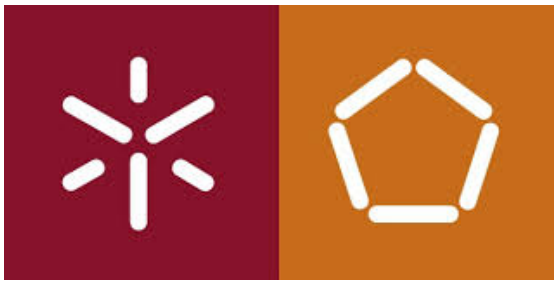

Universidade do Minho - Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Agentes Inteligentes

Trabalho prático
Gestão do fluxo de tráfego aéreo

Carlos José Gomes Campos a74745
José Pedro Ferreira Oliveira a78806
Ludgero da Silva Diogo pg38417

26 de Novembro de 2018

Resumo

Este relatório pretende descrever o desenho da arquitetura da solução e a implementação do sistema multiagente, desenvolvido no âmbito da unidade curricular de Agentes Inteligentes com o intuito de apresentar uma possível solução para o problema de gestão do fluxo do tráfego aéreo, utilizando técnicas de computação baseadas em Agentes.

Conteúdo

1 Introdução

1.1 Contextualização

Este relatório surge no âmbito do trabalho prático da unidade curricular de Agentes Inteligentes, que pertence ao perfil de Sistemas Inteligentes do Mestrado Integrado em Engenharia Informática, da Universidade do Minho.

1.2 Caso de Estudo

O projeto tem por base a concepção de um sistema, desenvolvido num ambiente JADE (Java Agent Development Framework), para monitorizar uma rede de sensores de captura de localização GPS, integrados num ambiente de controlo de tráfego aéreo.

Os intervenientes são as aeronaves e os aeroportos, que devem estabelecer comunicações para a negociação, com o objetivo de automatizar o processo de gestão de viagem das várias aeronaves, tendo em conta um conjunto de variáveis fulcrais que caracterizam este processo.

1.3 Estrutura do Relatório

Este projeto está dividido em duas fases, na primeira é apresentada a parte de modelação do projeto, utilizando a linguagem UML (Unified Modeling Language) adaptada ao novo paradigma de computação baseada em Agentes, na segunda fase é descrito o processo de implementação.

Na secção 2 é elaborado um estado de arte, para contextualização do problema em causa e descrição de alguns sistemas semelhantes ao apresentado neste documento. Na secção 2.2 serão descritos os diagramas de sequência que ilustram as trocas de mensagens entre agentes nos seus diferentes estados. Na secção 2.3 serão explicados os diagramas de atividades de maneira a apresentar a primeira abordagem aos algoritmos usados pelos agentes face à necessidade de negociar/agir com outros elementos do sistema.

De seguida, na secção 2.4 será apresentado o diagrama de estado para o agente aeronave e na secção 2.5 será ilustrado o diagrama de classes idealizado para este sistema multiagente.

Em 3 são apresentadas com mais pormenor as estratégias e os algoritmos utilizados nos diferentes estágios do problema (Descolagem, Viagem e Aterragem). Na secção 4 é apresentado o caso de teste utilizado pelo grupo para testar as funcionalidades do programa.

O relatório termina com as conclusões na secção 5 onde é feita uma reflexão acerca do trabalho realizado bem como do trabalho futuro.

2 Estado de arte

O problema do controlo do fluxo do tráfego aéreo tem vindo a agravar-se devido ao aumento do volume do tráfego em determinadas regiões, a utilização de sistemas multiagente para auxiliar neste processo, é um dos desafios atuais mais importantes para as entidades responsáveis por assegurar estes serviços por todo o globo, em Portugal a empresa NAV, nos Estados Unidos a FAA (Federal Aviation Administration).

Com vista a resolver esta situação, existem várias propostas de sistemas que simulam a gestão do tráfego aéreo, entre os quais podemos destacar: ACES (Airspace Concept Evaluation System) um simulador de espaço aéreo baseado em agentes, que representam os elementos participantes (aeronaves, aeroportos, controladores, etc.) e possui a capacidade de realizar avaliações custo-benefício em várias situações.

O IMPACT (Intelligent agent-based Model for Policy Analysis of Collaborative Traffic flow management) é um modelo baseado em inteligência em grupo entre agentes controladores e agentes aeronaves, usado para avaliar possíveis respostas a falhas de capacidade nos aeroportos (fluxo intenso, condições meteorológicas, recursos insuficientes). Assim, os agentes controladores decidem impor ou não atrasos propositados (GDPs - Ground Delay Programs) e os agentes aeronaves escolhem as ações que minimizam o custo estimado.

O STEAM(Tambe, 1997) tem sido usado para avaliar um sistema colaborativo para sincronização de tráfego em tempo real, através da utilização de apenas agentes controladores (agentes utilizadores não participam nas negociações) do setor e algumas entidades de coordenação de nível superior, coordenam as ações para a resolução de problemas.

Existem várias propostas de sistemas multiagente que seguem diferentes estratégias e combinam com outras técnicas de aprendizagem associadas a inteligência artificial (Swarm-Intelligence, Reinforcement Learning, etc.). No entanto, a maioria tem por princípio a comunicação entre agentes (reativos ou proativos) de maneira a regular as trajetórias das aeronaves, controlando zonas de elevado fluxo através de técnicas de recalculação de trajetórias de aeronaves, utilização de distâncias de segurança entre agentes do sistema (Miles in trail), adoção de programas de atraso propositado (Ground delays), entre outros.

2.1 Modelação

2.2 Diagramas de sequência

Para facilitar a compreensão do leitor, decidimos dividir esta parte da modelação em três diagramas de sequência, em que cada diagrama representa um estágio do processo de viagem do avião (descolagem, período de negociação e aterragem)

Nos diagramas de sequência são ilustradas as mensagens que são trocadas entre agentes, mostrando o tipo de *performatives* (tipo de mensagem) utilizadas e uma breve referência ao seu conteúdo.

2.2.1 Descolagem

A primeira mensagem que o agente aeronave envia é um *request* para o aeroporto em que se encontra, de maneira a verificar se existem condições suficientes para o início da viagem. Caso isso não se verifique, o agente toma uma atitude persistente e continua a enviar o pedido periodicamente, até que receba um *confirm* da estação.

Posteriormente, é executado um processo semelhante com a estação destino e por consequência da confirmação deste último pedido, o avião inicia a viagem (utilizando uma performative *request-when* para pedir, e uma performative *agree* para confirmar). Por último, este envia um *inform* à estação local a informar que descolou.

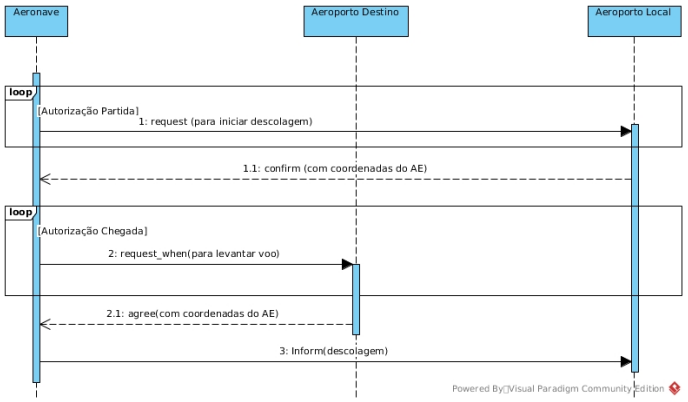

Figura 1: Diagrama sequência que descreve a descolagem

2.2.2 Negociação em aeronaves

Durante o voo, os agentes aeronave enviam constantemente as suas coordenadas a outras aeronaves que estejam em processo de viagem (utilizando performatives *inform*) e recebem coordenadas dos mesmos. Assim, é possível descobrir se estão numa rota de colisão e dependendo do rumo dos aviões, tomar medidas evasivas tais como, abrandar, acelerar e virar. Posteriormente, caso seja

verificado uma modificação de trajetória/velocidade, são enviadas ao aeroporto destino as características da viagem que sofreram alteração no processo de negociação, para que seja recalculada a lista de prioridades de aterragem (utilizando uma performative *inform-if*).

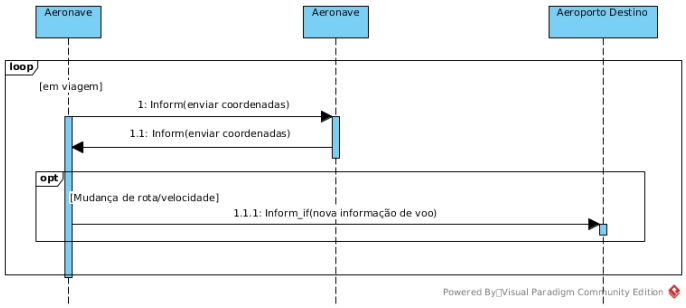

Figura 2: Diagrama sequência que descreve a viagem

2.2.3 Aterragem

Quando o avião entra na área de aproximação do aeroporto destino envia um *propose* para aterrar, ao qual a estação responde com uma confirmação através de um *accept-proposal*, cujo conteúdo é a velocidade que o avião deve adquirir nesta fase final.

Quando o mesmo aterra e termina a sua viagem, informa a estação de tal acontecimento, para que sejam geridos os recursos do aeroporto usados neste processo.

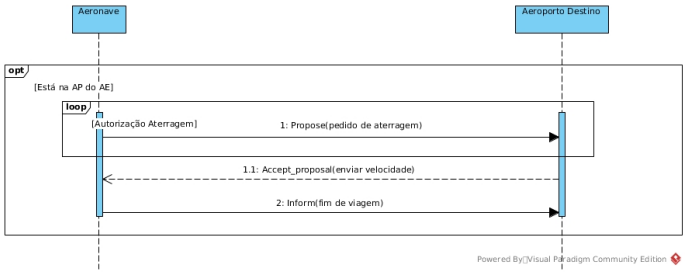

Figura 3: Diagrama sequência que descreve a aterragem

2.3 Diagramas de atividade

Nestes diagramas de atividade são explicados os principais processos executados pelos agentes que advêm da troca de mensagens entre si, pretende-se assim descrever com mais rigor as atividades internas dos agentes.

Tal como nos diagramas de sequência, também foi feita uma divisão em três estágios do processo de viagem.

2.3.1 Descolagem

Inicialmente, o agente aeronave verifica a condição meteorológica do aeroporto em que se encontra e verifica a disponibilidade de pistas, só prossegue caso estas condições sejam verificadas. Caso isso se verifique, pede ao aeroporto destino autorização para descolar, sendo que o agente estação só dá luz verde se tiver lugar para estacionamento e se as condições meteorológicas forem suficientes para uma aterragem em segurança.

Caso alguma destas condições não se verifique, a aeronave persiste no envio do pedido à estação origem ou à estação destino.

Ao mesmo tempo que faz o pedido ao aeroporto destino, é enviada a informação relativa à viagem, que são as coordenadas iniciais e a velocidade, para que a estação destino possa calcular a prioridade de aterragem da aeronave.

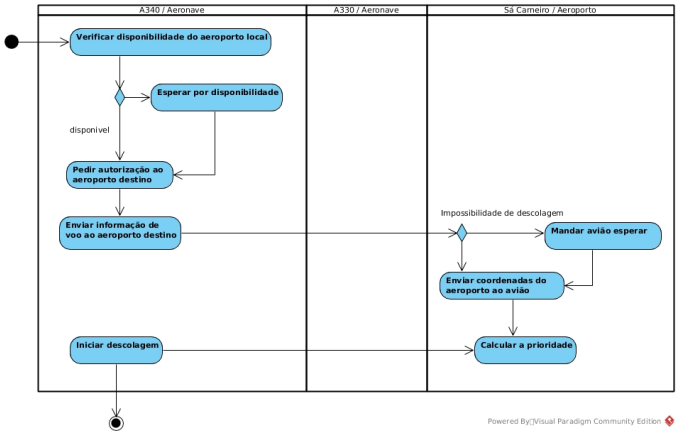

Figura 4: Diagrama de atividade que descreve a descolagem

2.3.2 Negociação entre aeronaves

No decorrer da viagem, cada avião envia em broadcast as suas coordenadas para todos os outros aviões em processo de viagem. Simultaneamente, está constantemente a receber as coordenadas dos outros aviões, para efetuar o cálculo da distância entre ambos e assim descobrir se outros aviões estão na sua área protegida ou não.

Caso o avião esteja na área de proteção de outro, o avião verifica se existe a possibilidade de cruzamento de rotas, assim verifica se a direção da rota do outro avião é a mesma que a sua e o sentido é o oposto, ou seja, verificar a existência de uma colisão frontal, neste caso o avião efetua um ligeiro desvio à direita.

Caso exista apenas cruzamento de rotas, o avião que estiver mais perto do destino diminui a velocidade, e o que está mais longe do seu destino, aumenta.

Se existir alguma mudança de rota/velocidade, o avião informa a estação destino que tal aconteceu enviando os novos parâmetros que sofreram alterações,

para que esta possa recalcular a prioridade de chegada.

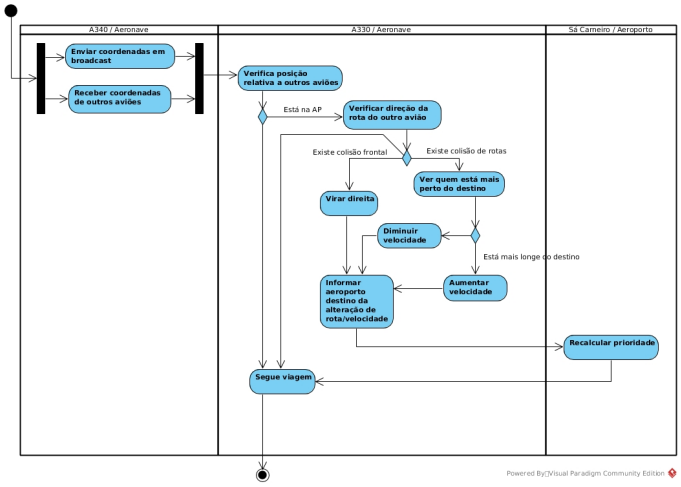

Figura 5: Diagrama de atividade que descreve a viagem

Este diagrama de atividades é visto apenas de uma perspetiva, mas ambos os aviões fazem o cálculo da distância e tomam uma decisão que evita o choque que é independente da decisão do outro. Deste modo, o sistema torna-se mais tolerante a falhas, como por exemplo para caso da troca de uma das mensagens não ocorrer.

Estas decisões são independentes, pois se ambos os aviões estiverem prestes a colidir de frente, ambos fazem um desvio para a sua direita. Caso tenham apenas cruzamento de rotas eles tomam uma decisão baseada em factos absolutos, como é a variável da distância a percorrer.

Esta negociação é um pouco individualizada, porque os aviões apenas recebem e enviam informação referente a localizações e com base nisso tomam decisões que evitam colisões, mas não passam por um período de "discussão"para ver quem é que faz o quê.

2.3.3 Aterragem

O período de aterragem começa quando o agente aeronave entra na área de proteção do agente estação destino. Quando isto acontece, a aeronave pede autorização para aterrar, de seguida o agente estação indica a velocidade que a aeronave deve seguir, tendo por base a lista de prioridades previamente calculada. Deste modo, é possível evitar colisões entre aeronaves e facilitar a gestão dos recursos do aeroporto.

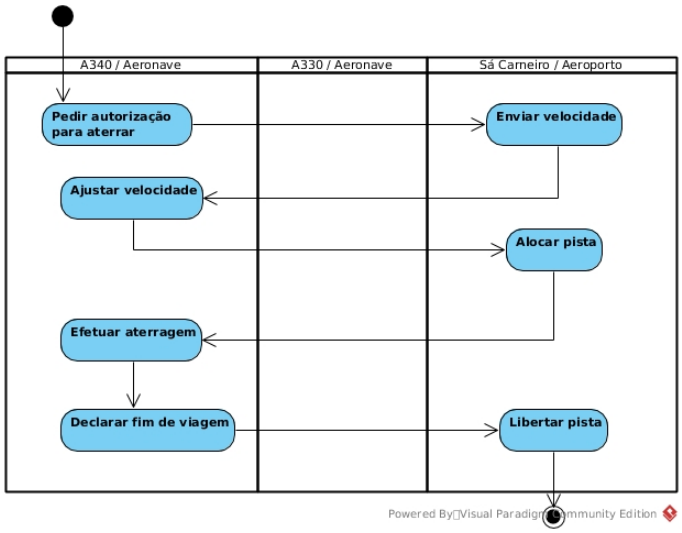

Figura 6: Diagrama de atividade que descreve a aterragem

2.4 Diagramas de estado

Este diagrama tem como objetivo representar os possíveis estados de comunicação e as transições entre os mesmos que o agente aeronave pode ter ao longo do seu ciclo de vida no sistema. A inicialização de um agente aeronave passa pela sua associação a um aeroporto, sendo essa a sua posição inicial e aí permanece no estado *Parado* até que seja efetuada uma viagem. Ainda neste estado pode comunicar com as estações com o intuito de iniciar uma viagem.

Quando se encontra em viagem, o Agente Aeronave exibe dois comportamentos, no estado *Partilhar Posição* divulga a sua posição aos restantes AA's e quando for detetado um conflito o agente passa ao estado *Negociar com AA*, onde procura chegar a um entendimento com o outro agente, voltando ao estado anterior quando a situação estiver resolvida. Em paralelo, este agente terá a possibilidade de comunicar com o aeroporto destino, de maneira a informar as eventuais alterações às características da viagem que tenham sido modificadas no processo de negociação de conflitos (velocidade, trajetória, etc.), e quando está em processo de aterragem.

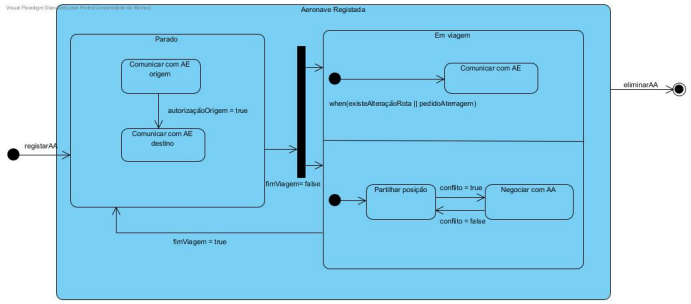

Figura 7: Diagrama de estado de um agente aeronave

2.5 Diagrama de classes

Pelo diagrama de classes podemos observar como será a estrutura da solução que estamos a propor. Neste caso, podemos ver que existem três classes: a classe *AgenteAeronave*, que representa os aviões, a classe *AgenteEstacao*, que representa os aeroportos e a classe *AgenteInterface* que mostra ao utilizador como os aviões se movimentam entre diferentes aeroportos. Estas três classes são especificações da classe Agent, que é uma classe disponibilizada pela biblioteca JADE.

É de notar que o Agente Aeronave possui um conjunto de classes internas por composição, *ReceberMsg* (receber mensagens tanto das aeronaves como das estações) e *Aterragem* (realizar o processo de aterragem) que são especificações da classe abstrata *CyclicBehaviour*, estas classes adquirem um comportamento cíclico que executa o método *action()*. Para além destas, temos a classe *Descolagem*(realizar o processo de descolagem), *EnviarCoords*(enviar periodicamente a localização às aeronaves em viagem) e *Movimento*(simular movimento

da aeronave), especificações da classe abstrata *TickerBehaviour* e adquirem um comportamento periódico através do método *onTick()*.

O Agente Estação possui uma classe *ReceberPedidos* que representa um comportamento cíclico para a receção de todas as mensagens provenientes das aeronaves. O Agente Interface possui um comportamento cíclico *ReceberCoords* responsável por escutar as mensagens com as informações dos aviões mais relevantes para expor ao utilizador.

O agente Estação também possui três funções privadas, que tratam da lista de prioridades. Tanto o adicionaLista e o removeLista, fazem a inserção e a remoção, respetivamente, de forma ordenada num ArrayList de String, em que cada posição tem o nome do agente concatenado com o tempo de espetativa da viagem. O getVelocidade devolve o índice da informação correspondente ao avião que quer aterrar no momento em que este método é chamado.

Em cada classe, estão representadas as variáveis de instância e os métodos mais relevantes.

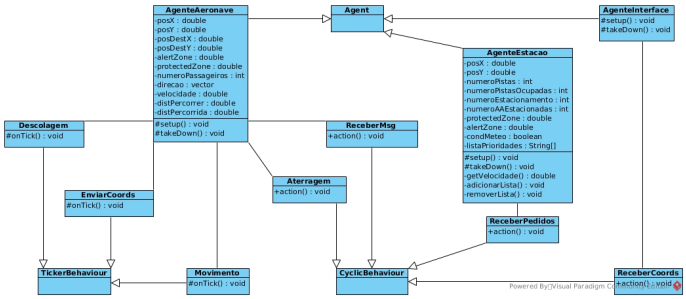

Figura 8: Diagrama de classes

3 Implementação

3.1 Descolagem

3.1.1 Agente Aeronave

O processo de descolagem é tratado no comportamento *TickerBehaviour Descolagem*. Periodicamente, a aeronave envia uma mensagem com o *Performative Request* para o aeroporto local até obter autorização, quando isso acontece passa a enviar uma mensagem com o *Performative Request_ When* para a estação destino até obter uma resposta. Esta mensagem contém a informação relativa ao voo, ou seja, o nome da aeronave, as coordenadas do local de onde vai iniciar a viagem e a velocidade média que vai adquirir. No momento que tem as autorizações das duas estações, envia um *Inform* para a estação origem com o intuito de informar o momento de saída dessa estação.

Deste modo, pretendemos garantir que o avião só descola se tiver possibilidades de aterrar no destino, evitando assim que ande aos círculos em volta da estação destino, à espera dessas condições.

O motivo pelo o qual nos levou a escolher este comportamento como *TickerBehaviour*, foi para que os agentes aeronaves possam executar várias viagens ao longo do seu tempo de vida.

3.1.2 Agente Estação

No processo de descolagem de uma aeronave, o agente estação vai receber dois tipos de pedidos da aeronave, que serão processados pelo *CyclicBehaviour receberPedidos*.

- Para o caso em que a mensagem recebida tem um *Performative Request*, é verificado se existem condições meteorológicas favoráveis para a saída da aeronave da estação atual e se o número de pistas ocupadas não ultrapassa o total de pistas existentes. Se assim for, é feita uma reserva da pista para descolar e é decrementado o número de estacionamentos ocupados. De seguida, é enviada uma mensagem com o *Performative Confirm* ao agente aeronave com as coordenadas da estação atual em que este se encontra (neste caso, não seria completamente necessário o envio desta informação, pois a aeronave já sabe à partida a sua última posição. No entanto, foi implementado para que os aviões possam fazer várias viagens).

```
[Excerto do tratamento do pedido request]

if (msg != null &&
    msg.getPerformative() == ACLMessage.REQUEST &&
    condMeteo &&
    nrPistasOcupadas < totalPistas) {
 nrPistasOcupadas++;
```

```
nrEstOcupados--;
resp.setContent(coordX+";"+coordY);
resp.setPerformative(ACLMessage.CONFIRM);
send(resp);
}
```

- No caso da mensagem ser um *Performative Request_ When*, é verificado se existem estacionamentos disponíveis para receber a aeronave e se as condições meteorológicas são favoráveis no aeroporto destino. Caso isso se verifique, é feita uma reserva para estacionamento no aeroporto destino é adicionada à lista de prioridades a aeronave que efetuou o pedido, tendo por base o tempo que esta vai demorar a chegar. De seguida, é enviado um *Performative Agree* para a aeronave com as características da estação destino (a dimensão da área de alerta, área protegida, as coordenadas e a distância a percorrer).

3.2 Em viagem

3.2.1 Agente Aeronave

No decorrer da viagem cada aeronave envia um conjunto de informações às restantes aeronaves que se encontram em voo através do *TickerBehaviour EnviarCoords*.

É de salientar que toda a informação enviada pelo avião, é previamente calculada por ele e não por outros.

```
[Excerto que descreve a informação enviada em broadcast]
...
    msg.addReceiver(provider);
msg.setContent(coordX+";"+coordY+";"+direcaoX+";"+direcaoY+";"
+distPercorrer+";"+destCoordX+";"+destCoordY+";"+origemCoordX+";"
+origemCoordY+";"+nrPassageiros);
send(msg);
}
```

Estas informações são fundamentais no processo de negociação caso existam conflitos na trajetória entre aeronaves.

Através do *CyclicBehaviour ReceberMsg* cada aeronave consegue receber as informações dos voos, enviadas pelas restantes aeronaves e prever situações de risco envolvendo diversos cenários.

No caso em que existe a possibilidade de choque frontal, tal como foi referido na secção da modelação, ambos os aviões fazem um desvio à sua direita, e direcionam-se de novo para o seu destino, tal como vemos na Figura 9.

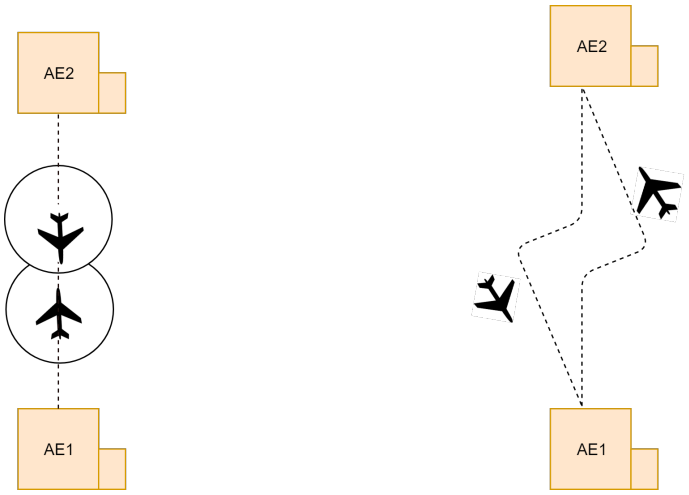

Figura 9: Conflito frontal entre aeronaves

```
[Excerto que descreve o desvio à direita, quando existe colisão frontal
(Comportamento ReceberMsg)]
...
if(origemCoordX==destXOutro && origemCoordY==destYOutro &&
origemXOutro==destCoordX && origemYOutro==destCoordY) {
    double rx = (direcaoX * Math.cos(Math.toRadians(-45))) -
    (direcaoY * Math.sin(Math.toRadians(-45)));
    double ry = (direcaoX * Math.sin(Math.toRadians(-45))) +
    (direcaoY * Math.cos(Math.toRadians(-45)));
    direcaoX = rx;
    direcaoY = ry;
    alterouDir=true;
}
```

Consideramos que existe choque frontal, quando a origem de um avião é o destino do outro e vice-versa, isto supondo que os aviões não voam por cima de aeroportos, a menos que aterrem lá.

O desvio é simplesmente a rotação do vetor direção do avião, em 45º para a direita. O valor da flag *alterouDir* é alterado, para que depois possa ser feito o cálculo do redirecionamento do avião para o destino.

```
[Excerto que mostra o cálculo do redirecionamento do avião à origem
(Comportamento Movimento)]
...
if(alterouDir) {
distPercorrer=Math.sqrt(((Math.pow((destCoordX - coordX), 2))
+ (Math.pow((destCoordY - coordY), 2))));
direcaoX=(destCoordX-coordX)/distPercorrer;
```

```
direcaoY=(destCoordY-coordY)/distPercorrer;
tempoFalta=distPercorrer/velocidade;
}
```

Por fim, temos o caso em que existe uma colisão de rotas, quando isto acontece os aviões alteram a sua velocidade, consoante a sua posição em relação ao destino.

Por outras palavras, o avião que estiver mais perto do destino diminui a velocidade e o outro aumenta, tal como é mostrado na Figura 10

O método usado para saber se existe colisão de rotas passa pelo cálculo do ângulo formado pelos vetores de direção de cada avião.

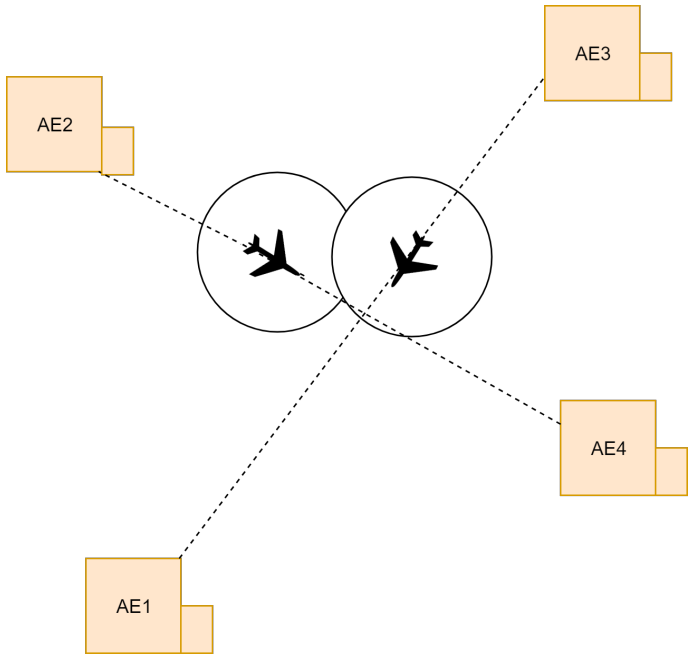

Figura 10: Conflito com cruzamento de trajetórias entre aeronaves

```
[Excerto que mostra a alteração de velocidade, quando existe colisão
de rotas (Comportamento Movimento)]
...
}else if((ang!=0 && ang!=Math.PI) && (destCoordX!=destXOutro ||
destCoordY!=destYOutro) && (origemCoordX!=origemXOutro ||
origemCoordY!=origemYOutro)) {
    ...
    if(distOutro>distPercorrer && velocidade<zonaProtegidaEstacao+
    zonaProtegidaEstacao/2) {
     velocidade+=zonaProtegidaEstacao/2;
     alterouVel=true;
```

```
    } else if(distOutro<distPercorrer && velocidade>zonaProtegidaEstacao-
    zonaProtegidaEstacao/2){
     velocidade-=zonaProtegidaEstacao/2;
     alterouVel=true;
    }
...
```

É considerado que existe conflito entre rotas quando o ângulo entre elas é diferente de 0^o (sendo 0^o, os aviões vão a par um do outro) ou quando é diferente de 180^o (sendo 180^o, os aviões vão em direções opostas, mas não um contra o outro) e se os seus destinos e origens são diferentes. As velocidades só variam dentro de um intervalo pré-estabelecido.

Quando uma destas duas hipóteses acontece, é enviado para a estação o tempo decorrido, e o tempo que ainda falta para chegar ao destino, para que esta possa recalcular as prioridades de chegada.

```
[Excerto que mostra o envio dos tempos (Comportamento Movimento)]
...
if(alterouDir || alterouVel) {
    try {
        ...
        sd.setType(aes.get(conta+1));
                ...
                ACLMessage(ACLMessage.INFORM_IF);
                ...
                if(alterouDir) {
                    distPercorrer=Math.sqrt(((Math.pow((destCoordX - coordX), 2))
                    + (Math.pow((destCoordY - coordY), 2))));
                    direcaoX=(destCoordX-coordX)/distPercorrer;
                    direcaoY=(destCoordY-coordY)/distPercorrer;
                    tempoFalta=distPercorrer/velocidade;
                }
                msg2.setContent(name+";"+tempoDecorrido+";"+tempoFalta);
                send(msg2);
    ...
    alterouDir=false;
    alterouVel=false;
}
...
```

3.2.2 Agente Estação

Quando a estação recebe uma mensagem com a performative do tipo *Inform_If*, remove o elemento da lista de prioridades correspondente àquele avião e adiciona-o de novo com os novos tempos de viagem. Isto só é possível porque tanto a remoção, como a inserção implementadas, garantem a ordenação da lista de prioridades.

```
else if(msg != null && msg.getPerformative() == ACLMessage.INFORM_IF) {
...
removeLista(coordsAviao[0]);
adicionaLista(temp1+temp2,coordsAviao[0]);
}
```

3.2.3 Agente Interface

O Agente Interface recebe as coordenadas do Agente Aeronave, assim é possível mostrar através de um gráfico em tempo real (com o uso da biblioteca JFreeChart) a posição dos mesmos no espaço, bem como a sua área protegida, com as posições conhecidas do Agente Estação é possível fazer o mesmo para este.

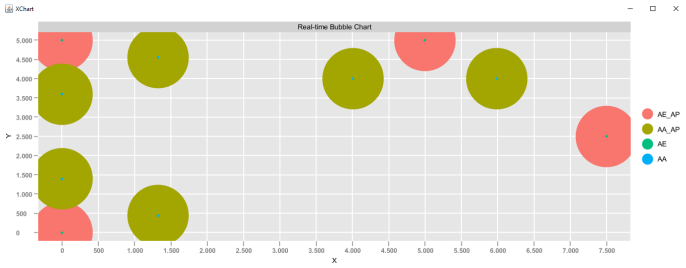

Figura 11: Interface gráfica

3.3 Aterragem

3.3.1 Agente Aeronave

A aterragem de uma aeronave é gerida no *TickerBehaviour Aterragem*, periodicamente, enquanto a aeronave se encontra em viagem, é verificado através da posição atual da aeronave se esta se encontra dentro da área de proteção do aeroporto destino, através da fórmula da distância. Caso isso se verifique, é enviada uma mensagem com o *Performative Propose* ao aeroporto destino com o pedido de permissão de aterragem.

De seguida, é verificado quando é que de facto o fim da viagem ocorre, sendo que a aeronave passa a verificar se já se encontra dentro da zona de alerta do aeroporto (o grupo assumiu que o fim de viagem seria quando fosse atingida a área de alerta do aeroporto, para prevenir erros em que o avião podia ultrapassar o ponto exato do aeroporto). Quando esta zona for atingida, é enviada uma mensagem com o *Performative Inform* que indica à estação o fim de viagem, permitindo que seja feita a gestão dos recursos da mesma.

Finalmente, são atualizadas uma série de variáveis para o início de outra possível viagem.

3.3.2 Agente Estação

No *CyclicBehaviour receberPedidos* é processada a aterragem de cada agente que tem como destino uma determinada estação, aqui serão recebidos dois tipos de pedidos:

- No caso em que recebe um *Propose* a estação procede à reserva de uma pista para a aeronave aterrar (incrementa o total de pistas ocupadas), bem como ao cálculo da velocidade que esta deve adquirir para que seja respeitada a lista de prioridades de aeronaves em viagem. O cálculo efetuado tem por base o índice da posição na lista de prioridades em que a aeronave está e que vai servir de fator multiplicativo no ajuste da velocidade. Posteriormente, é enviada uma mensagem com o *Performative Accept_ Proposal* e que contém a velocidade anteriormente referida.

```
[Excerto que descreve a ação da estação a um Propose]

if(msg != null &&
msg.getPerformative() == ACLMessage.PROPOSE &&
nrPistasOcupadas<totalPistas) {

 nrPistasOcupadas++;
 resp.setPerformative(ACLMessage.ACCEPT_PROPOSAL);
 double vel=zonaProtegida-10*getVelocidade(msg.getContent());
 resp.setContent(vel+"");
 send(resp);
}
```

- Para o caso em que seja recebido um *Inform*, a estação desaloca a pista que foi usada para aterrar (decrementa o número de pistas ocupadas) e remove da lista de prioridades a aeronave em questão.

```
[Excerto que descreve a ação da estação a um Inform]
if(msg != null && msg.getPerformative() == ACLMessage.INFORM) {
    String n=msg.getContent();
    removeLista(n);
    nrPistasOcupadas--;
}
```

4 Caso de teste

Para testar o programa elaboramos uma topologia com quatro estações e seis aeronaves, de acordo com a figura 12. Com este sistema simples foi possível verificar que os agentes tomaram as decisões esperadas neste contexto. Para verificar a escalabilidade do programa, foi testado o mesmo cenário mas com um número maior de aeronaves, tendo corrido tudo dentro do esperado.

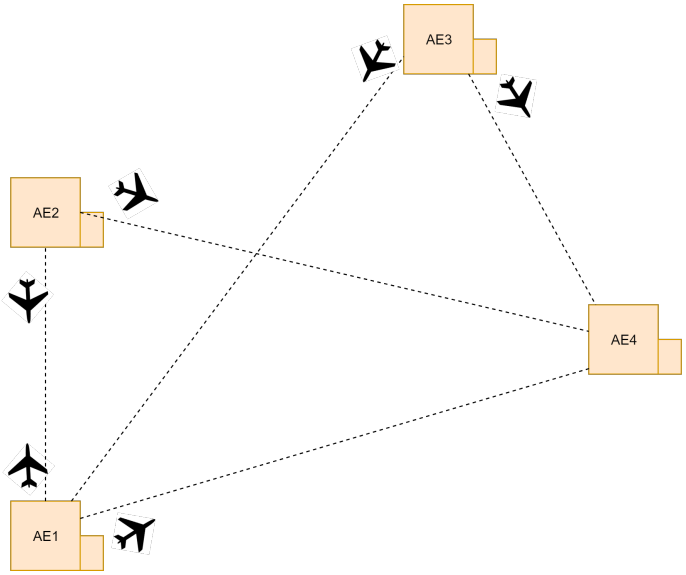

Figura 12: Topologia de teste

5 Conclusão

Após a realização deste projeto, o grupo adquiriu conhecimentos relativos aos principais conceitos de computação baseada em Agentes, aplicando-os na conceção de uma arquitetura para a resolução de um problema real, como é o caso do controlo do fluxo do tráfego aéreo.

Inicialmente, foi feita uma pesquisa acerca do problema relatado no enunciado e verificou-se a existência de sistemas multiagente atualmente a serem testados por várias entidades, com objetivo de auxiliar o controlo do fluxo do tráfego aéreo por meio de agentes inteligentes. Assim, analisando as várias arquiteturas, foi possível ter um ideia das técnicas já utilizadas e das vantagens e desvantagens de cada uma.

Na primeira fase do projeto, idealizou-se o sistema de software a produzir, através da modelação UML e da variação da linguagem AUML, foi possível representar com algum detalhe as estratégias para a resolução dos desafios apresentados. No que diz respeito às estratégias desenvolvidas, o grupo tentou simplificar os processos de negociação entre agentes, tornando-os proativos e cooperativos.

Na segunda fase, a implementação foi relativamente fácil, uma vez que os membros do grupo, se sentiam à vontade com o desenvolvimento de software na plataforma JADE, tendo por base os conhecimentos adquiridos ao longo das aulas. O processo mais difícil de implementar foi a negociação entre aeronaves, mais precisamente as manobras evasivas que cada aeronave deverá efetuar, face às várias situações de conflito.

É de realçar que esta solução possui a capacidade implementar vários aviões de tipos diferentes, isto é, aviões com velocidades e áreas de aproximação diferentes, o que torna este programa mais flexível. O mesmo pode ser dito para as Estações, que podem ter vários números de pistas e estacionamentos. Para tal, basta adicionar mais argumentos no Container, e trata-los nas funções *setup()* respetivas.

Visto que na vida real, uma estação comunica com milhares de aeronaves por dia, tentámos diminuir ao mínimo a sua carga nas comunicações, fazendo com que as aeronaves enviem apenas mensagem para descolar, para aterrar e quando alteram as suas rotas/velocidades.

Ao tornar este sistema descentralizado, visto que não existe uma entidade que faz o controlo de todo o processo, tornamos a sua escalabilidade possível.

Em suma, o grupo faz uma apreciação positiva do trabalho realizado, pois considera que apesar de algumas falhas o objetivo principal foi atingido com sucesso.

Referências

[1] Shawn R. NASA, Peter A. Jarvis Perot Services Government Systems, Francis Y. Enomoto NASA, Maarten Sierhuis USRA-RIACS, Bart-Jan van Putten USRA-RIACS, Kapil S. Sheth NASA. A Multi-Agent Simulation of Collaborative Air Traffic Flow Management `https://ti.arc.nasa.gov/m/pub-archive/1425h/1425%20(Wolfe).pdf`

[2] Adrian K. Agogino University of California Santa Cruz, Kagan Tumer Oregon State University. A Multiagent Approach to Managing Air Traffic Flow