ASMa

Universidade do Minho
Escola de Engenharia
Mestrado em Engenharia Informática

Agentes e Sistemas Multiagente

Ano lectivo 2023/2024

Sistema Multiagente para controlo de tráfego aéreo

Grupo 8

Miguel Silva Pinto, PG54105

Orlando José da Cunha Palmeira, PG54123

Pedro Miguel Castilho Martins, PG54146

Braga, 17 de Maio de 2024

Índice

Índice de figuras

1 Introdução

Este relatório foi elaborado no âmbito do projecto prático da unidade curricular de Agentes e Sistemas Multiagente e tem como objectivo descrever a concepção e implementação de um sistema multiagente que implementa um ambiente virtual de controlo de tráfego aéreo.

Este sistema multiagente foi implementado na linguagem de programação Python, utilizando a plataforma SPADE para o desenvolvimento de agentes.

1.1 Caso de estudo

A temática escolhida pelo grupo no trabalho de investigação preliminar a este projecto foi no âmbito dos transportes inteligentes. Desta forma, o grupo decidiu implementar um sistema multiagente de controlo de tráfego aéreo composto por vários agentes responsáveis por coordenar a dinâmica de um ecossistema de tráfego aéreo constituído por aeroportos em diversas cidades.

O ambiente virtual será constituído por diferentes aeroportos em várias cidades, e os aeroportos terão de gerir os seus recursos para permitir um fluxo organizado do tráfego aéreo que lhes é requisitado. Para que este ambiente apresente um tráfego aéreo organizado, é requerido que obedeça a certas restrições, como a ocupação dos hangares dos aeroportos, o número de *runways* disponíveis e as condições meteorológicas em cada aeroporto.

1.2 Principais objectivos

No que diz respeito aos objectivos do sistema multiagente proposto, o grupo pretende implementar um sistema cooperativo onde os diversos agentes interagem para permitir a realização de viagens entre aeroportos. Assim, estabelecemos os seguintes objectivos:

- **Cumprimento das viagens**: As viagens programadas pelo sistema devem ser realizadas com sucesso.
- **Eficiência operacional**: Promover uma utilização equilibrada dos recursos fornecidos (aviões) pelos aeroportos de modo a permitir uma redução dos atrasos e dos tempos de espera.
- **Adaptabilidade**: O sistema deve ter a capacidade de lidar com situações adversas, tais como: (i) condições meteorológicas que impeçam aterragens/descolagens, (ii) sobrelotação dos hangares ou pistas e (iii) falta de aviões nos hangares para permitir a concretização de um voo.
- **Coordenação entre Agentes**: Os agentes devem ser capazes de comunicar e cooperar eficazmente uns com os outros para coordenar a deslocação das aeronaves de modo a evitar incoerências/erros nos resultados.

2 Sistema Multiagente Desenvolvido

Neste capítulo será apresentada uma descrição completa do sistema multiagente, acompanhada de vários diagramas **AUML** que visam explicar a estrutura e funcionamento do sistema multiagente desenvolvido.

2.1 Domínio do sistema

Para começar, desenhamos um modelo de domínio do sistema multiagente que dá uma visão abstrata das entidades constituintes do sistema e das relações que existem entre estas entidades, no contexto do problema a resolver. O seguinte modelo apresenta entidades comuns em sistemas reais de tráfego aéreo e que irão fazer parte do nosso sistema.

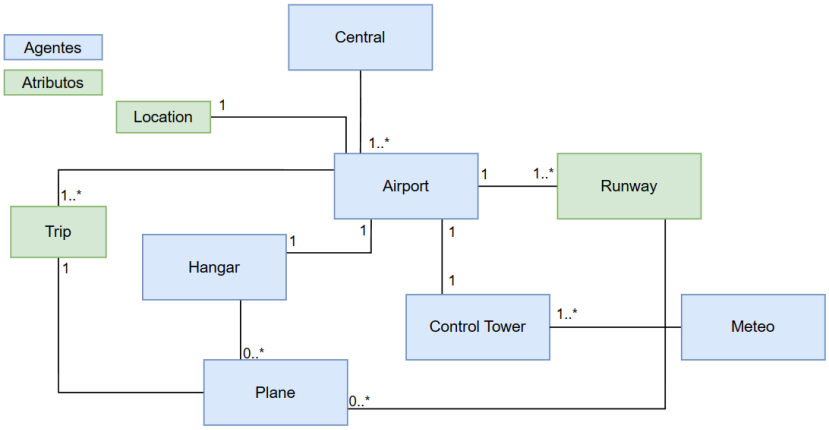


Figura 2.1: Modelo de domínio

Fazendo uma breve descrição do modelo apresentado, irá haver uma **central** que estará conectada com os vários **aeroportos** integrantes do sistema. Os aeroportos estarão localizados numa certa localização e cada um destes aeroportos terá a sua **torre de controlo**, o seu **hangar** e as suas pistas de aterragem disponíveis. Os hangares irão armazenar os **aviões** que se encontram num determinado aeroporto e a Torre de Controlo estará conectada a uma **estação de meteorologia** central que irá indicar as condições meteorológicas nas pistas do aeroporto. E por fim, irão chegar pedidos de viagens aos aeroportos que irão fazer os aviões se deslocarem entre aeroportos.

Através da análise deste modelo de domínio, foi decidido que para atender os objetivos do nosso sistema, as entidades apresentadas a azul irão se traduzir em agentes do sistema multiagente.

2.2 Definição e Descrição dos Agentes

O sistema multiagente proposto é composto por seis tipos de agentes: **Central**, **Meteo**, **Aeroporto**, **Hangar**, **Torre de Controlo** e **Avião**.

- **Central** - é responsável por gerar periodicamente um conjunto de viagens (*Trips*) que são atribuídas aos aeroportos, podendo estes números ser configurados. Para além disso, também é responsável por tentar equilibrar a ocupação dos hangares consoante as mensagens de status

do espaço disponível, enviadas pelos hangares, através da geração de viagens para equilíbrio da ocupação dos hangares.

- **Meteo** - é responsável por enviar as informações meteorológicas para cada Torre de Controlo acerca do estado do tempo na sua localização. Pode funcionar em diferentes modos, consoante a configuração fornecida: o modo "*manual*", em que é permitido ao utilizador, mudar o estado do tempo em cada aeroporto; o modo "*current*" em que é utilizada a API da OpenWeather para obter o estado do tempo em tempo real naquela localização; e o modo "*past*" em que também é utilizada a API da OpenWeather, mas utiliza uma timestamp fornecida por ficheiro de configuração em que são simuladas as condições meteorológicas registadas a partir dessa timestamp.

- **Aeroporto (Airport)** - existe um agente Airport por cada localização e é responsável por receber pedidos de voos (*flights*) e encaminhá-los para o respetivo hangar, para que o avião necessário ao voo seja reservado.

- **Hangar** - o agente Hangar está associado a um aeroporto de uma determinada localização e é responsável pela gestão do número de aviões presentes no hangar e reservar aviões para um *flight*.

- **Torre de Controlo (CT)** - o agente Control Tower está associado a um aeroporto de uma determinada localização e é o agente responsável pela gestão de descolagens e aterragens, tendo que ter em conta diversos fatores como as condições meteorológicas, pistas disponíveis e ocupação do hangar do seu aeroporto.

- **Avião (Plane)** - o agente Plane é a entidade necessária para realizar um *flight* e que irá ser gerido pelos outros agentes para a concretização dessas viagens. Este agente pode estar em três estados: a voar (*Flying*), aterrado (*Landed*) ou à espera de permissão para aterrar (*Waiting_Landing_Perm*).

Tendo em conta a descrição prévia de cada agente e das suas tarefas, construímos o seguinte diagrama de classes que mostra a implementação das classes-agente e as suas relações, fornecendo uma visão clara da arquitetura do sistema multiagente.

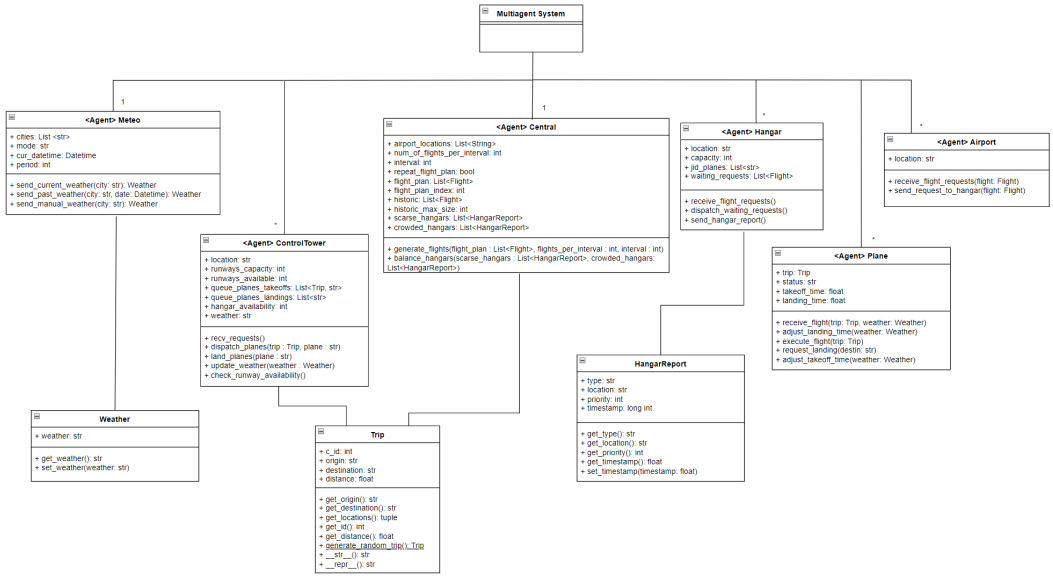


Figura 2.2: Diagrama de Classes

2.3 Interações entre os Agentes

No sistema multiagente proposto, os agentes interagem de maneira dinâmica para realizar as operações necessárias para o funcionamento do sistema de gestão aeroportuária.

A comunicação entre os agentes é baseada numa lógica de ambiente cooperativo, onde o foco está no aumento da utilidade global do sistema em vez da utilidade individual de cada agente. Isso implica que os agentes devem trabalhar em conjunto para maximizar o desempenho global, coordenando as suas ações e partilhando informações relevantes.

A comunicação entre os agentes foi definida utilizando a linguagem de comunicação de agentes (ACL) da FIPA (*Foundation for Intelligent Physical Agents*), que estabelece padrões e protocolos para interações entre agentes, possibilitando a troca de mensagens de maneira estruturada e eficiente, tendo sido seguidas todas as normas da FIPA-ACL.

Nesta secção, exploraremos em detalhe as interações entre os seis tipos de agentes, que permitem a realização das funcionalidades do sistema multiagente.

2.3.1 Descolagem

1. A **central** envia um flight ao **aeroporto** cujo flight tem origem (performative: *request*, body: *Trip*).
2. O **aeroporto** envia um pedido de avião ao **hangar** (performative: *request*, body: *Trip*). O hangar adiciona a Trip à queue de pedidos em espera e posteriormente, verifica a possibilidade de reserva de um avião no seu hangar.
3. O **hangar** envia o jid do avião selecionado à **CT** e a respetiva Trip a realizar (performative: *inform*, body: *{plane_jid: String, trip: Trip}*).
4. A CT ao receber a mensagem do hangar, adiciona o pedido à queue de pedidos de descolagem. Após ser verificado que existem condições para a descolagem (runways disponíveis e estado do tempo favorável), a **CT** reserva uma runway e envia mensagem para descolar ao **Plane**, indicando o estado do tempo ao avião, o que irá afetar na rapidez da descolagem (performative: *inform*, body: *{trip: Trip, weather: Weather}*).
5. Após concluir a descolagem, o **Plane** envia mensagem de confirmação de término da descolagem à **CT** (performative: *confirm*, body: *"plane_jid"*) e a CT liberta a *runway*.

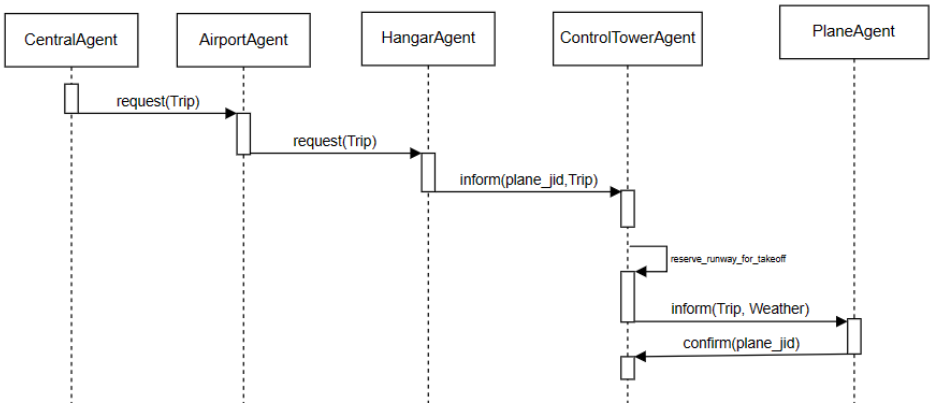


Figura 2.3: Diagrama de sequência - Descolagem

2.3.2 Voo

A lógica de voo simula um tempo de viagem proporcional à distância entre as localizações dos aeroportos origem e destino. Esta distância é obtida através da biblioteca *GeoPy* do Python que facilita o acesso a API's de geocodificação e que nos permite obter a informação relativa à distância entre duas localizações.

Para manter o sistema a fornecer resultados em tempo útil, aplicamos um fator de conversão que faz com que um quilómetro seja realizado em 0.015 segundos. Para dar uma ideia prática do efeito desse fator, uma distância em linha reta de Lisboa ao Porto é de 275 Km, o que se traduz num voo de 4,125 segundos.

2.3.3 Aterragem

1. O **Plane** após ter passado o tempo de voo, chega ao destino e envia mensagem de pedido de aterragem à **CT**(performative: *request*, body: `"plane_jid"`).
2. A **CT** adiciona um pedido de aterragem na queue de aterragens e posteriormente verifica se as condições (runways disponíveis, Lugar disponível no hangar e estado do tempo favorável) permitem aterrar e quando for possível, é enviada uma mensagem ao **Plane** com o estado do tempo que irá afetar a rapidez da aterragem (performative: *confirm*, body: *Weather*).
3. Após concluir a aterragem, o **Plane** envia mensagem de aterragem à **CT**, que liberta a runway, e ao **Hangar**, que adiciona o seu jid à lista de aviões presentes. (performative: *inform*, body: `"plane_jid"`)

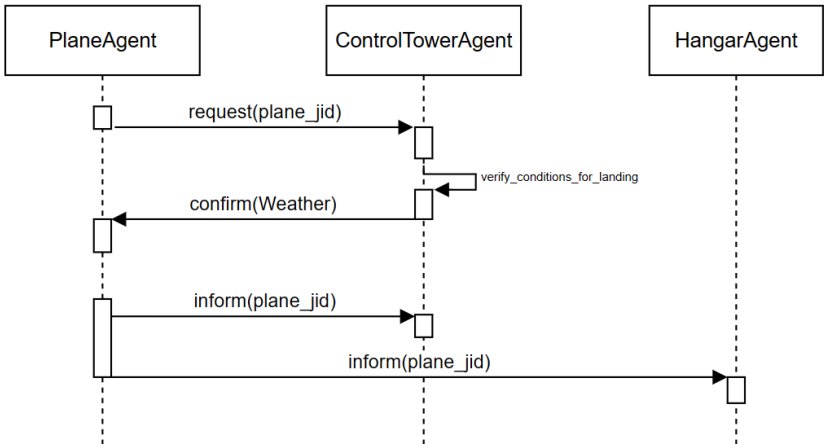


Figura 2.4: Diagrama de sequência - Aterragem

2.3.4 Balanceamento de Hangares

Esta funcionalidade que permite corrigir desbalanceamento no número de aviões em certos hangares, tentando sempre transferir aviões de hangares cheios, para hangares que estejam mais vazios e vice-versa.

1. Ao receber e ao enviar aviões, o **hangar** irá verificar o seu estado de ocupação. Caso o espaço livre seja menor que 3, irá mandar uma mensagem indicativa de que o seu hangar está cheio. Caso a sua ocupação seja menor que 3, irá mandar uma mensagem indicativa de que o seu hangar está escasso. (performative: *inform*, body: `HangarReport`).
2. A **Central** ao receber as mensagens, coloca-as numa queue de hangares cheios ou vazios, dependendo da mensagem. Posteriormente o agente faz a verificação das queues de hangares cheios

e vazios e analisa se pode gerar uma combinação de viagem que permita levar um avião de um hangar cheio para um hangar vazio e assim balancear as ocupações dos hangares (performative: *request*, body: `Trip`).

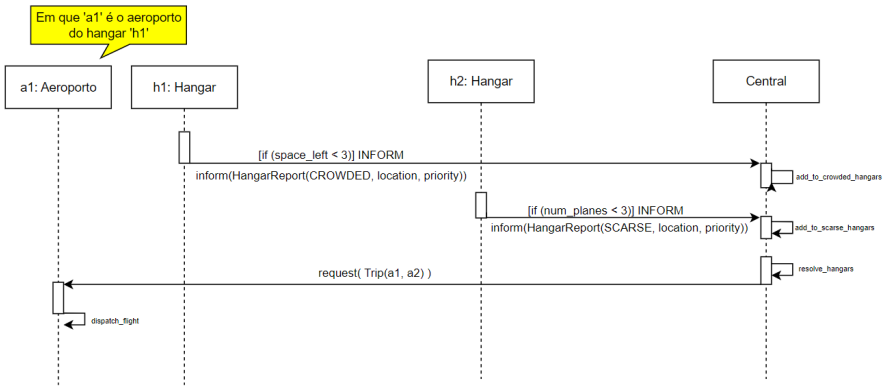


Figura 2.5: Diagrama de sequência - Balanceamento de Hangares

O sistema para além de gerar viagens quando há pelo menos dois aeroportos, um sobrelotado e outro vazio, também é capaz de lidar com casos em que todos os hangares problemáticos têm apenas problema de sobrecarga ou de desocupação. Isso acontece quando existe algum desses hangares que está sem resolução há mais de 20 segundos, fazendo com que a Central verifique os aeroportos disponíveis, ou seja, aeroportos que não estejam sinalizados, e verifica o que está mais próximo. Assim gera a viagem de balanceamento ótima e garante uma melhor distribuição dos aviões pelos hangares.

2.3.5 Informação Meteorológica

Relativamente às informações meteorológicas, as **Control Towers** recebem a informação pelo agente **Meteo**. O agente Meteo pode fornecer diferentes tipos de informação meteorológica: *current*, *past* ou *manual*. Se for *current*, o agente Meteo irá mandar para todas as Control Towers, o estado de tempo atual na cidade onde o seu aeroporto se situa. Se for no modo *past* tem de ser indicada a timestamp da altura que se pretende obter os dados meteorológicos. Além disso, cada vez que enviar uma mensagem com o estado do tempo, a próxima mensagem será respetiva à hora seguinte, ou seja, as mensagens subsequentes fornecerão informações meteorológicas para horas consecutivas após o timestamp fornecido. Se for no modo *manual*, o estado do tempo é alterado manualmente pelo utilizador, e o agente Meteo envia o estado de tempo, resultante das alterações manuais, à Torre de Controlo alterada.

Em todos estes modos, o agente Meteo envia a informação meteorológica a cada torre de controlo com o seu estado do tempo periodicamente com a seguinte estrutura de mensagem: (performative: *inform*, body: `Weather`).

As informações meteorológicas irão influenciar o funcionamento das Control Towers, mais concretamente, irão afetar as aterragens e descolagens. Os possíveis estados de tempo são: *Clear*, *Clouds*, *Smoke*, *Mist*, *Haze*, *Dust*, *Drizzle*, *Fog*, *Sand*, *Rain*, *Ash*, *Squalls*, *Snow*, *Volcanic Ash*, *Tornado*, *Thunderstorm*. De todos estes estados, os últimos três implicam a impossibilidade de realizar aterragens ou descolagens, enquanto que os restantes apenas fazem com que as aterragens e descolagens demorem um pouco mais, sendo o intervalo de tempo padrão multiplicado por um fator que varia consoante o estado de tempo específico. Aqui apresentamos os fatores para cada estado temporal:

- Clear: 1.0
- Clouds: 1.3

- Smoke: 1.3
- Mist: 1.4
- Haze: 1.4
- Dust: 1.5
- Drizzle: 1.6
- Fog: 1.8
- Sand: 1.8
- Rain: 2.0
- Ash: 2.2
- Squalls: 2.3
- Snow: 2.5

2.3.6 Diagrama de Atividades

De forma a resumir todas as interações previamente explicadas, é possível representá-las num diagrama de atividades que fornece uma visão geral das atividades do sistema e das interações entre os diferentes agentes.

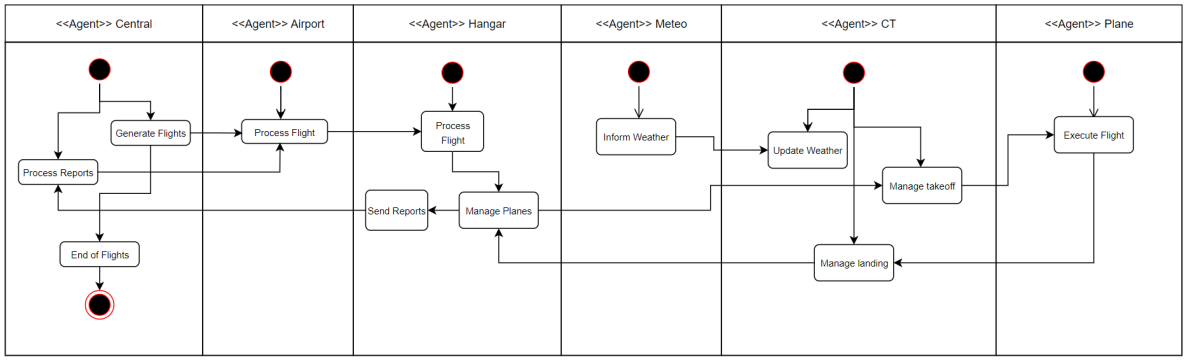


Figura 2.6: Diagrama de Atividades

2.4 Outros detalhes do sistema

Nesta secção, apresentam-se detalhes fora do contexto direto de sistemas multiagente, mas ainda assim, importantes ao funcionamento da aplicação.

2.4.1 Ficheiro de configuração

Para executar o programa, é necessário um ficheiro de configuração que instancia certos parâmetros que o sistema precisa. Existem vários parâmetros que o sistema é capaz de configurar dinamicamente. Estes parâmetros podem ser divididos em três diferentes tipos: *airports*, *flights* e *weather*.

De seguida apresenta-se uma lista dos parâmetros configuráveis e um exemplo de um ficheiro *json* com todas as possíveis parametrizações.

- **Airports**:
  - As localizações dos aeroportos que irão fazer parte do sistema de tráfego aéreo;
  - O número de aviões, capacidade dos hangares e pistas.
- **Flights**:
  - `interval`: O intervalo de tempo entre o envio de voos;
  - `num_of_fligths_per_interval`: O número de voos gerados por intervalo de tempo;

  - `plan`: O plano de voo que a central irá pôr em prática no sistema, ou seja, quais os voos que irão ser simulados. O plano pode ser definido manualmente, ou pode ser aleatório, caso indique a string "random". Definindo manualmente, apenas tem de indicar a origem e destino dos voos, podendo especificar o número de repetições do voo com o atributo `reps`;
  - `repeat`: Caso seja especificado um plano concreto, é possível indicar que o plano se repita quando a lista de voos acabar.
  - `count`: Caso não seja especificado um plano concreto, é possível indicar o número de voos aleatórios que se pretende gerar.
- **Weather**:
  - `mode`: O modo de obtenção das condições meteorológicas: *current*, *past* ou *manual*;
  - `from`: Caso seja especificado o modo *past*, é necessário especificar a data passada, cujas condições meteorológicas irão ser replicadas no sistema (não é possível indicar uma data anterior a um ano atrás);
  - `period`: É possível especificar a frequência com que o agente Meteo envia as informações de meteorologia às Control Towers.

```
{
    "airports": {
        "Lisboa": {"num_planes": 2, "hangar_capacity": 5, "runways": 4},
        "Porto": {"num_planes": 4, "hangar_capacity": 5, "runways": 3},
        "Faro": {"num_planes": 3, "hangar_capacity": 5, "runways": 2}
    },
    "flights": {
        "interval": 10, // intervalo de tempo entre voos gerados (opcional. default: 10)
        "num_of_flights_per_interval": 1, // número de voos gerados por intervalo (opcional. default: 1)
        "plan": [
            {
                "origin": "Lisboa",
                "destination": "Porto",
                "reps": 2
            },
            {
                "origin": "Faro",
                "destination": "Lisboa",
                "reps": 4
            }
        ],
        "repeat": false // se true, repete o plano de voos infinitamente (opcional. default: false)
    },
    "weather": {
        "mode": "past", // "manual" ou "current" ou "past"
        "from": "2024-01-04 20:00:00", // a data só pode ir no máximo até um ano atrás
        "period": 15 // em segundos (opcional. default: 30)
    }
}
```

2.4.2 Interface

De maneira a visualizar o estado do sistema ao longo da sua execução, o grupo decidiu criar uma interface gráfica que consiga apresentar numa só tela, toda a informação relativa aos diferentes agentes presentes no sistema. Para tal, recorremos à biblioteca **Tkinter** do Python que fornece ferramentas para criar GUI's (Graphical User Interface) para aplicações Python.

Nesta interface, são apresentadas as informações principais dos agentes ***Central***, ***Airport's***, ***Hangar's***, ***Control Tower's*** and ***Plane's***. Para além disso também é apresentado ao lado os *logs* do sistema relativos aos eventos ocorridos em que os logs mais recentes aparecem em cima.

A seguinte imagem, demonstra a interface num sistema com 3 aeroportos diferentes e com 8 aviões.

Figura 2.7: Interface gráfica do programa.

Relativamente à mudança do estado do tempo, a interface demonstrada encontra-se no modo *manual* de obtenção do estado do tempo. Neste modo a interface mostra um botão "`Change Weather`"que permite alternar entre "*Clear*", "*Rain*"e "*Thunderstorm*".

3 Resultados e análise crítica

Nesta secção apresentaremos alguns cenários da execução do sistema, que abrangem os diferentes eventos que podem ocorrer ao longo da execução do sistema, com a intenção de dar uma melhor visão das interações entre os agentes e da maneira que eles reagem conforme as diversas situações.

3.1 Descolagem e Aterragem sem conflitos

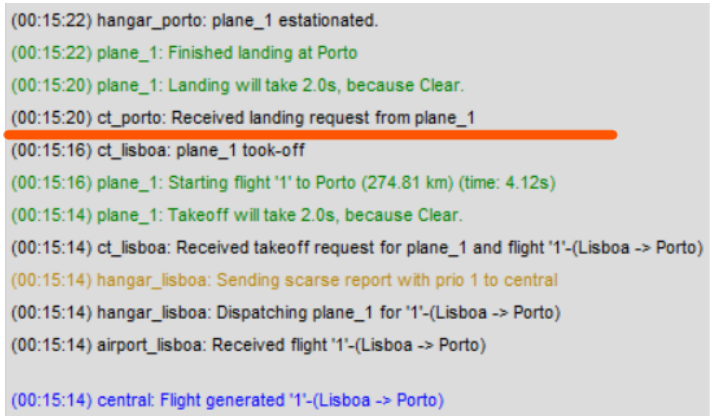


Figura 3.1: Cenário de Descolagem e Aterragem

A linha laranja divide entre a parte inicial do processo que leva à descolagem do avião (parte inferior) e a parte de aterragem do avião (parte superior). Aqui podemos ver o processo de passagem das mensagens pelo aeroporto, hangar, control tower e finalmente ao plane designado pelo hangar. É também possível verificar o impacto que o tempo tem no tempo de descolagem e aterragem dos aviões e também a distância e tempo que a viagem vai demorar. Pelas timestamps dos logs, verificamos que o avião demorou os 4 segundos previstos.

3.2 Impossibilidade de aterragem - Hangar cheio

Este cenário demonstra a impossibilidade de aterrar quando o hangar está cheio.

Figura 3.2: Cenário de Hangar Cheio

Aqui é possível verificar que o hangar destino (Porto) encontra-se cheio, logo a Control Tower não permitiu a aterragem, ao Plane_1, não tendo respondido para permitir aterrar, fazendo com que o avião fique no estado "Waiting for landing perm".

3.3 Impossibilidade de aterragem - Runways ocupadas

Para simular casos em que as runways de um aeroporto estão todas ocupadas, configurou-se um cenário em que se geram várias viagens para o aeroporto do Porto e este aeroporto apenas tem uma runway. Neste cenário os aviões demoram por padrão 3 segundos a aterrar, e com a configuração do tempo com "Rain"o tempo para aterragem duplica. Ou seja se num espaço de 6 segundos tentarem aterrar 2 aviões, o segundo terá de esperar.

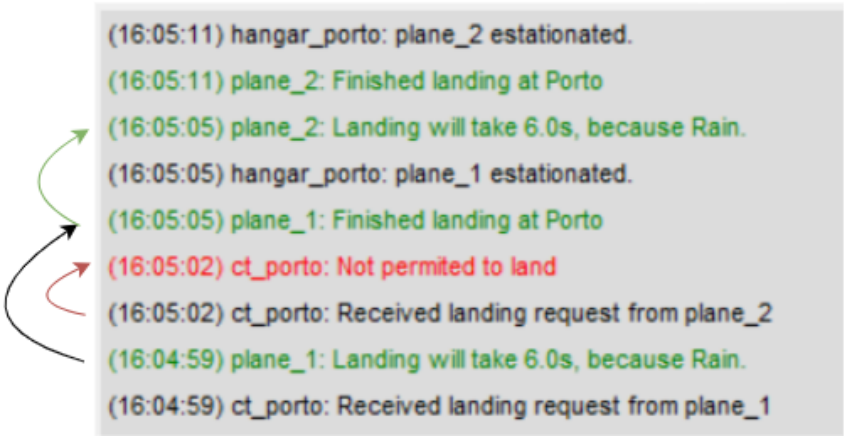


Figura 3.3: Cenário de Runways ocupadas

Como podemos ver, ao longo da seta preta, está a decorrer a aterragem do primeiro avião, e entretanto, no começo da seta vermelha, aparece um novo avião que pede para aterrar, mas é lhe negada a permissão. Quando o primeiro avião acaba de aterrar, representado no início da seta verde,

podemos ver que despoleta a permissão para o segundo avião poder aterrar, que está indicado no final da seta verde.

3.4 Funcionalidade *Past Weather*

Neste cenário é possível demonstrar várias funcionalidades ao mesmo tempo. Aproveitando um exemplo real de meteorologia dos dias 16 de março de 2024 em que ocorreram grandes cheias no Dubai devido às tempestades, configuramos um sistema que apresenta aeroportos à volta do Dubai e definimos que a configuração do estado do tempo fosse no modo *past*, reproduzindo as condições meteorológicas dessa altura.

Como o estado do tempo nesta altura no Dubai foi de *Thunderstorm*, o agente Meteo irá informar as Control Towers e consequentemente não irão permitir a descolagem nem aterragem de aviões.

Figura 3.4: Cenário com Funcionalidade *Past Weather* e Impossibilidade de Aterragem

Como podemos ver no inicio em baixo, o estado do tempo registado em 16 de abril de 2024 às 12h em Dubai e Sharjah é de *Thunderstorm* e o agente Meteo define esse tempo para os aeroportos dessas cidades. Com *Thunderstorm*, as Control Towers não permitem a descolagem ou aterragem de aviões.

Ao serem gerados voos a partir de Dubai, podemos ver, no decorrer da primeira seta, que o pedido de voo, só chega até à Control Tower e não passa daí pois a CT proíbe a descolagem de voos em muito más condições meteorológicas. O mesmo acontece no segundo pedido de voo, indicado na segunda seta.

Posteriormente quando o agente Meteo envia as novas informações do estado do tempo, indicadas no retângulo verde, estas condições já permitem a descolagem de aviões, e despoleta o envio da ordem de descolagem aos aviões que ficaram retidos em espera para descolar.

3.5 Balanceamento de Hangares

Neste cenário, queremos demonstrar o processo de balancear hangares.

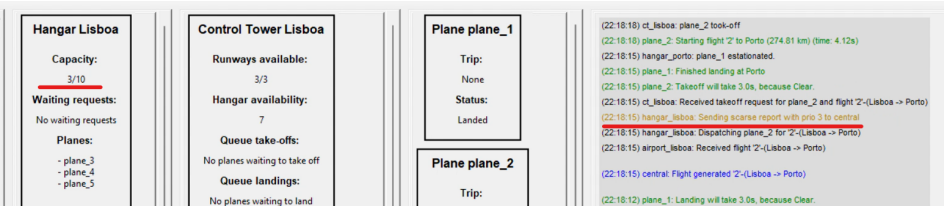


Figura 3.5: Envio de Hangar Scarse message

Aqui podemos ver que o número de aviões no hangar Lisboa passa a 3 com a descolagem de um avião do seu aeroporto para realizar o *flight* número 2, estando o número de hangares e o log de envio de mensagem sublinhados na imagem.

De seguida, o aeroporto do Porto ao receber o *plane_2*, passa a estar com apenas 3 lugares disponíveis no hangar, despoletando o envio de uma mensagem de *CrowdedHangar*.

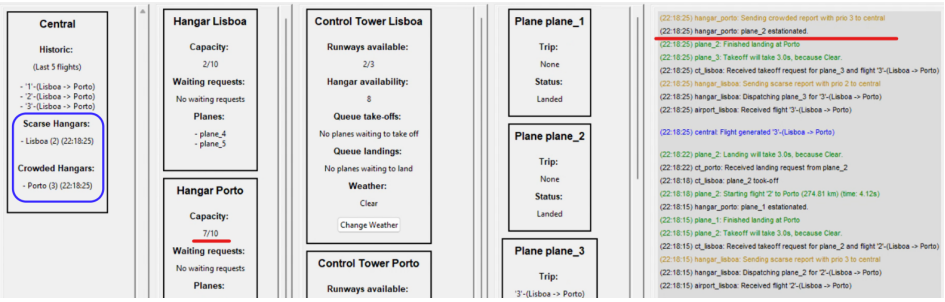


Figura 3.6: Envio de Hangar Crowded message

Como se pode ver, a Central passa a saber de dois hangares que se encontram desbalanceados, e desta forma a Central gera a seguinte viagem de balanceamento de hangares para resolver o problema.

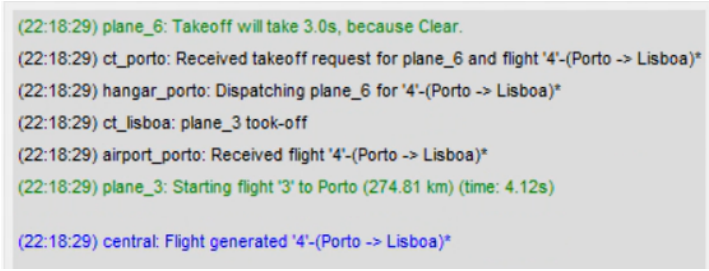
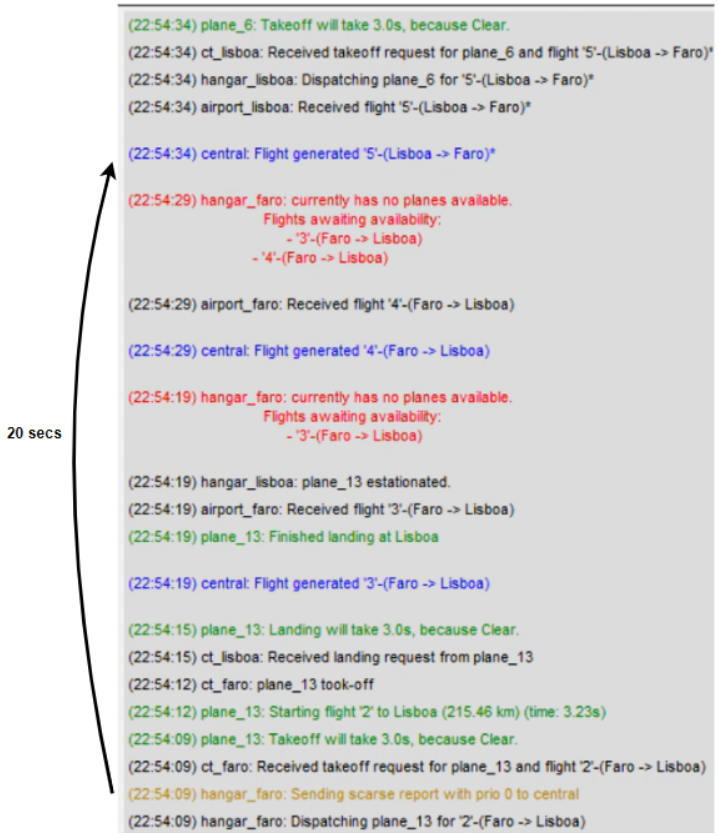


Figura 3.7: Geração de mensagem de balanceamento

Para distinguir as viagens de balanceamento e as viagens normais, um Flight resultante de uma ação de balanceamento apresenta um asterisco no final, como se pode ver no log.

Relativamente à funcionalidade de balancear um hangar após 20 segundos sem haver nenhum hangar com o problema oposto, demonstra-se um cenário de um hangar que envia uma mensagem de ScarseHangar à Central, sem haver nenhum outro Hangar que esteja cheio e sirva de fonte de aviões ao hangar vazio.

(a) Estado da informação dos hangares na Central



(b) Envio da mensagem de balanceamento

Figura 3.8: Cenário de balanceamento isolado

Como podemos ver na imagem da esquerda apenas existe o Scarse Hangar de Faro e na imagem da direita vemos que durante os 20 segundos não é enviado nenhum pedido de Crowded Hangar. Passando

os 20 segundos, a Central verifica os possíveis aeroportos que pode ir buscar aviões, mesmo apesar de não estarem cheios, e vê qual é o mais próximo de Faro, que neste caso com Porto, Lisboa e Faro, o de Lisboa é o mais próximo, gerando um voo de balanceamento no sentido Lisboa-Faro.

3.6 Estatísticas de voo

Ao longo do decorrer do programa, quando um voo é terminado, o avião gera um pequeno relatório que indica os tempos de espera que teve de esperar para aterrar e descolar. Os aviões escrevem este relatório num ficheiro *json* que tem uma lista de objetos com a informação de cada voo com a seguinte estrutura.

```
[
    {
        "id": 1,
        "flight": "'1'-(Lisboa -> Porto)",
        "waiting_takeoff": 2.51,
        "waiting_landing": 0.0
    },
    {
        "id": 2,
        "flight": "'2'-(Lisboa -> Porto)",
        "waiting_takeoff": 0.01,
        "waiting_landing": 5.32
    },
    ...
]
```

Para sumariar o resultado destas estatísticas, no final da execução do programa é gerada um ficheiro que apresenta estatísticas desses tempos medidos nos voos em segundos, como a média, mediana, valores mínimos e máximos.

Aproveitando o exemplo previamente mostrado configurado no Dubai na altura das recentes e mediáticas cheias, podemos ver que ocorreram vários atrasos nos voos devido ao encerramento das pistas pelas Control Towers, obtendo as seguintes estatísticas.

```
"total_flights": 10,
"average": {
    "waiting_takeoff": 9.18,
    "waiting_landing": 10.43
},
"median": {
    "waiting_takeoff": 3.09,
    "waiting_landing": 0.48
},
"min": {
    "waiting_takeoff": 0.01,
    "waiting_landing": 0.0
},
"max": {
    "waiting_takeoff": 28.84,
    "waiting_landing": 29.96
}
```

Através destas estatísticas podemos constatar que o efeito das tempestades teve bastante impacto nos tempos de espera dos aviões, dado que foram impedidas aterragens e descolagens com o estado do tempo a *Thunderstorm*.

Para avaliar as melhorias trazidas com o mecanismo de balanceamento de hangares também podemos aproveitar esta funcionalidade de apresentação de estatísticas e tirar melhores conclusões. Partindo da seguinte configuração, testamos uma versão sem o mecanismo de balanceamento e com o mecanismo de balanceamento. É uma configuração relativamente simples, mas com um período criação de voos de 2 segundos, o que se traduz numa frequência bastante elevada para o número de aviões disponíveis:

```
{
    "airports": {
        "Lisboa": { "num_planes": 6, "hangar_capacity": 8, "runways": 5 }
            ,
        "Porto": { "num_planes": 6, "hangar_capacity": 8, "runways": 5 },
        "Faro": { "num_planes": 6, "hangar_capacity": 8, "runways": 5 }
    },
    "flights": {
        "plan": "random",
        "interval": 1,
        "count": 20
    },
    "weather": {
        "mode": "manual"
    }
}
```

Estes são os resultados da versão **sem balanceamento**:

```
"total_flights": 14,
"average": {
    "waiting_takeoff": 1.23,
    "waiting_landing": 2.25
},
"median": {
    "waiting_takeoff": 0.02,
    "waiting_landing": 0.01
},
"min": {
    "waiting_takeoff": 0.01,
    "waiting_landing": 0.0
},
"max": {
    "waiting_takeoff": 15.89,
    "waiting_landing": 16.81
}
```

E estes são os resultados da versão **com o mecanismo de balanceamento** ativo:

```
"total_flights": 26,
"average": {
    "waiting_takeoff": 0.49,
    "waiting_landing": 0.33
},
```

```
"median": {
    "waiting_takeoff": 0.03,
    "waiting_landing": 0.01
},
"min": {
    "waiting_takeoff": 0.01,
    "waiting_landing": 0.0
},
"max": {
    "waiting_takeoff": 2.83,
    "waiting_landing": 2.81
}
```


Em ambos os casos, a Central gerou 20 voos normais, mas é notório que a versão sem balanceamento, só conseguiu cumprir 14 voos pois chegaram pedidos de voo que os aeroportos não eram capazes de atender devido à falta de aviões, o que não é tratado na versão sem balanceamento. Na versão com balanceamento são apresentados 26 voos, 6 a mais do que os 20, porque foram gerados 6 voos de balanceamento, que resultaram na execução completa do plano de voos atribuído ao sistema e mais eficiente.

4 Trabalho Futuro

No âmbito do desenvolvimento futuro de um sistema multiagentes, haveria várias direções promissoras a considerar:

- **Configuração Mais Completa**: Proporcionar uma interface que permitisse ao utilizador definir os parâmetros do sistema de forma mais intuitiva. Ao invés de depender de um ficheiro JSON para configurar o sistema, o utilizador poderia ser guiado através de um processo interativo, onde cada parâmetro era explicado detalhadamente. Outra melhoria para a configuração envolveria a possibilidade de definir dinamicamente vários outros parâmetros relativos ao sistema, como por exemplo: o tempo de aterragem e descolagem dos aviões, os fatores de atraso nas aterragens e descolagens relativos aos diferentes estados do tempo, condições da ativação dos alertas de hangares vazio ou cheio, definição da velocidade de aviões, etc.
- **Consideração do Número de Passageiros**: Incluir informações mais detalhadas nos voos, como por exemplo, o número de passageiros em cada viagem, para que isso pudesse influenciar a prioridade de aterragem ou descolagem, especialmente em situações de sobrelotação das pistas. Esta inclusão tornaria o sistema mais adaptável às condições reais do tráfego aéreo.
- **Algoritmo de Balanceamento de Hangares Aprimorado**: Refinar o algoritmo de balanceamento dos hangares para abranger uma gama mais ampla de situações de desequilíbrio. Atualmente, o sistema gera viagens a partir de um conjunto restrito de condições pré-definidas. Talvez pudéssemos aprimorar o sistema de balanceamento para procurar resolver certos casos de desbalanceamento de forma mais otimizada, com um algoritmo que verificasse de forma mais intensiva as diferentes possibilidades de balanceamento e com base em critérios como, distância da viagem e estado futuro dos hangares envolvidos.
- **Sugestões de Aterragens Alternativas**: Quando um certo aeroporto não se encontra em condições para permitir aterragens, a sua torre de controlo poderia responder com uma sugestão de aeroporto de alternativa mais próximo para aterragem aos aviões que tenham pedido para aterrar. Após haver um desbloqueio do aeroporto que o avião pretendia aterrar, a torre de controlo comunicaria com o avião, a possibilidade de voltar a tentar.
- **Reinforcement Learning**: Uma outra ideia mais complexa de implementar, seria integrar técnicas de *reinforcement learning*, permitindo que os agentes aprendessem uma política de gestão dos aviões autonomamente, com base em dados reais de tráfego aéreo real, que também poderiam ser uma funcionalidade em que se buscaria voos de uma determinada altura, de forma similar ao que implementamos com a simulação de condições meteorológicas no passado.

5 Conclusões

Em suma, a realização deste trabalho prático permitiu consolidar os conceitos abordados na unidade curricular de Agentes e Sistemas Multiagente, passando pelos conceitos de agente, *behaviors*, normas FIPA e Agent UML.

Fazendo uma análise critica ao nosso trabalho, consideramos que desenvolvemos uma arquitetura interessante que potencializa a colaboração entre os diversos agentes inseridos num contexto de gestão de tráfego aéreo. Destacamos a implementação de funcionalidades interessantes e relativamente complexas como a interface gráfica, as várias opções de parametrização do sistema com diferentes modos de atuação para os vários agentes e a integração do sistema com sensorização virtual, através da utilização de API's que fornecem serviços de geolocalização para permitir ter uma noção mais aproximada à realidade das localidades e também de serviços de meteorologia para simular condições atmosféricas da maneira que o utilizador assim entender. No entanto, o trabalho ainda tem muito espaço para melhoria, havendo diversos aspetos nos quais podia evoluir, tais como os que foram apontados no capítulo Trabalho Futuro.