas são enviadas a cada segundo para que assim o agente Central consiga sempre saber as coordenadas atualizadas de cada um dos seus agentes.

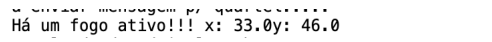

Figura 9: *Informação de fogo ativo*

Posteriormente, como é possível observar na figura 9, é despoletado um fogo pelo agente incendiário. Este comportamento é algo que se repete de 5 em 5 segundos. As coordenadas do fogo são geradas aleatóriamente.

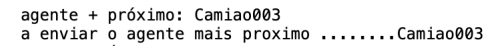

Figura 10: *Selecionado o agente mais próximo ao incêndio*

Como é possível ver na figura 10, depois de receber a informação de um fogo ativo o quartel vai calcular qual o agente participativo que está mais próximo e selecioná-lo para ir apagar o incêndio.

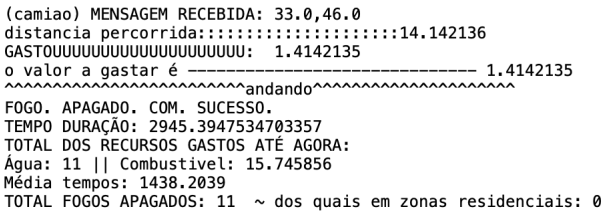

Figura 11: *Deslocação e apagar o incêndio*

Na figura 11 é possível visualizar o momento da receção da mensagem por parte do camião que foi considerado o agente participativo mais próximo do fogo ativo. Imediatamente a seguir é feito o display da informação relativamente á distância percorrida por este agente para o local do incêndio. Também é mostrado o valor do combustível a gastar devido a essa deslocação efetuada. Posteriormente, o agente participativo "caminha"até ao local do incêndio e apaga-o com sucesso. Também é feita uma temporização de desde que o incêndio está ativo (e é recepcionado pelo Quartel) até ao momento em que a mensagem de sucesso é recebida. Portanto, é calculada a diferença entre os dois tempos. O tempo é calculado em milisegundos.

Também, ao fim de cada incêndio apagado com sucesso é apresentado um log de estatísticas. Ou seja, na imagem é possível ver o número total de recursos gastos até ao momento de execução. No caso da imagem, 11 unidades de água e 15.745 unidades de combustivel foram utilizados para apagar 11 incêndios, dos quais 0 foram em zonas residenciais.

O Quartel quando recebe as coordenadas do fogo vindas do incendiário, vai analisar as coordenadas e ver se estas se encontram na área residencial (um ArrayList de Points que estão instanciados na classe Quartel) e se o fogo pertencer a essa lista de Points, o fogo será considerado grave por estar a ocorrer numa zona habitacional.

Para além de a duração do fogo ser imprimida para o ecrã, este valor também é guardado num ArrayList para depois poder-se fazer as médias dos tempos dos fogos, como é possível ver na imagem onde está presente a média da duração dos incêndios.

7 Conclusão

Durante o desenvolvimento do projeto encontramos algumas dificuldades, muitas das quais conseguimos ultrapassar. No entanto, a parte que parecia dar menos problemas acabou por ser a que não conseguimos terminar.

A Interface, é a classe que trata de desenhar a informação no ecrã e, por isso, é importante apenas para transmitir facilmente a informação relevante à simulação ao utilizador. Apesar de termos conseguido implementar o que queríamos, uma das partes essenciais não ficou a funcionar como era esperado. Quando é preciso atualizar o mapa para desenhar os agentes participativos e os fogos, a interface não consegue desenhá-los, sendo que apenas conseguimos ver as suas posições e informação relevante através da informação da consola. Este é um problema que não gostaríamos de ter, uma vez que queríamos ser capazes de mostar essa informação facilmente, juntamente com a representação visual do deslocamento dos agentes.

Apesar deste problema, podemos dizer que conseguimos alcançar os nossos objetivos iniciais para a realização do projeto. Caso tivéssemos mais tempo para realização do trabalho, poderíamos melhorar mais o nosso programa, e é algo que iremos discutir na seção seguinte.

7.1 Trabalho futuro

Devido às restrições de tempo para o desenvolvimento do projeto não foi possível incorporar o JESS no projeto para a tomada de decisões por parte do Quartel, por exemplo. Portanto, algo que poderia acrescentar valor ao projeto seria incorporar o JESS.

Outra coisa que seria importante termos a funcionar seria a Interface. Como mencionamos anteriormente, as restrições de tempo não permitiram a completa implementação desta *feature*, pelo que seria um foco para o trabalho futuro.

Referências

[1] Beatriz López, Bianca Innocenti, and Dídac Busquets. A multiagent system for coordinating ambulances for emergency medical services. *Intelligent Systems, IEEE*, pages 50 – 57, 10 2008.

[2] RoboCup. Agent simulation,http://rescuesim.robocup.org/competitions/agent-simulation-competition/, 2019.