



UNIVERSIDADE DO MINHO

MESTRADO EM ENGENHARIA INFORMÁTICA  
AGENTES E SISTEMAS MULTIAGENTE

---

**Sistema multiagente para a gestão de um aeroporto**

---



Henrique Parola  
pg50415



Ana Henriques  
pg50196



# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>3</b>
1.1	Domínio do projeto . . . . .	3
1.2	Motivação e objetivos . . . . .	3
1.3	Estratégia de desenvolvimento . . . . .	4
<b>2</b>	<b>Sistema multiagente desenvolvido</b>	<b>4</b>
2.1	Características e tipos de agentes . . . . .	4
2.2	Mensagens protocolares . . . . .	8
2.3	Arquitetura baseada em agentes . . . . .	9
<b>3</b>	<b>Resultados Obtidos</b>	<b>13</b>
<b>4</b>	<b>Funcionalidades Extra</b>	<b>19</b>
<b>5</b>	<b>Análise Crítica e Trabalho Futuro</b>	<b>20</b>



## Listas de Figuras

1	Diagrama de Classes e de Pacotes . . . . .	9
2	Máquina de Estados do Avião . . . . .	10
3	Diagrama de Atividades . . . . .	11
4	Diagramas de Sequência . . . . .	12
5	Teste 1 - Instante inicial . . . . .	13
6	Teste 1 - Primeiros <i>requests</i> . . . . .	14
7	Teste 1 - Compra de vagas . . . . .	14
8	Teste 1 - Fim da simulação . . . . .	15
9	Teste 2 - Instante inicial . . . . .	16
10	Teste 2 - primeiros <i>requests</i> . . . . .	16
11	Teste 2 - Compra de vagas . . . . .	17
12	Teste 2 - Processamento dos <i>requests</i> pendentes . . . . .	17
13	Teste 2 - Fim da simulação . . . . .	18
14	Teste 2 - Tempo de espera no ar . . . . .	18
15	Tela de configuração . . . . .	19



# 1 Introdução

## 1.1 Domínio do projeto

O projeto em questão envolve a concepção e desenvolvimento de uma arquitetura distribuída para gerir uma infraestrutura aeroportuária. Isso implica a criação de uma estrutura que permita a comunicação e interação eficiente entre os agentes responsáveis pela gestão das partidas e chegadas de aviões. Este sistema multiagente terá, então, a capacidade de coordenar e monitorizar as atividades de diferentes agentes envolvidos, como os aviões, a torre de controlo, as companhias aéreas e outros atores relevantes para este contexto. Através de uma arquitetura baseada em agentes, procura-se otimizar processos cruciais na gestão do tráfego aéreo e proporcionar uma melhor experiência aos passageiros.

## 1.2 Motivação e objetivos

A gestão eficiente das entradas e saídas de aviões em aeroportos é um desafio complexo, que requer um planeamento cuidadoso e a coordenação por parte de uma entidade central. Com um sistema multiagente, é possível otimizar o fluxo de tráfego aéreo ao considerar fatores como a disponibilidade das pistas e a capacidade das gares, resultando numa redução de congestionamentos. Além disso, com a coleção de informações em tempo real, nomeadamente os estados dos aviões e a disponibilidade dos recursos, o sistema pode-se adaptar rapidamente a situações de emergência ou mudanças repentinhas, como cancelamentos de voos ou a crescente demanda de serviços. A tomada de decisão inteligente contribui para uma melhor gestão das operações aeroportuárias, melhorando a capacidade de resposta do aeroporto face a imprevistos.

O objetivo do projeto reside, consequentemente, no *design* e na modelação de um sistema distribuído baseado em agentes para cumprir uma gestão estável e eficiente de um aeroporto. Entre os requisitos do sistema, destacam-se:

- O sistema deve ter uma Torre de Controlo que será responsável pela gestão dos aviões que pretendem aterrissar ou descolar;
- Um avião deve conter no mínimo um (1) ID, (2) companhia, (3) tipo (mercadoria ou comercial), (4) origem e (5) destino;
- A Torre de Controlo deve questionar o Gestor de Gares do aeroporto se existe alguma gare (local de estacionamento/ou de operação) disponível para um dado avião estacionar;
- Cada pista do aeroporto deve ter uma geolocalização associada, na forma de coordenada (X,Y);
- A existência de uma pista livre, juntamente com uma vaga disponível na gare, levará a Torre de Controlo a informar um determinado avião de que é possível aterrissar. Caso contrário terá de informar o avião que terá de aguardar. Além disso, deve haver um limite de aviões em fila de espera para aterrissar no aeroporto;
- O sistema deve ter uma entidade encarregue de apresentar a informação geral sobre os voos em operação. Para este fim, deverá existir um agente que mostre toda a informação sobre aviões que estão para descolar ou aterrissar.



### 1.3 Estratégia de desenvolvimento

O projeto foi concebido com uso da linguagem *Python*, com foco na plataforma *Spade* para a criação de sistemas multiagentes. A ampla variedade de recursos que a linguagem *Python* disponibiliza, combinada com a especialização da plataforma *Spade*, garantem uma implementação flexível, capaz de lidar com a complexidade e a constante evolução inerentes a um ambiente aeroportuário. O sistema foi inicialmente modelado através de diagramas UML, incluindo diagramas de atividades, classes, sequência e máquina de estados, os quais serão apresentados de forma abrangente no capítulo 2.

O primeiro passo empreendido pelo grupo de trabalho consistiu na seleção e definição dos agentes que integrariam o sistema. Numa fase prematura, foram identificados apenas cinco agentes, entre os quais um deles, o *Airline*, foi concebido como um aspecto adicional do grupo. Depois de um aprimoramento mais cuidadoso, foram somados outros cinco agentes, cada um responsável pela *dashboard* dos cinco agentes iniciais. A razão por detrás desta decisão foi possibilitar um *tracking* granular de todas as fases do sistema, para todos os agentes. O segundo passo envolveu o planeamento das várias interações entre os agentes e das atividades que cada um desempenharia em resposta a determinados eventos. Nesse sentido, a construção dos diagramas UML assumiu um papel primordial, fornecendo uma base sólida para o tal planeamento ao delinear e estruturar as comunicações e tarefas atribuídas a cada agente.

A avaliação do funcionamento do sistema assentou-se em dois cenários de teste previamente estabelecido. Estes cenários foram desenvolvidos num ambiente determinístico e controlado, permitindo a observação das interações entre os agentes do sistema. Os resultados desses testes serão discutidos com mais detalhe no capítulo 3. Por forma a observar a evolução do cenário de simulação, o grupo de trabalho propôs a concretização de uma interface gráfica altamente sugestiva do estado do sistema. Esta interface foi construída através da biblioteca CustomTkinter em Python e, para além da visualização do sistema em tempo real, fornece variadas opções de configuração para a simulação do sistema.

## 2 Sistema multiagente desenvolvido

A presente secção abrange os aspetos essenciais do sistema implementado, proporcionando uma visão aprofundada das características, arquitetura e interações dos agentes envolvidos. Compreender as particularidades desses elementos é fundamental para compreender o próprio funcionamento do sistema multiagente.

### 2.1 Características e tipos de agentes

#### Airplane

O agente *Airplane* representa um único **Avião** que pretende aterrizar ou levantar voo neste aeroporto. Para tal, o mesmo precisa primeiramente da autorização da *ControlTower* que, na ausência de espaços disponíveis ou na existência de aviões da sua companhia já em espera, impossibilitará a ação requerida. Cada avião é caracterizado por (1) um ID, (2) uma companhia aérea, (3) um tipo de transporte, (4) um estado, (5) uma origem, (6) um



destino, (7) uma data, (8) um nível de prioridade e (9) uma gare na qual está estacionado. Os *behaviours* associados a este agente são:

- **InformDashboardInitState:** Avisa o agente *DashboardAirplane* acerca da criação do avião. Para isso, indica o seu ID, o seu estado, o ID da sua companhia aérea e o tipo de transporte administrado pelo avião.
- **WantsToLand:** Envia à *ControlTower* a solicitação para aterrissar no aeroporto, fornecendo simultaneamente os seus dados para posterior processamento. O *Airplane* notifica, também, a *DashboardAirplane* para apresentar os dados desse avião.
- **WantsToTakeOff:** Envia à *ControlTower* a solicitação para descolar no aeroporto, fornecendo simultaneamente os seus dados para posterior processamento. O *Airplane* notifica, também, a *DashboardAirplane* para apresentar os dados desse avião.
- **ReceiveBehaviour:** Recebe respostas da *ControlTower* sobre o seu pedido, alertando sempre a *DashboardAirplane* sobre o novo estado do avião:
  - Se o avião pretende aterrissar e a lista de aviões em espera para aterrissar estiver cheia, então a resposta é para se deslocar para outro aeroporto.
  - Se não existir espaços disponíveis para a concretização da ação (em termos de pistas ou gares livres) a resposta é para aguardar, modificando o estado para *waiting\_land* ou *waiting\_takeoff*.
  - Se existirem os recursos necessários, a ação é autorizada. Consequentemente, caso o pedido seja para aterrissar, o estado do avião é alterado para *landing* e, passado algum tempo, para *in\_station*. Caso seja para levantar voo, o estado passa para *taking\_off* e, finalmente, para *flying*.

## Airline

O agente *Airline* representa uma **Companhia Aérea** e tem como principal objetivo realizar a compra de vagas nas gares do aeroporto. Deste modo, a atividade de aterrissar um avião depende obrigatoriamente da existência de vagas compradas pela sua companhia aérea. Os *behaviours* associados a este agente são:

- **BuySpots:** Envia a proposta de compra de vagas nas gares ([mensagem protocolar \*BuySpots\* 2.2](#)), que é negociada diretamente com o *StationManager*.
- **ReceiveProposalAnswer:** Recebe a resposta do *StationManager* sobre o sucesso (ou não) da compra das vagas. Caso houver sucesso na compra, informa o agente *DashboardAirline* sobre os novos valores do número de vagas obtidas (para a atualização dos dados da *Airline* na *dashboard*).
- **InformDashboardInitState:** Avisa o agente *DashboardAirline* da existência da *Airline*. Isto é, o seu ID, número de vagas para aviões de mercadoria e para aviões comerciais que a companhia aérea tem no aeroporto.



## StationManager

O agente *StationManager* representa o **Gestor dos Gares** e tem como função gerir o estado de cada gare presente no aeroporto. Cada gare é constituída por (1) um ID, (2) localização, (3) capacidade de vagas para aviões comerciais e de mercadoria e (4) um mapeamento do número de vagas compradas e utilizadas por cada companhia aérea. Os *behaviours* associados a este agente são:

- **ReceiveAirlinesProposals:** Recebe as propostas de compra de vagas das companhias aéreas.
- **EvaluateAirlinesProposals:** Comportamento periódico no qual, a cada intervalo de tempo, avalia as propostas das companhias aéreas. As propostas são avaliadas ordenadamente pelo preço por vaga proposto (propor um preço alto por vaga implica ter a proposta avaliada antes das outras). Na situação do número de vagas solicitado na proposta ser disponibilizado, é então enviada a resposta de sucesso de compra para o agente *Airline* correspondente.
- **ReceiveSpotQuery:** Responde ao agente *ControlTower* sobre a *query*: "*Quais são as gares com vagas do tipo X disponíveis da companhia aérea Y?*".
- **InformDashboardInitState:** Informa o agente *DashboardStation* sobre as gares existentes. Para isso, indica o ID de cada gare, assim como o número de vagas existentes para aviões comerciais e de mercadoria.
- **UpdateStationAvailability:** Recebe do agente *ControlTower* a informação de que uma vaga da companhia aérea **X** na gare **Y** está livre ou foi ocupada. Desta forma, o estado correspondente à tal gare é atualizado. Para além disto, é atualizado o agente *DashboardStation* sobre o número de vagas disponíveis nesta gare.

## RunwayManager

O agente *RunwayManager* representa o **Gestor de Pistas**, que assume o cargo de gerir o estado de cada pista presente no aeroporto. Cada pista é caracterizada por (1) um ID, (2) uma localização e (3) um estado de disponibilidade. Este gestor acarreta, por isso, funções como devolver as pistas que se encontram disponíveis e atualizar internamente o seu estado antes e depois de serem utilizadas. Os *behaviours* associados a este agente são:

- **InformDashboardInitRunway:** Informa o agente *DashboardRunway* sobre as pistas existentes. Para isso, indica o ID de cada pista, assim como a sua localização, no formato de coordenadas (**X,Y**), e o seu estado de disponibilidade.
- **ReceiveSpotQuery:** Envia uma resposta para a *ControlTower* relativamente à *query*: "*Quais são as pistas que não estão a ser utilizadas de momento?*".
- **UpdateRunwayAvailability:** Recebe do agente *ControlTower* a informação de que uma pista está livre ou foi ocupada, para que possa atualizar o seu estado. Adicionalmente, envia uma mensagem ao agente *DashboardRunway* com a finalidade de alterar, também, os dados que este apresenta.



## ControlTower

O agente *ControlTower* representa a **Torre de Controlo**, responsável por controlar e coordenar todas as ações que acontecem no aeroporto. A *ControlTower* é constituída por três dicionários: *queueInTheAir* – para cada companhia aérea, armazena uma lista de pedidos de aviões que estão em espera; *requestsInProcess* – para cada avião, armazena o pedido ([mensagem protocolar \*RequestFromAirplane\* 2.2](#)) em processamento; e *lockRequests*, cuja semântica é idêntica à do *requestsInProcess*, que possibilita uma espécie de zona de restrição de maneira a que os pedidos possam ser tratados um a um e não haja conflitos entre aviões na reserva de espaços. Este agente ainda estabelece, por *default*, um limite máximo para a fila de espera (= 30). Por questões de estatística, também guarda o tempo médio de espera por parte dos aviões, bem como o número daqueles que aterraram.

A *ControlTower* possui somente um comportamento (cíclico), que fica à escuta de pedidos e respostas dos outros agentes. Esse comportamento funciona do seguinte modo:

- **Performativa *request*:** Se chegar um pedido e a lista de espera não estiver cheia<sup>1</sup>, a Torre de Controlo processa-o. Para tal, sendo a solicitação do tipo *land*, pergunta ao *StationManager* pelas gares disponíveis; caso contrário, pede logo ao *RunwayManager* as pistas livres. O pedido é, assim, assinalado como “em processamento”.
- **Performativa *cancel*:** Caso o avião desista de esperar por uma resposta, a Torre de Controlo recebe um aviso de que o mesmo partiu para outro aeroporto.
- **Performativa *refuse*:** Na situação de não existir espaços livres, quer sejam pistas, quer sejam gares, a Torre de Controlo é notificada do sucedido. Como resultado, o pedido é colocado em espera e o avião é informado para aguardar. Se já tiver sido reservado uma *runway* que não está a ser ocupada, então é processado o próximo pedido com mais prioridade.
- **Performativa *inform*:** O avião comunica o seu atual estado de forma a manter atualizada a *ControlTower* e a *DashboardControlTower*. Além disso, se o estado do avião for *in\_station* ou *flying*, a pista e a gare usadas são liberadas, respetivamente, pelos seus respetivos gestores. Isto permite que o próximo pedido com mais prioridade seja tratado com esses precisos espaços.
- **Performativa *inform-if*:** Sempre que uma *airline* comprar novas vagas de estacionamento nas *stations*, então o avião com mais prioridade dessa companhia que está à espera tem a oportunidade de aterrizar.
- **Performativa *confirm*:** Quando existem vagas de estacionamento livres, a Torre de Controlo solicita pistas ao *RunwayManager*. Por outro lado, havendo *runways* desocupadas, uma delas é selecionada aleatoriamente e, em seguida, é determinada a gare mais próxima dessa pista<sup>1</sup>. Deste modo, a *ControlTower* informa a *RunwayManager* e a *StationManager* de que esses recursos foram reservados, permitindo que eles alterem os seus estados.

<sup>1</sup> Apenas se o tipo do pedido for *land*, i.e., para aterrizar.



Em qualquer tipo de performativa, a *ControlTower* envia uma mensagem à *DashboardControlTower* para que esta possa apresentar, em tempo real, todas as ações realizadas pelos aviões que entram e saem do aeroporto ao longo do funcionamento do sistema.

## 2.2 Mensagens protocolares

- **BuySpots:** Mensagem protocolar enviada na proposta de compra de vagas, sendo composta por (1) ID da companhia aérea, (2) número de vagas a serem compradas, (3) preço proposto **por vaga** e (4) tipo de vaga (comercial ou de mercadoria).
- **DashboardAirlinesMessage:** Mensagem enviada para a *DashboardAirline* alterar os dados das companhias aéreas, sendo constituída por (1) o tipo de mensagem (*negotiation*, *info* ou *update*), (2) o texto da negociação, (3) os dados da *airline* e (4) o estado da negociação (*propose*, *success* ou *fail*).
- **DashboardAirplaneMessage:** Mensagem enviada para a *DashboardAirplane* alterar os dados dos aviões, sendo constituída por (1) o tipo de mensagem (*create* ou *update*) e (2) os dados do avião em questão.
- **DashboardControlTowerMessage:** Mensagem enviada para a *DashboardControlTower* apresentar as ações decorridas ao longo do ciclo de vida do sistema, sendo constituída por (1) o tipo de mensagem, (2) o texto da mensagem, (3) o tipo de pedido (*land* ou *take\_off*), (4) o estado do avião, (5) o texto de aprovação, (6) o texto sobre as novas vagas compradas, (7) o tempo médio na fila de espera e (8) o texto de recusa.
- **DashboardRunwayMessage:** Mensagem enviada para a *DashboardRunway* alterar os dados das pistas, possuindo (1) o tipo de mensagem (*info* ou *update*) e (2) a lista de runways criadas (opcional) **ou** (2) os dados da pista a atualizar (opcional).
- **DashboardStationMessage:** Mensagem enviada para a *DashboardStation* alterar os dados acerca das gares, incorporando (1) o tipo de mensagem (*info* ou *update*) e (2) a lista de gares criadas (opcional) **ou** (2) os dados da gare a atualizar (opcional).
- **IsRunwayAvailable:** Mensagem enviada para atualizar o estado de disponibilidade de uma determinada pista, exigindo (1) o tal estado e (2) a pista em questão.
- **IsStationAvailable:** Mensagem enviada para atualizar a capacidade de uma determinada gare, exigindo (1) o estado de disponibilidade, (2) a gare em questão, (3) o ID da sua *airline* e (4) o tipo do transporte.
- **NewSpotsAvailable:** Mensagem enviada para atualizar o número de vagas adquiridas por uma companhia aérea, sendo composta por (1) o ID da *airline*, (2) o tipo do transporte e (3) o número de vagas que possui.
- **RequestFromAirplane:** Mensagem enviada sempre que avião pretende aterravar ou descolar, preenchida à medida que os agentes vão interagindo. Esta é composta por (1) o ID do *avião*, (2) o tipo do transporte e (3) o estado do avião, (4) o ID da *airline*, (5) a data em que foi feito, (6) o nível de prioridade, (7) a gare onde poderá estacionar (opcional), (8) a pista onde poderá aterravar ou descolar (opcional) e (9) a gare onde está inicialmente estacionada (opcional).

## 2.3 Arquitetura baseada em agentes

A modelação dos diagramas 1, 2, 3 e 4 foi essencial para representar visualmente a arquitetura e a lógica por trás do sistema multiagente, permitindo assim uma compreensão clara acerca das interações entre os agentes e das atividades que cada um desempenha.

### Diagrama de Classes e de Pacotes

Representado na Figura 1 como a fusão dos diagramas de classes e de pacotes, este diagrama ilustra as classes dos cinco agentes principais e das suas respectivas *dashboards*, localizadas numa diretoria diferente. Cada agente possui um pacote chamado “behaviours”, que contém os comportamentos que cada um desempenha. Quer os comportamentos dos agentes, quer o das *dashboards*, possuem a mesma estrutura de variáveis e métodos.

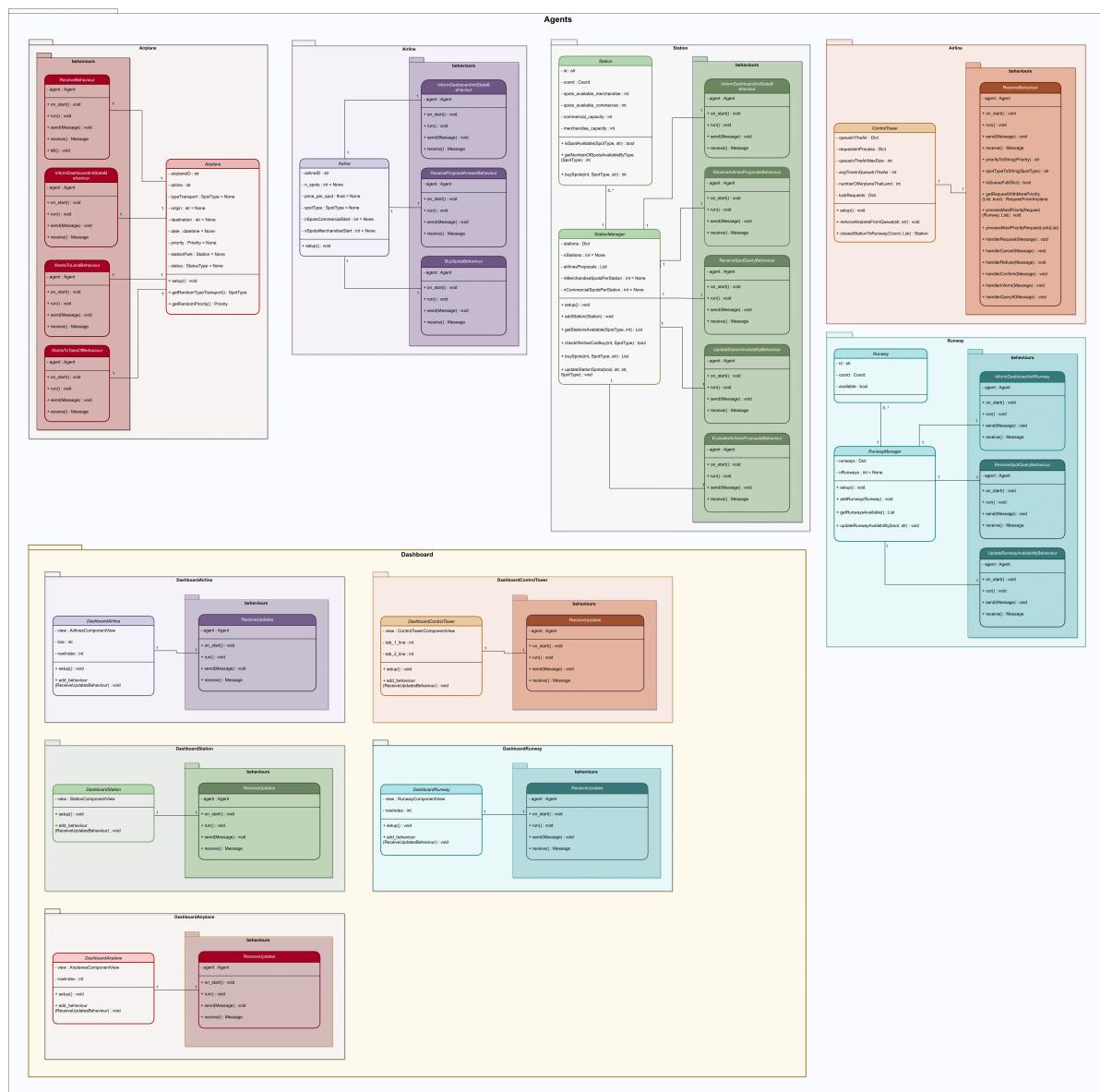


Figura 1: Diagrama de Classes e de Pacotes

## Máquina de Estados

Apesar de não incorporar o *Finite State Machine Behaviour* do *Spade*, o agente *Airplane* possui uma estrutura muito semelhante à de uma máquina de estados. Por este motivo, a Figura 2 mostra os diferentes estados nos quais um avião pode estar. Como decisões de implementação, optou-se por definir um tempo único de **circulação na pista**, relativo ao período de transição de um avião do estado *taking-off* para o estado *flying*, assim como para o intervalo de tempo entre os estados *landing* e *in\_station*.

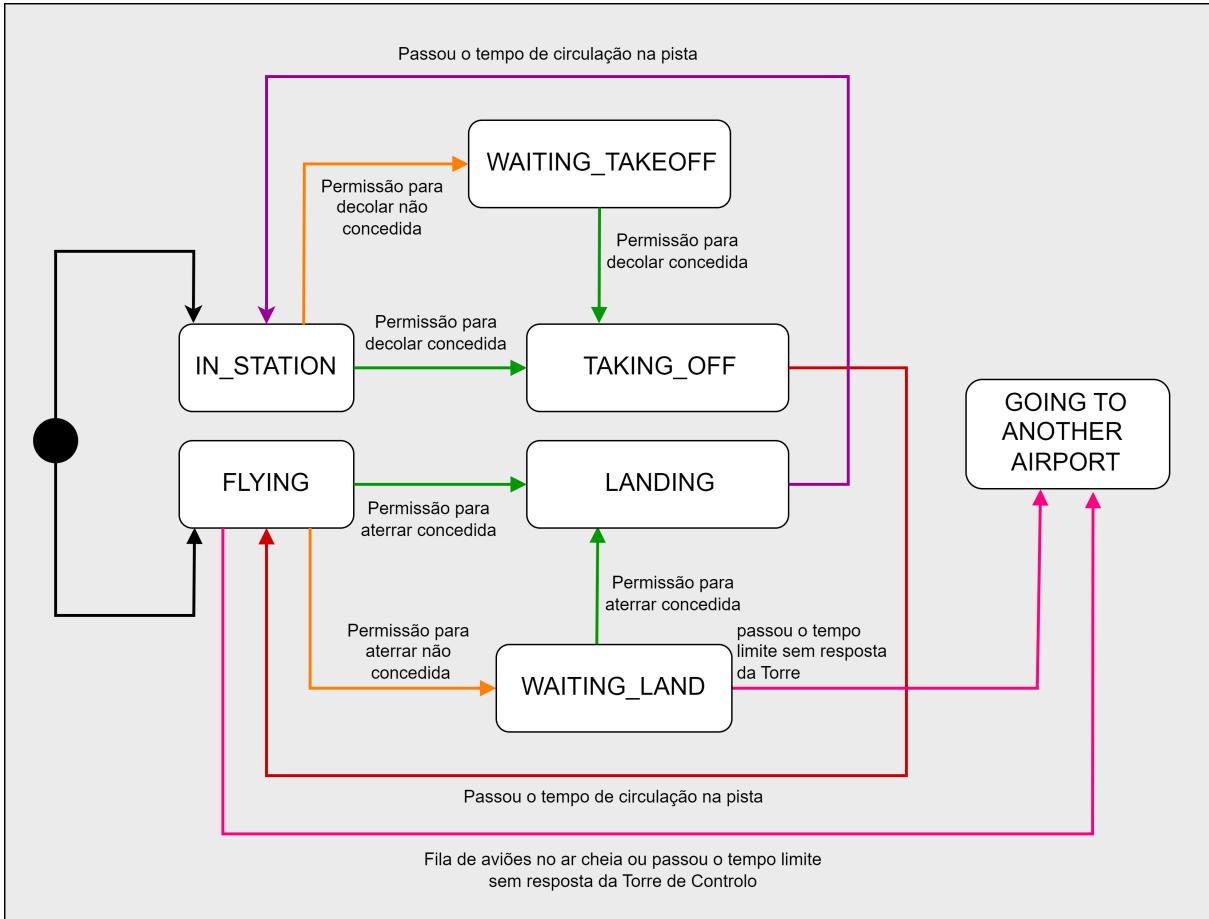


Figura 2: Máquina de Estados do Avião

## Diagrama de Atividades

O diagrama de atividades, presente na Figura 3, ilustra o fluxo de trabalho completo, resultante das interações entre os agentes e das atividades executadas por cada um. O processo começa quando o avião solicita aterrissar ou descolar. Na situação da fila de espera estar cheia, a Torre de Controlo informa ao requerente que deve partir para outro aeroporto. Caso contrário, o pedido é processado. Se a solicitação for para **aterrar**, a Torre de Controlo consulta o Gestor de Gares para verificar a disponibilidade das *stations*. Em caso afirmativo, os resultados são armazenados pelo agente central, que então pede ao Gestor de Pistas informações sobre as *runways* livres. Em contrapartida, se a solicitação for para **levantar voo**, a Torre de Controlo entra em contato imediato com o Gestor de Pistas. Após receber os dados sobre essas pistas, e se o objetivo for pousar, a Torre de

Controlo calcula a gare mais próxima da pista sorteada. Por fim, o agente central informa o avião acerca dos espaços necessários para realizar a ação requerida, atualiza a *dashboard* com as novas informações sobre os voos e o fluxo de trabalho é concluído. Vale ressaltar que, caso não haja pistas ou gares desocupadas, é sugerido ao avião que aguarde.

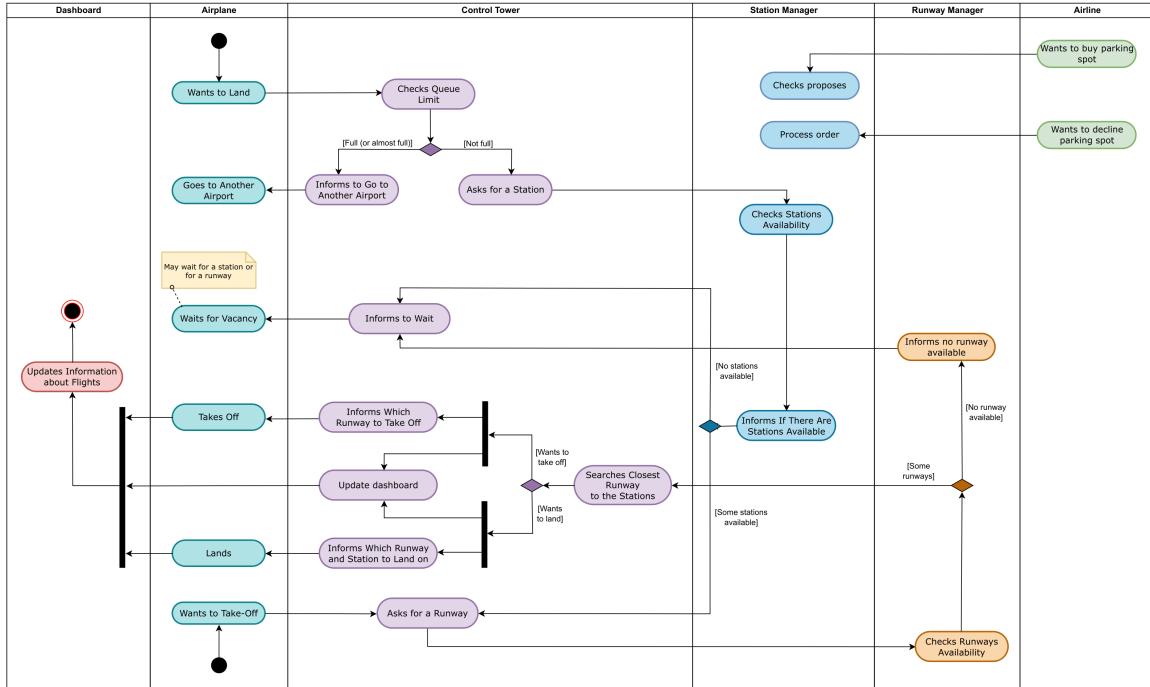


Figura 3: Diagrama de Atividades

## Diagramas de Sequência

A Figura 4 mostra alguns dos diálogos mais relevantes entre as entidades do sistema. O diagrama “Land” mostra como o *request* de aterragem é tratado. Primeiramente, é verificado se a fila de aviões em espera no ar está cheia. Em caso afirmativo, é imediatamente recusado o *request* e informado o avião para se dirigir para outro aeroporto. Caso contrário, o *request* é processado. Este processamento é desenvolvido no diagrama “Process Land Request”, uma vez que este é um processo comum e reutilizável noutras fases do sistema (como acontece também no diagrama “Inform In Station”, onde um avião libera uma pista e é processado o *request* mais prioritário à espera de uma pista).

O processamento de um pedido de aterragem é composto por três fases. A primeira corresponde ao agente *ControlTower* interrogando o *StationManager* sobre quais as **gares** disponíveis para o avião do *request*. A segunda fase, semelhante à primeira, destina-se, por sua vez, à obtenção das **pistas** candidatas para a realização da aterragem. No caso de insucesso em alguma dessas fases, o avião é solicitado a esperar e o seu *request* é mantido numa lista de *requests* a serem tratados no futuro. No caso de sucesso, é realizada a terceira fase do processo. Esta fase refere-se à seleção da pista e da gare mais próxima para a concretização da atividade. Adicionalmente, os agentes *StationManager* e *RunwayManager* são atualizados com a informação da gare e da pista selecionadas, respetivamente, de modo a “reservá-las” e torná-las indisponíveis. Por fim, o avião é, então, informado de que pode realizar a aterragem.

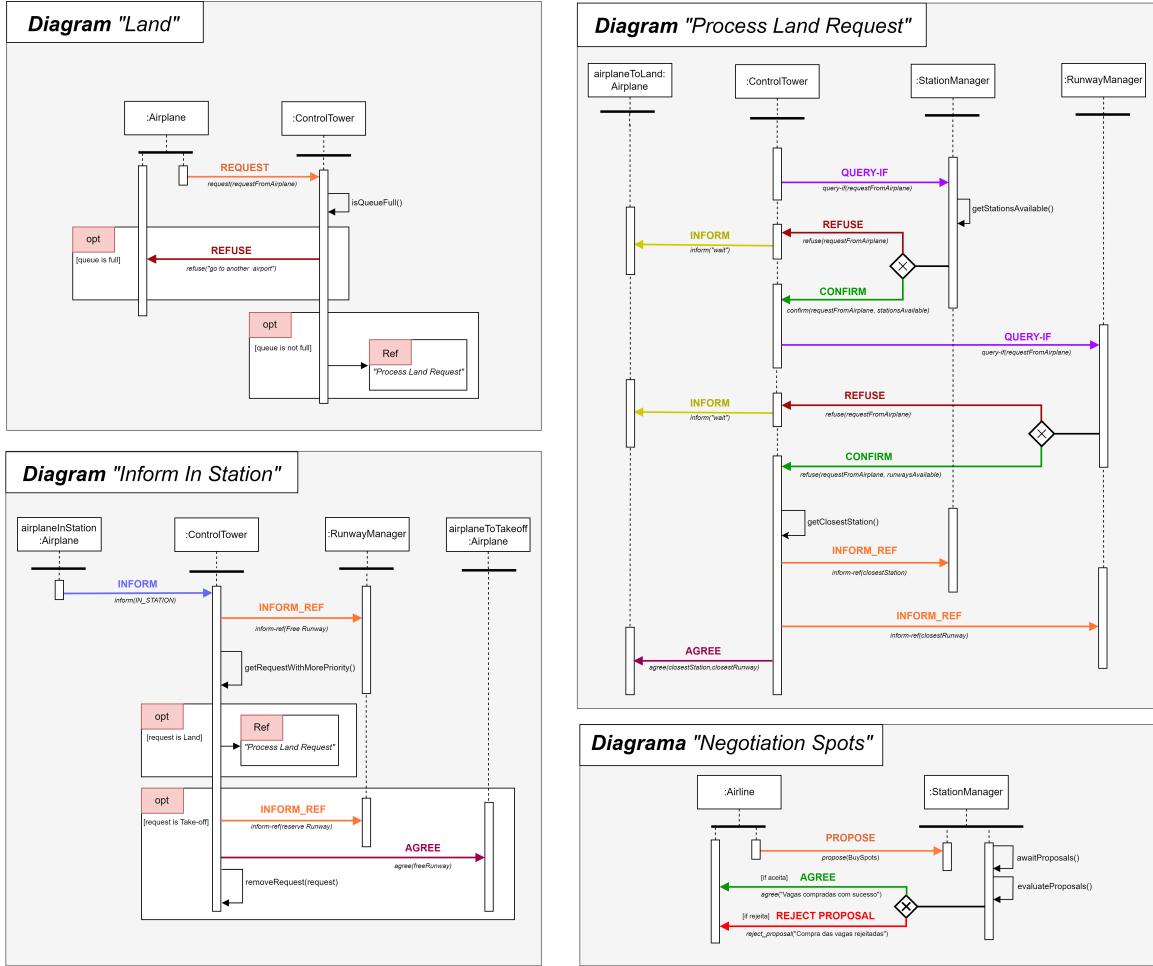


Figura 4: Diagramas de Sequência

O processo inerente a um *request* de descolagem é muito parecido com o previamente explicado para a aterragem. A diferença reside na ausência da etapa de obtenção das gares disponíveis, uma vez que, nesta ação, o avião já se encontra estacionado e apenas é preciso uma pista. Outro cenário possível é quando um avião avisa a torre de controlo de que está a voar. Esta informação proporciona a liberação de uma pista que, por sua vez, desencadeia o processamento do *request* mais prioritário à espera. Assim, esse cenário apresenta semelhanças com as ações ilustradas no diagrama “Inform In Station”.

Além do fluxo de tráfego aéreo no aeroporto, o diagrama “*Negotiation Spots*” mostra o fluxo de mensagens na negociação de compras de vagas de estacionamento entre as companhias aéreas e o *StationManager*. Assim que uma proposta de compra é recebida pelo *StationManager*, a mesma é armazenada numa lista que contém as propostas de todas as companhias aéreas. Após um determinado tempo, esta proposta é analisada e o respetivo *feedback* é enviado de volta para a *Airline* correspondente.

Um detalhe que vale a pena mencionar é ausência das comunicações com os agentes *dashboards* nos diagramas anteriores. Essa decisão foi tomada para assegurar a legibilidade e reduzir a complexidade dos diagramas. Porém, é importante ressaltar que, em diversos momentos, as *dashboards* em são notificadas das ações realizadas.



## 3 Resultados Obtidos

### Teste 1

O cenário de teste concebido ambicionava analisar o comportamento do sistema perante a existência de uma única pista no aeroporto. Ao induzir esta limitação, poderíamos observar as prioridades dos *requests* atuando como critério de escolha pela utilização pista. Para facilitar a compreensão dos resultados, optou-se pela utilização de apenas dois aviões com intenção de aterrarr e dois aviões com intenção de levantar voo. Além disto, uma única companhia aérea comum a todos os aviões é criada, por forma a testar também a compra de vagas nas gares e observar como o sistema reage a esta nova aquisição. Em relação às gares, foi criado somente uma com capacidade suficiente para lidar com a demanda de aviões deste cenário. A Figura 5 mostra precisamente a configuração completa e inicial do teste desenvolvido.

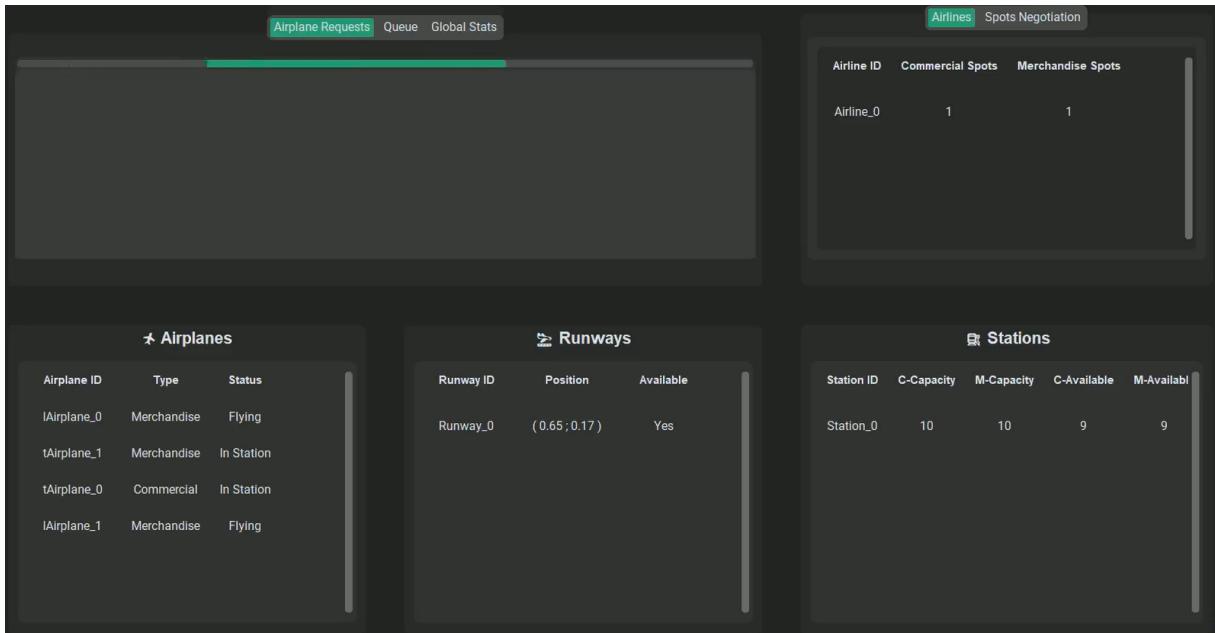


Figura 5: Teste 1 - Instante inicial

Como se pode verificar, a gare começa logo com duas vagas ocupadas (uma comercial e outra de mercadoria) pelos aviões que pretendem descolar. Uma vez que ambos os aviões estacionados são da companhia aérea *Airline\_0*, tal *airline* já inicia a simulação com uma vaga comprada para aviões comerciais e outra para aviões de mercadoria. Adicionalmente, na ausência de *requests*, a única pista do sistema encontra-se disponível.

A Figura 6 mostra o estado da simulação após todos os aviões enviarem *requests* à Torre de Controlo. Como era de se esperar, os pedidos de aterragem dos aviões *lAirplane\_1* e *lAirplane\_0* foram **negados**. Isto porque, a companhia aérea destes aviões, até então, apenas tinha direito a **uma** vaga para aviões de mercadoria. Em contrapartida, foi dada a permissão de levantar voo ao avião *tAirplane\_0*, dado haver disponibilidade da pista *Runway\_0*. Todavia, essa permissão foi negada ao pedido seguinte, também para descolar, do avião *tAirplane\_1*. Isto porque, sendo essa pista reservada para a aterragem do avião *tAirplane\_0*, de momento não se encontra mais desocupada.



The screenshot shows the Control Tower interface with three main panels:

- Airplane Requests:** Displays log entries for airplane requests and permissions. One entry shows an airplane from Airline\_0 requesting to land at 15:28:59 with priority HIGH, and another shows a proposal for 2 spots of type Merchandise.
- Runways:** Shows a single runway available at position (0.65, 0.17).
- Stations:** Shows one station with C-Capacity 10 and M-Capacity 10, both currently available.

 Figura 6: Teste 1 - Primeiros *requests*

Ainda na Figura 6, é possível detetar a atualização dos estados dos aviões, encontrando-se três deles num estado de espera enquanto o avião *tAirplane\_1* está no processo de descolagem (*taking off*). O número de vagas disponíveis para aviões comerciais também foi modificado, uma vez que o avião em atividade de descolagem libertou a sua vaga. Para além disto, pode ser notada a proposta de compra de duas vagas para aviões de mercadorias, efetuada pela companhia aérea. Esta negociação, ao ser aceite, desencadeia uma nova avaliação da Torre de Controlo sobre os pedidos anteriormente negados. Com a adição de novos *spots*, talvez seja possível atender algum deles. Porém, como a única pista do aeroporto ainda está a ser utilizada, não foi dada permissão aos *requests* de aterragem reavaliados (como pode ser visto na Figura 7).

The screenshot shows the Control Tower interface with the following changes:

- Airplane Requests:** The log now includes a green message indicating the acceptance of the proposal for 2 spots of type Merchandise.
- Runways:** The runway availability status has been updated.
- Stations:** The station availability status has been updated.

Figura 7: Teste 1 - Compra de vagas

A Torre de Controlo recebe uma mensagem quando um determinado avião levanta voo. Isso ocorre na altura em que o agente *Airplane* transita do estado *taking-off* para o estado *flying*, informando a *ControlTower*. Esta ação demonstra a liberação de uma pista, permitindo que a Torre de Controlo reavalie os pedidos anteriormente negados e escolha o mais prioritário para tirar proveito da pista agora livre.

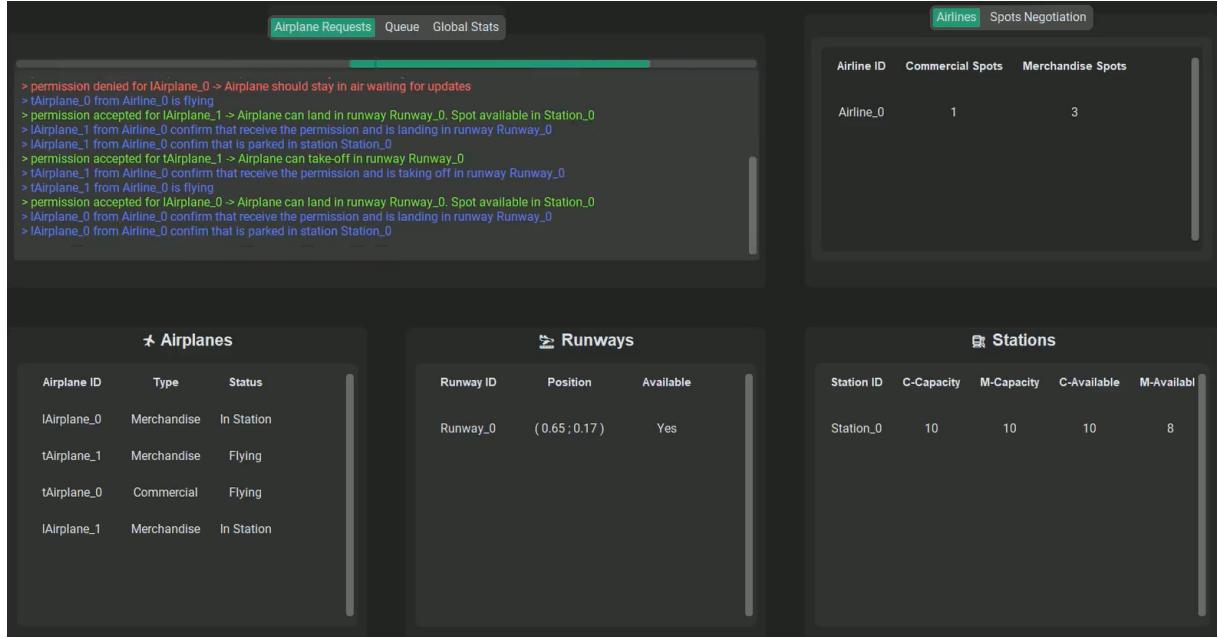


Figura 8: Teste 1 - Fim da simulação

A Figura 8 ilustra o resultado da simulação após o avião *tAirplane\_0* informar ao agente central deste sistema que está a voar. Conforme explicado antes, este agente reavaliou os *requests* pendentes e decidiu atender o *request* do *lAirplane\_1*. A justificação para essa tomada de decisão está apresentada na Figura 6, onde é possível observar que a solicitação desse avião possuía o nível de prioridade mais elevado (nível *high*), juntamente com o do avião *tAirplane\_1*. Como o *request* do avião *lAirplane\_1* foi enviado **primeiro**, o **critério da data de emissão do pedido** serviu para resolver o conflito.

Depois do avião *lAirplane\_1* aterrizar, o *tAirplane\_0* descolou, seguido da aterragem do avião *lAirplane\_0*, visto possuir o pedido de menor prioridade. É importante salientar que o *request* do *lAirplane\_0* só pôde ser atendido já que a sua companhia aérea possui três vagas para aviões de mercadoria nas gares, anteriormente adquiridas. Como esperado, o estado final dos aviões é **oposto** aos seus estados iniciais: aqueles que começaram no estado *flying* terminaram a simulação no estado *in\_station*, enquanto os que começaram no estado *in\_station* acabaram no estado *flying*. Além disso, a única pista do aeroporto encontra-se disponível. Quanto às gares, a única existente possui todas as vagas de aviões comerciais livres, com apenas duas vagas ocupadas por aviões de mercadoria.

Com esta simulação, pôde-se testar os níveis de prioridade e as datas dos pedidos, a gestão do estado das pistas e das gares, a negociação de compra de vagas nas gares por uma companhia aérea e a visualização, em tempo real, da evolução do sistema na interface gráfica. Adicionalmente, como demonstrado pelas figuras, procurou-se utilizar as mensagens descritivas e completas para a compreensão do processamento que ocorre na Torre de Controle. Estas mensagens oferecem informações desde a data e nível de prioridade dos *requests*, até à decisão de aprovação ou recusa sobre os pedidos avaliados.



## Teste 2

O segundo cenário de teste foi planeado para a observação do tratamento paralelo de *requests* dos aviões. Isto é, diferente do primeiro cenário em que só havia uma única pista compartilhada pelos aviões, neste cenário são utilizadas **duas** pistas. Para além disto, são gerados **seis** aviões para aumentar a complexidade do teste. A imagem 9 mostra a configuração inicial do cenário.

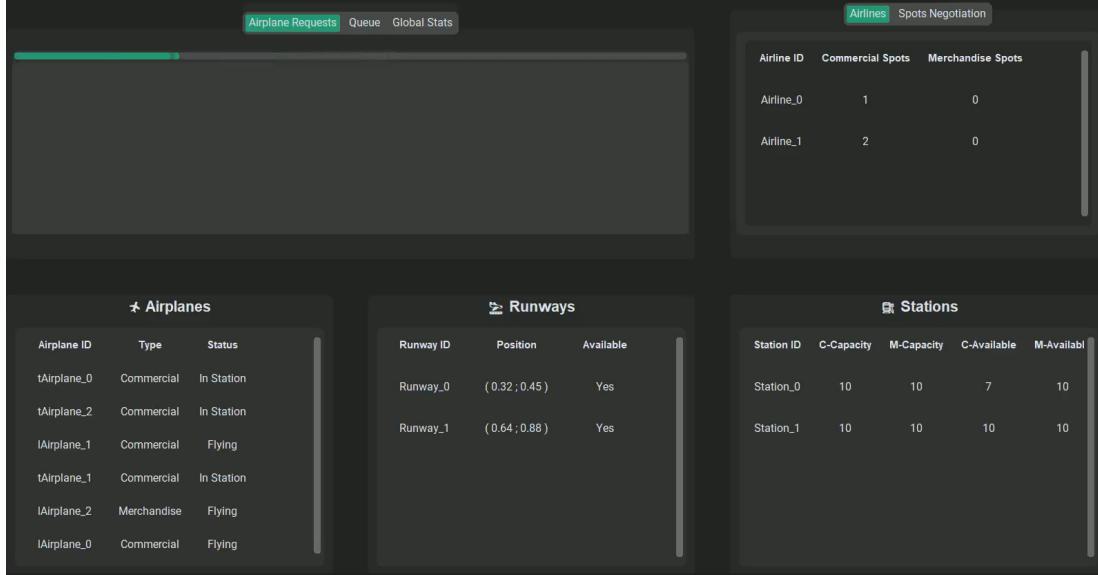


Figura 9: Teste 2 - Instante inicial

Conforme o previsto, inicialmente as duas pistas estão disponíveis. Neste cenário foram também criados **duas** gares, cada uma com dez vagas de aviões comerciais e dez vagas para aviões de mercadoria. Uma vez que existem **três** aviões comerciais na gare *Station\_0*, existem então somente **sete** lugares para aviões comerciais nesta gare.

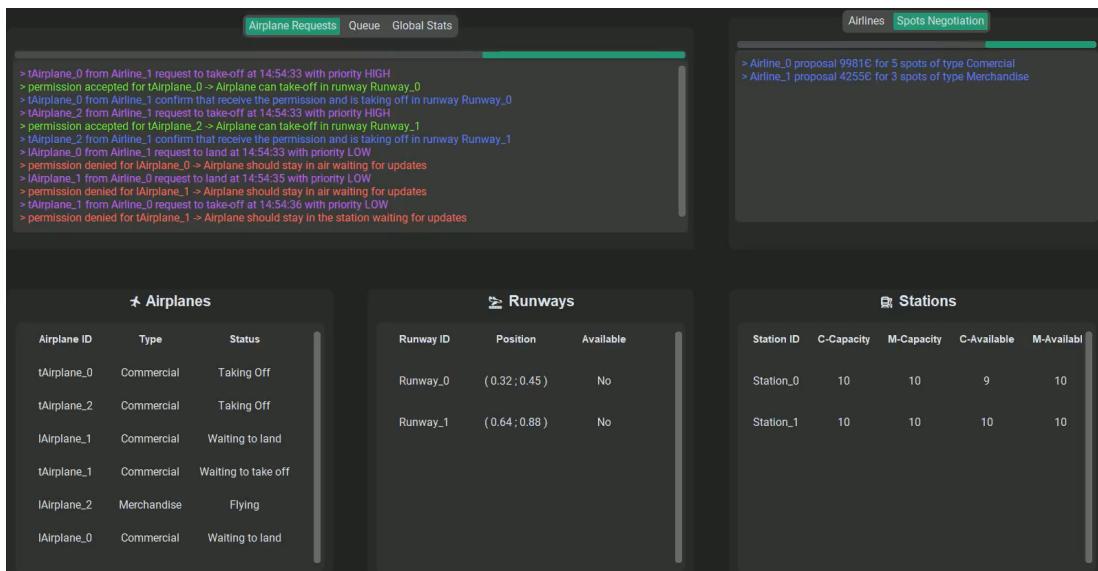


Figura 10: Teste 2 - primeiros *requests*



A Figura 10 mostra o tratamento dos primeiros *requests* enviados pelos aviões. Como os dois primeiros *requests* foram para levantar voo e ambas as pistas estavam disponíveis, então tais pedidos foram aceites (é possível ainda observar quais foram as pistas atribuídas a cada avião). Em contrapartida, os restantes aviões tiveram os seus pedidos negados, uma vez que as pistas do aeroporto já estavam reservadas. Nesta mesma imagem é possível ver o estado das pistas atualizado, assim como a liberação das duas vagas na gare *Station\_0* pelos aviões em processo de levantar voo (que antes estava com sete vagas disponíveis). Com exceção do avião *tAirplane\_2* que ainda não recebeu a resposta da torre de controlo sobre o seu *request*, todos os restantes aviões que tiveram os seus pedidos negados estão num estado de **espera**.



Figura 11: Teste 2 - Compra de vagas

A Figura 11 mostra o que acontece na simulação após as propostas de compra de vagas serem avaliadas. Os pedidos de aterrizar pendentes de aprovação foram reavaliados (somente dos aviões das companhias aéreas que compraram vagas). O *request* do avião *tAirplane\_1* não foi reavaliado uma vez que este deseja levantar voo e, portanto, não está interessado na obtenção de uma vaga na gare. A conclusão desta etapa da simulação foi, novamente, a negação dos pedidos, visto que as duas pistas do aeroporto ainda estão a ser utilizadas.

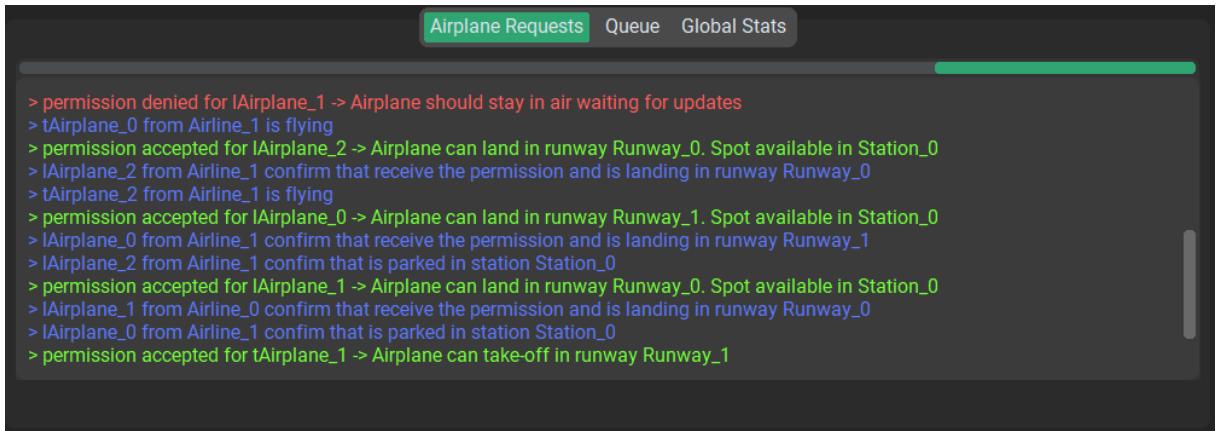


Figura 12: Teste 2 - Processamento dos *requests* pendentes

A Figura 12 revela como ocorreu o tratamento dos pedidos pendentes de aprovação conforme as pistas foram liberadas. Após o avião *tAirplane\_1* notificar a torre de controlo que já está no ar, o *request* de aterrizar do avião *tAirplane\_2* foi aprovado (dado este ser o de maior prioridade como pode ser visto na imagem 10). De seguida, o avião *tAirplane\_2* também notifica que está no ar, o que ocasiona na aprovação do *request* do



avião *tAirplane\_0*. Este avião havia enviado um *request* de aterragem com prioridade igual ao dos restantes aviões, porém, por ser o mais antigo, foi tratado em primeiro lugar. À medida que os aviões em processo de aterragem terminavam as suas operações, os restantes pedidos eram tratados. A imagem 12 ainda mostra que são enviadas a pista e a gare para os aviões com pedidos de aterrizar aceites.

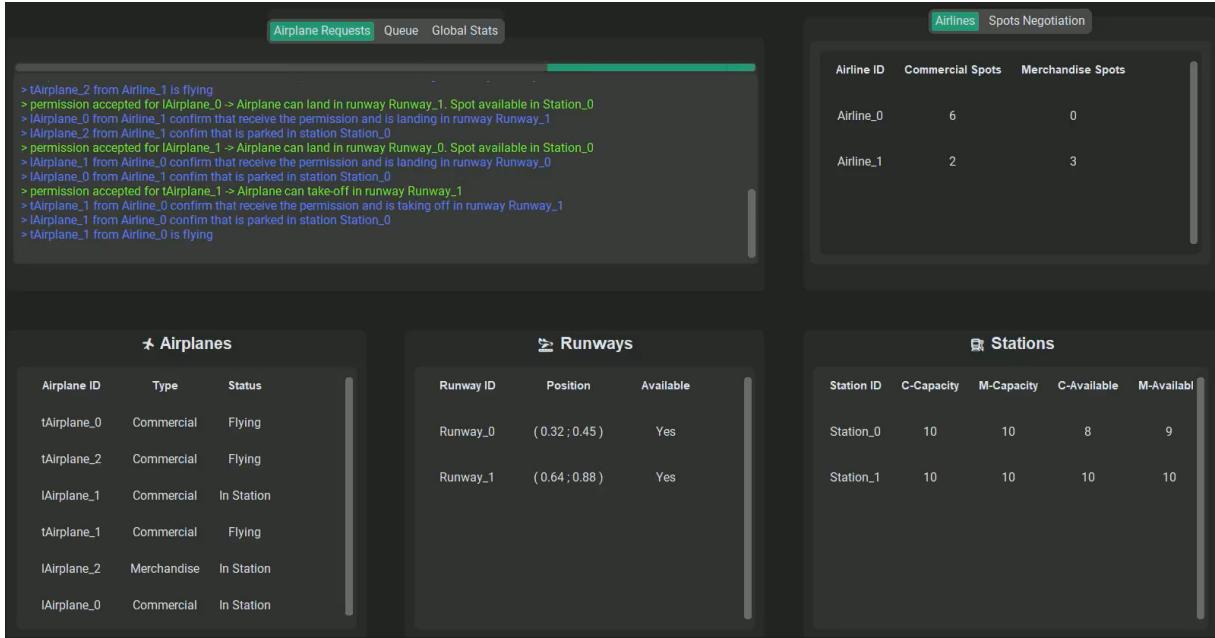


Figura 13: Teste 2 - Fim da simulação

Finalmente, a Figura 13 retrata o aspecto final da simulação. Os aviões que começaram no estado *flying* encontram-se, agora, no estado *in station* (e vice-versa). Os aviões nas gares estão todos na *Station\_0*, sendo **dois** deles do tipo **comercial** e **um** do tipo de **mercadoria**. Estes valores estão em congruência com os **oito** lugares disponíveis para aviões comerciais e **nove** lugares disponíveis para aviões de mercadoria nesta gare. As **duas** pistas do aeroporto também tiveram os seus estados atualizados como **livres**. Para além disto, a Figura 14 mostra mais uma **feature** do sistema, que é a informação sobre o tempo médio de espera dos aviões que aguardam pelos seus *requests* de aterragem.

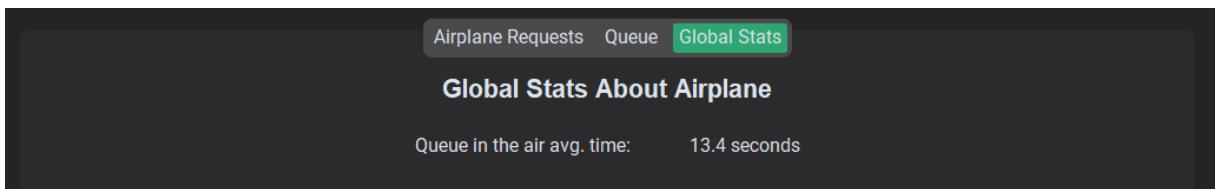


Figura 14: Teste 2 - Tempo de espera no ar

Com esta simulação, pode-se mais uma vez observar os níveis de prioridade e as datas dos pedidos agindo como critério de escolha em conflitos. Para além disto, foi observado a distribuição de diferentes pistas para aviões que realizaram o seu processo de levantar voo de forma **simultânea**. Os resultados obtidos vão de encontro com o primeiro teste realizado, reforçando a consistência do sistema desenvolvido.



## 4 Funcionalidades Extra

- **Prioridade dos *requests*:** Cada *request*, seja de aterragem ou descolagem, vem associado a um nível de prioridade. Este nível pode ser *low*, *medium* ou *high*, servindo como critério de seleção do *request* mais prioritário em momentos de conflito entre aviões que disputam por pistas/gares livres. No caso de prioridades iguais, o *request* mais antigo é o escolhido.
- **Negociação e distribuição de vagas nos gares pelas companhias aéreas:** Cada gare possui uma capacidade distinta de vagas para aviões comerciais e de mercadorias. Cada companhia aérea, por sua vez, possui um número de vagas adquiridas em cada gare para os seus aviões estacionarem. As companhias aéreas enviam propostas de compra de vagas para o agente *StationManager*, que as avaliará, atribuindo as vagas segundo o preço por vaga proposto.
- **Tempo média de espera dos aviões no ar:** É calculado o tempo médio que os aviões aguardam pela permissão de aterrizar.
- **Extensão do agente *Dashboard* proposto:** Um dos requisitos propostos envolvia a criação de um agente responsável por apresentar a informação geral sobre os voos em operação. Como consequência, o grupo de trabalho concebeu um sistema que não se limita apenas em apresentar informação sobre os voos. Na verdade, são ainda agregadas informações relativas ao (1) estado dos aviões, (2) estado de ocupação das gares, estado das pistas (3), (4) companhias aéreas e (5) mensagens trocadas na negociação de vagas. Para tal, foram criados agentes *dashboards* distintos para cada propósito de agregação de informação, sendo estes os agentes *DashboardAirplane*, *DashboardAirline*, *DashboardStation* e *DashboardRunway*.
- **Interface Gráfica e configuração detalhada da simulação:** A evolução de cada agente do sistema pode ser visualizada, em tempo real, através da interface gráfica concebida – Figuras 5, 6, 7 e 8. Para além disto, antes da simulação começar, o utilizador pode configurar alguns parâmetros, como pode ser visto na Figura 15.

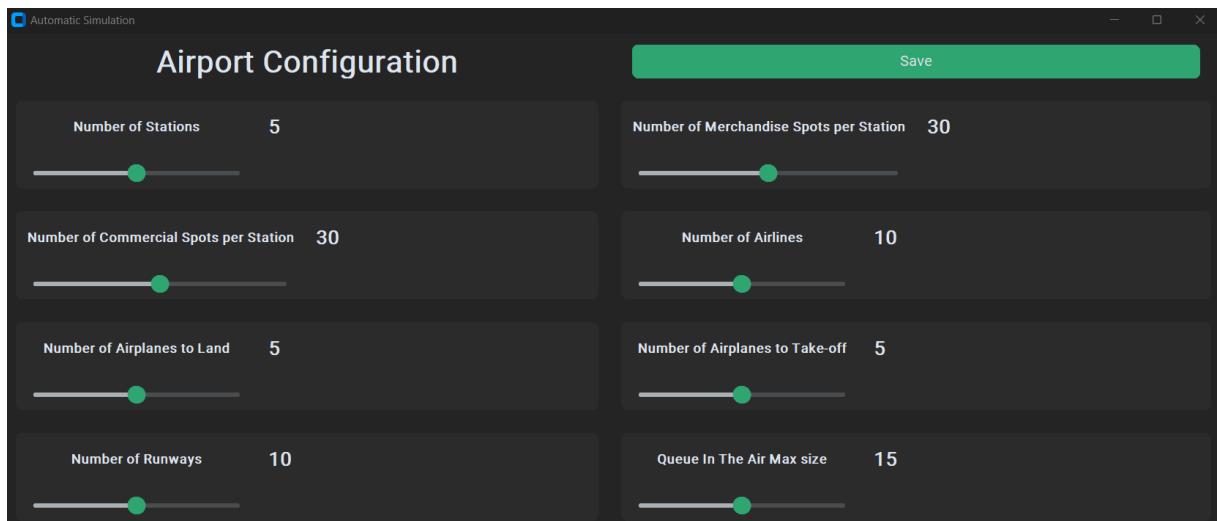


Figura 15: Tela de configuração



## 5 Análise Crítica e Trabalho Futuro

Uma refatoração do agente *Airplane* é um possível trabalho a considerar no futuro. Conforme enunciado no capítulo 2, este agente possui uma estrutura comportamental que pode ser descrita em termos de uma **máquina de estados**. Neste momento, o *Airplane* implementa um único *cyclic behaviour* para o recebimento de mensagens e, juntamente com a instância *status* correspondente ao seu estado, realiza todas as operações inerentes a este *behaviour*. Refactorar este comportamento para um *FSMBehaviour* seria interessante para uma melhor organização e simplificação do código.

No que diz respeito ao modelo do aeroporto, para o aproximar ainda mais da realidade, como trabalho futuro poderiam ser incorporados **tipos de pistas** específicos para os tipos de aviões. Isso porque, as aeronaves variam em tamanho, peso e características de decolagem e pouso, sendo necessário ter pistas adequadas para cada tipo de avião. Deste modo, novas características podem ser adotadas para a entidade avião, à medida que novas formas de cálculo de pistas podem surgir na *ControlTower*.

Relativamente ao ambiente de simulação desenvolvido através da interface gráfica, apesar do mesmo garantir uma enorme flexibilidade para simular diversos cenários, tal funcionalidade ainda pode ser melhorada. A título de exemplo temos fornecer ao utilizador a possibilidade de configuração da simulação a partir de um ficheiro de configuração (em formato *JSON*, *XML*, entre outros). Neste ficheiro, o utilizador poderia descrever as **características individuais** desejadas para cada agente. Isto é, enquanto que no modelo atual o utilizador pode gerar **X** aviões para aterrkar e **Y** aviões para levantar voo (sem poder controlar o ID, companhia aérea, tipo de avião, etc.), com a incorporação de um ficheiro de configuração, o utilizador poderia informar todas as características idealizadas para o avião criado. Esta proposta não precisa necessariamente de ser através de um ficheiro de configuração, podendo se basear numa própria expansão da interface gráfica.

Para além deste aspecto de configuração, um ficheiro de *logs* seria também uma boa adição ao trabalho. Isto porque, apesar de garantirmos a visualização em tempo real da simulação com a interface gráfica, com um ficheiro de *logs* poderíamos dar ao utilizador a possibilidade de rever todas as ações executadas na simulação. Esta adição poderia ser facilmente integrada, com a mesma lógica que são atualizados os agentes *dashboards*. Contudo, neste caso, a escrita para um ficheiro seria feita.

Em suma, o sistema concebido abrange os requisitos mínimos propostos, assim como algumas funcionalidades extras que enriquecem tanto o modelo do aeroporto, como a experiência de simulação do sistema. O cenário de teste comprova a execução esperada das ações para um cenário modesto, mas que abrange diversos aspectos do sistema.