Controle de Tráfego Aéreo Autônomo: Simulação de Sistemas Multiagentes com Framework JaCaMo na Plataforma Unity

Bernardo Viero, Alexandre Zamberlan
Curso de Ciência da Computação
UFN - Universidade Franciscana
Santa Maria - RS
bernardo.viero@ufn.edu.br, alexz@ufn.edu.br

***Resumo*—Este projeto integrou as áreas de Sistemas Multiagentes (SMA) e Simulação para desenvolver um sistema de gestão e controle autônomo de tráfego aéreo utilizando Inteligência Artificial Distribuída. Foram modelados agentes que representam aeronaves e pistas de pouso e decolagem, com capacidades de percepção, planejamento, execução e comunicação. Esses agentes exibiram comportamentos emergentes e dinâmicos, adaptando-se a situações geradas pela simulação por meio de estratégias de planejamento, comunicação e negociação. O desenvolvimento foi realizado em duas partes: no servidor, utilizou-se a linguagem Java com AgentSpeak(L) e o *framework* JaCaMo para gerenciar a lógica dos agentes e a infraestrutura de comunicação. Para a implementação visual e simulação gráfica, foi utilizada a linguagem C# com a plataforma Unity, proporcionando uma interface interativa que facilitou a observação e análise dos comportamentos dos agentes em tempo de simulação. Esse ambiente simulado permitiu a criação de cenários dinâmicos, contribuindo para o estudo de abordagens autônomas na gestão de tráfego aéreo.**

***Palavras-chave* : Inteligência Emergente; Agentes Inteligentes; Jason; Planejamento; Comunicação; Percepção; Tempo de simulação.**

I. Introdução

A Simulação Computacional tem um papel essencial na gestão e controle do tráfego aéreo, apoiando a tomada de decisões e a segurança das operações. A integração de Sistemas Multiagentes com Simulação Computacional permite novas abordagens para a gestão autônoma do tráfego aéreo, utilizando Inteligência Artificial Distribuída. Ao combinar essas áreas, cria-se um ambiente dinâmico, onde agentes simulados representam aeronaves e pistas de pouso, com capacidades de percepção, planejamento, execução e comunicação.

A simulação facilita a compreensão do comportamento desses sistemas, gerando cenários desafiadores e servindo como ferramenta para avaliar estratégias de controle e mitigar riscos. Tecnologias como Java, JaCaMo, AgentSpeak(L) e Unity possibilitam a criação de ambientes interativos e visualmente detalhados, facilitando a análise do comportamento inteligente das aeronaves.

Este projeto visou desenvolver um ambiente de simulação baseado em Sistemas Multiagentes, comparando seus resultados com um sistema centralizado controlado por torre. Espera-se, ao final, uma contribuição significativa para o avanço dessa área, com alternativas e percepções para os desafios atuais.

A. Justificativa

Este trabalho explorou uma perspectiva recente, utilizando Sistemas Multiagentes por meio do ambiente Jason, para desenvolver um sistema autônomo. A integração dessa tecnologia em um ambiente de simulação Unity ofereceu a oportunidade de avaliar e aprimorar o desempenho do sistema em diferentes cenários, contribuindo assim para qualificar as discussões na área de controle de tráfego aéreo autônomo. Pois de acordo com a Aeroflap [1], em 2023 , cerca de 570 aviões decolam ou pousam no Aeroporto Internacional de Guarulhos (São Paulo, Brasil) diariamente, com um fluxo de uma aeronave a cada 1 minuto. . Além disso, este trabalho teve como base a pesquisa realizada por Gabriel Dal Forno Mello, que integrou, via *socket*, Jason com Unity.

B. Objetivos

Projetar, implementar e avaliar um ambiente de simulação via a teoria de Sistemas Multiagentes que garanta animação gráfica e o monitoramento do comportamento inteligente de aeronaves não tripuladas em um cenário de aeroporto com uma pista de pouso e decolagem, *n* aviões sobrevoando, com uma fila de espera, o aeroporto e aviões prontos para decolagem.

Os objetivos específicos foram: pesquisar, compilar e escrever sobre Sistemas Multiagentes e Simulação; pesquisar, compilar e escrever sobre metodologias e ferramentas ou ambientes de projeto e implementação de Sistemas Multiagentes; pesquisar, compilar e escrever sobre trabalhos relacionados; projetar e implementar servidor Sistema Multiagentes via JaCaMo e cliente via Unity; definir e aplicar um ambiente de avaliação e teste: metas e planos dos agentes (pista e aviões); valores das variáveis para aviões (altura, nível de combustível e escala); quantidade de aviões a decolar e/ou pousar; comparar os resultados no estudo de caso.

Com isso, avaliou-se se o auto gerenciamento dos agentes, por meio de negociação e planejamento descentralizado, minimiza as filas de espera e/ou promove um fluxo mais eficiente de decolagens e pousos.

Para uma melhor compreensão do projeto, o texto está dividido nas seguintes seções: Referencial Teórico, que aborda áreas, conceitos e boas práticas que foram utilizadas no contexto deste trabalho; Metodologia, que detalha o procedimento de realização do trabalho; Considerações Finais, que apresentam apontamentos importantes destacados ao longo do estudo; e Referências, que listam as obras utilizadas como base para a elaboração do texto.

II. Referencial Teórico

Nesta seção, destacam-se os tópicos centrais relacionados à proposta, bem como as estratégias empregadas durante a construção do projeto. Englobando áreas como Controle de Tráfego Aéreo, Simulação, Sistemas Multiagentes, Inteligência Artificial Distribuída, modelo de comunicação TCP/IP e suas principais ferramentas.

A. Controle de Tráfego Aéreo

O controle de tráfego aéreo é um sistema essencial para garantir a segurança e a eficiência das operações aéreas em todo o mundo. Em sua essência, é responsável por gerenciar o fluxo de aeronaves nos céus, coordenando suas rotas, altitudes e velocidades para evitar colisões e garantir o fluxo ordenado do tráfego aéreo [2].

O controle de tráfego aéreo inclui a comunicação constante entre os controladores e os pilotos. A finalidade do controle de tráfego aéreo é garantir a segurança das operações aéreas, minimizando o risco de colisões entre aeronaves e mantendo o fluxo de tráfego de forma eficiente. Ao monitorar e coordenar o movimento das aeronaves, os controladores de tráfego aéreo ajudam a prevenir acidentes e a manter os céus seguros para todos os que voam [3].

Apesar de existirem recursos tecnológicos e humanos, a gestão do controle de tráfego aéreo lida com um volume muito grande de informações e processos, fazendo com que controladores humanos sejam sobrecarregados em suas atividades.

B. Simulação

A Simulação é um método que envolve a criação de um modelo de um sistema real e a condução de experimentos com esse modelo, com o objetivo de compreender os comportamentos do sistema ou avaliar estratégias para sua operação [4]. Essa prática pode ser vista como uma ferramenta computacional que permite o desenvolvimento, teste e estudo de teorias, visando a compreensão de sistemas reais e o desenvolvimento de sistemas computacionais [5].

A partir dessa definição, torna-se evidente a significativa interseção entre os conceitos de Simulação e Sistemas Multiagentes, demonstrando que ambos podem se beneficiar mutuamente. Os Sistemas Multiagentes proporcionam uma plataforma robusta para a criação de simulações complexas, abrangendo diversos componentes, como sistemas sociais ou sociedades artificiais. Em toda simulação, ocorre a representação de um ambiente com elementos interativos (agentes), os quais respondem a eventos e obedecem a restrições impostas pelo próprio ambiente [5, 4].

1) Simulações de tráfego aéreo: Simulações de tráfego aéreo com veículos aéreos não tripulados (VANTs) utilizam Sistemas Multiagentes para modelar comportamento autônomo e cooperativo em ambiente virtual. Os VANTs, como agentes autônomos, possuem percepção, planejamento, comunicação e tomada de decisão, sendo classificados como Sistemas Sensíveis ao Contexto (*Context-Aware Computing*) [6].

Cada aeronave percebe o ambiente ao redor com sensores, como câmeras, radares e GPS, captando dados de posição, obstáculos e condições climáticas para navegação segura [7]. Com essas informações, a aeronave decide suas ações para alcançar objetivos como evitar colisões e respeitar o tráfego aéreo, guiada por algoritmos de controle e planejamento [7].

Os VANTs se comunicam para coordenar ações e compartilhar dados sobre rotas e condições de voo, por redes *Wi-Fi* ou rádio [8]. Na simulação, cada VANT é um agente individual que interage com o ambiente e com outros agentes segundo regras do modelo [9].

A análise dos resultados avalia o desempenho dos VANTs em diferentes cenários, considerando métricas como tempo de voo, distância, eficiência e segurança, identificando áreas para melhoria no sistema de controle e operação [8].

2) Ferramenta Unity como ambiente de Simulação Computacional: A Unity é uma ferramenta de desenvolvimento de *software* amplamente utilizada para criar aplicativos interativos em 2D e 3D, incluindo jogos, simulações e experiências de realidade virtual e aumentada [10]. No contexto de simulação computacional, a Unity pode ser empregada como um ambiente para modelar e simular sistemas elaborados, incluindo o controle de tráfego aéreo.

Na simulação do controle de tráfego aéreo, a Unity pode ser empregada para criar um ambiente virtual que represente um espaço aéreo realista, incluindo aeroportos, rotas de voo e aeronaves em movimento. Podendo ser utilizado modelos 2D de aeronaves e objetos terrestres, implementação de algoritmos para controle de tráfego aéreo e comunicações entre controladores e pilotos [9], [10].

Em suma, a Unity serve como um ambiente versátil para simulação computacional, oferecendo as ferramentas necessárias para criar simulações precisas e imersivas de sistemas complexos, como o controle de tráfego aéreo.

C. Inteligência Artificial Distribuída

A Inteligência Artificial Distribuída representa um campo avançado da Inteligência Artificial que se concentra na distribuição de processos de tomada de decisão e aprendizado

em sistemas compostos por múltiplos agentes autônomos. A Inteligência Artificial Distribuída se destaca por sua capacidade de resolver problemas complexos em ambientes dinâmicos e distribuídos [11]. Em sua essência, a Inteligência Artificial Distribuída aborda desafios que envolvem sistemas compostos por múltiplos agentes que precisam cooperar ou competir para alcançar objetivos comuns ou individuais. No contexto da simulação de tráfego aéreo, pode ser aplicada para modelar e simular o comportamento de diversos agentes, como aeronaves, controladores de tráfego aéreo e sistemas de navegação [11].

Pode-se citar como exemplo de técnicas para o desenvolvimento da Inteligência Artificial Distribuída [12]: Sistemas Multiagentes: permitem modelar e simular sistemas compostos por diversos agentes autônomos que interagem entre si, como aeronaves, controladores de tráfego aéreo e sistemas de navegação; Aprendizado por Reforço Distribuído: técnicas de aprendizado por reforço distribuído permitindo que os agentes aprendam a melhorar seu desempenho através da interação com o ambiente e com outros agentes, adaptando suas estratégias de acordo com as condições em constante mudança do sistema.

Com isso, a Inteligência Artificial Distribuída, conforme abordada por Russell e Norvig [11], oferece uma abordagem poderosa para resolver problemas complexos em sistemas distribuídos, como o controle de tráfego aéreo, por meio da distribuição de processos de tomada de decisão e aprendizado entre múltiplos agentes autônomos.

D. Sistemas Multiagentes (SMA)

SMA são conjuntos de agentes autônomos que interagem entre si e com o ambiente para atingir metas ou resolver problemas complexos, que seriam difíceis para um sistema monolítico[1]. Cada agente possui crenças e habilidades próprias, agindo de forma autônoma, percebendo, planejando e executando ações, que podem incluir interações com outros agentes e o ambiente [7].

Os SMA são ideais para modelar sistemas dinâmicos, como a proposta deste trabalho, onde a interação entre agentes é crucial para o comportamento global. Essa abordagem é eficaz em cenários de solução desconhecida ou mutável, onde a colaboração entre agentes otimiza a eficiência [14].

As principais características dos agentes incluem [15, 14, 16]: autonomia: agentes tomam decisões próprias com base em percepções e planos; habilidade social: os agentes colaboram para atingir objetivos comuns; adaptabilidade: os SMA se ajustam a mudanças no ambiente; escalabilidade: podem incluir muitos agentes, modelando sistemas complexos; cooperação e competição: agentes cooperam ou competem, conforme as regras do sistema.

[1]Sistema monolítico é um modelo em que o sistema operacional é uma coleção de procedimentos interligados, permitindo interações livres [13].

1) Swarm Intelligence: Inteligência de Enxame, ou Inteligência Coletiva, inspira-se no comportamento coletivo de sistemas naturais, como colônias de formigas, para criar algoritmos e sistemas artificiais. Em SMA, a *Swarm Intelligence* envolve múltiplos agentes seguindo regras simples e interagindo para produzir comportamentos adaptativos [17].

Sistemas baseados nessa inteligência resolvem problemas de busca e adaptação em ambientes dinâmicos e incertos, tornando-se resilientes, flexíveis e cooperativos [18].

2) Interpretador Jason: Jason é um interpretador da linguagem AgentSpeak(L), projetado para modelar e implementar agentes inteligentes, considerando crenças, objetivos e planos para o comportamento cognitivo[2] dos agentes. Esse comportamento é definido por regras de produção, permitindo que agentes tomem decisões, interajam e executem tarefas de forma autônoma [19]. Jason possibilita simular e testar estratégias de agentes em diferentes cenários, proporcionando *insights* e aprimoramentos. Compatível com várias plataformas, Jason é acessível para desenvolvedores e pesquisadores, com documentação ampla e uma comunidade ativa para suporte [19].

3) Framework JaCaMo: É um *framework* de programação Multiagentes que integra Jason para codificação de agentes, Cartago para modelar artefatos de ambiente e Moise para estruturar organizações Multiagentes [20]. Essa combinação cobre os níveis de abstração necessários para desenvolver sistemas multiagentes complexos, aplicáveis a áreas como finanças, gestão de tráfego e sistemas embarcados.

Jason define agentes com base em regras que descrevem percepções e ações (crenças, objetivos e planos) [18]. Cartago permite a criação de artefatos virtuais no ambiente dos agentes, enquanto Moise estrutura diretrizes para a organização e coordenação dos agentes [21].

A integração dessas três dimensões (agente, ambiente e organização) em JaCaMo facilita a criação de sistemas sofisticados e resilientes de forma eficiente [20]. O *framework* permite instalação em máquina local, configuração de extensões e pacotes no Visual Studio Code, utilizando o SDK Java versão 17.

E. Modelo de Comunicação TCP/IP

Segundo Mendes [22], o modelo TCP/IP é um conjunto de normas e protocolos essenciais para a troca de dados em redes de computadores, sendo a base tanto da Internet quanto das redes locais (LANs), com ampla aplicação em áreas como Automação Residencial e Internet das Coisas.

Elementos-chave do modelo incluem o *socket* e a porta lógica. Um *socket* combina um endereço IP e um número de porta, permitindo a comunicação entre sistemas em rede. Cada processo que precisa se comunicar cria um *socket* para enviar e receber dados [23]. Ao receber um pacote, o sistema operacional utiliza o endereço IP de destino e o

[2]Agentes aprendem, raciocinam e interagem de forma natural, ao invés de serem explicitamente programados [7].

número da porta para identificar o processo final, permitindo que múltiplos processos se comuniquem em um dispositivo usando portas diferentes [22]. Assim, o modelo TCP/IP, com sua estrutura modular e flexível, permite comunicação eficiente e confiável entre diversos dispositivos e sistemas em redes de computadores.

F. Trabalhos Correlatos

Nesta seção, são discutidos trabalhos que utilizam simulação e/ou Sistemas Multiagentes, contribuindo para o contexto do estudo realizado.

O estudo de Alexandre Zamberlan [14] investigou o uso de Sistemas Multiagentes em Simulações Computacionais com partículas poliméricas nanoestruturadas, representando fármacos encapsulados em polímeros. As partículas, modeladas como agentes autônomos, interagiam entre si e com o ambiente.

Gabriel Dal Forno [24] implementou um sistema Multiagentes para simulação de tráfego urbano autônomo. A arquitetura utilizou Unity para o simulador e Jason para o sistema Multiagentes, aplicando *Swarm Intelligence* [17]. O sistema funcionou como servidor, com o simulador como cliente, comunicando-se via TCP/IP.

O projeto de Carlos Pantoja [25] desenvolveu um agente inteligente para controlar uma aeronave no simulador X-Plane[3], realizando tarefas como decolagem, navegação e pouso. Usando aprendizado de máquina e controle de sistemas, o agente decolou de um aeroporto fixo e foi programado para decolar de qualquer aeroporto.

Finalmente, os trabalhos analisados utilizaram o interpretador Jason para simulações baseadas em Sistemas Multiagentes. Apenas os estudos de Dal Forno e Zamberlan [24] integraram Unity e C#, com comunicação via TCP/IP, base deste estudo. A pesquisa de Carlos Pantoja [25] também foi relevante, relacionando-se com a simulação aérea, o contexto desta pesquisa.

III. METODOLOGIA

A metodologia deste estudo baseia-se na pesquisa bibliográfica, complementada pela aplicação de um estudo de caso para o desenvolvimento e avaliação da simulação proposta, utilizando a abordagem de Sistemas Multiagentes. Para conduzir a pesquisa, foi adotada a metodologia JaCaMo, conforme descrita por Carlos Pantoja [21], juntamente com a técnica Kanban para o gerenciamento das atividades. Com atividades separadas em datas e separadas em categorias.

Para desenvolver o sistema Multiagentes (servidor da simulação), optou-se por utilizar o interpretador Jason, empregando as linguagens AgentSpeak(L) e Java, fundamentados na metodologia JaCaMo. Para o ambiente de simulação (cliente), foi selecionado o software Unity, utilizando C# como linguagem de programação. Esta escolha foi motivada pela variedade de classes disponíveis no Unity, que são úteis na criação de ambientes simulados, como colisores, vetores de movimento e facilidades para implementação de paralelismo. Por fim, foi decidido usar *sockets* para comunicação, por meio do protocolo TCP, entre a aplicação de simulação e a aplicação do sistema Multiagentes, devido à sua fácil implementação em ambas as linguagens utilizadas.

Para avaliar a proposta, um conjunto de cenários (com aviões, pistas e informações sobre o voo) foram produzidos para que os agentes gerem comportamentos emergentes e úteis em situações inesperadas.

A. Estudo de Caso

O estudo de caso é uma abordagem empírica que permite aproximar a teoria aprendida com a prática, avaliando aplicabilidade e eficácia em um contexto real [27].

Para esta pesquisa é importante definir e aplicar um ambiente de avaliação e teste que trate das metas e planos dos agentes (pista e aviões); dos valores das variáveis para aviões (altura, nível de combustível e se o aeroporto é uma escala desse avião em voo); e da quantidade de aviões a decolar e/ou pousar, já dos valores do agente pista (se possuem aviões e quais são suas posições). Com isso, buscou-se avaliar se o auto gerenciamento dos agentes, por meio de negociação e planejamento descentralizado, minimiza as filas de espera e/ou promove um fluxo mais eficiente de decolagens e pousos.

A avaliação do funcionamento dos agentes foi realizada por meio dos cenários definidos. Cada cenário é testado com diferentes valores para as variáveis mapeadas, que geram volumes de tráfego aéreo, visando entender o sistema.

A metodologia JaCaMo, como já apresentado, integra três plataformas: Jason, CArtAgO e Moise. Cada uma dessas plataformas lida com aspectos diferentes de um sistema Multiagentes: i) Jason trata da programação de agentes inteligentes usando a linguagem AgentSpeak; ii) CArtAgO fornece suporte para a criação e manipulação de artefatos, que são recursos compartilhados pelos agentes; Moise é utilizado para definir e gerenciar organizações Multiagentes, especificando estruturas sociais, papéis e normas.

Na geração da modelagem desse sistema Multiagentes, há alguns passos a seguir:

- Identificação dos Agentes e suas características
  - Agente Avião com propriedades como altitude, nível de combustível e se tem escala naquele aeroporto. Comportamentos como decolar, voar, pousar;
  - Agente Pista com propriedade como fila de aviões prontos para decolar e gerenciar o fluxo de aviões, tanto para decolagem quanto para pouso;
- Definição de Artefatos (usados pelos agentes para interagir com o ambiente)
  - Artefato Fila de Decolagem que gerencia a fila de aviões esperando para decolar;

[3]Simulador de voo realista com modelo baseado em forças aplicadas em diferentes áreas da aeronave [26].

  - Artefato Fila de Pouso que gerencia a fila de aviões esperando para pousar.
- Estrutura Organizacional (Moise)
  - Papéis: controlador de pouso (gerenciado pela Pista); controlador de decolagem (gerenciado pela Pista); piloto (papel assumido pelos agentes avião);
  - Normas: aviões devem negociar entre si e receber autorização para decolar ou pousar através das Pistas.

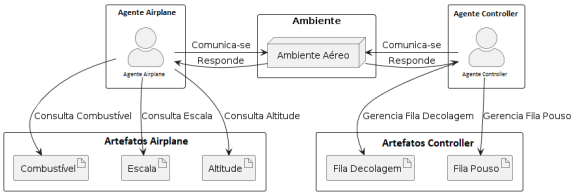

Figura 1. Diagrama de visão geral do sistema Multiagentes [28].

Com base nas definições discutidas, o diagrama da Figura 1 fornece uma visão geral do sistema Multiagentes usando a metodologia JaCaMo, mostrando como os agentes interagem com os artefatos para realizar o controle descentralizado do tráfego aéreo.

B. Organização do Projeto

A simulação do Sistema Multiagentes depende de diversos elementos. A Inteligência Artificial Distribuída é responsável por perceber, planejar e comunicar-se, resultando no Comportamento Emergente, em que os padrões de tráfego preestabelecidos são seguidos e adaptados a possíveis mudanças no ambiente de simulação. Essa interação é viabilizada pelo *framework* JaCaMo, que se comunica por meio do protocolo TCP/IP com o ambiente de simulação gráfica. As simulações gráficas, ao receberem comunicação do JaCaMo por meio da plataforma Unity, podem gerenciar graficamente esse ambiente. Isso inclui a exibição de aeronaves com rotas planejadas e detecção de colisões, torres de controle e pistas de pouso e decolagem.

C. Componentes do Sistema

Conforme demonstrado na Figura 2, a arquitetura adotada pelo projeto é de natureza cliente-servidor. O servidor corresponde à implementação do *framework* JaCaMo, encarregado de acionar ações por meio do interpretador Jason para a linguagem AgentSpeak(L) conforme as circunstâncias de cada agente. Desse modo, o sistema de controle de tráfego aéreo é capaz de gerenciar e coordenar o fluxo das aeronaves.

Por sua vez, o cliente assume a responsabilidade de receber tais ações provenientes do servidor e realizar simulações gráficas por meio da plataforma Unity. Em outras palavras, toda a parte de animações do tráfego aéreo, bem como os modelos de aeronaves e pistas, é renderizada por essa plataforma.

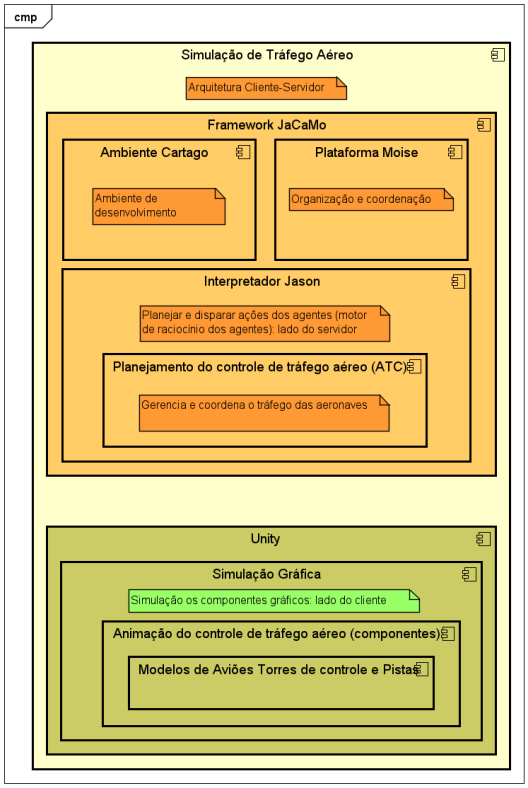

Figura 2. Diagrama de componentes [28].

D. Comunicação

Decidiu-se o uso de *socket* para comunicação, via protocolo TCP (tanto do servidor ao simulador, quanto do simulador ao servidor), entre a aplicação de simulação e a aplicação do Sistema Multiagentes, devido à sua implementação fácil em ambas as linguagens empregadas, além de interconectar dois sistemas heterogênos (servidor em Java, com AgentSpeak(L) e Jason; cliente em C#, com Unity). A Figura 3 ilustra a dinâmica entre o simulador e o Sistema Multiagentes em relação à comunicação entre as tecnologias.

Com o uso do sistema de comunicação baseado em *socket*, onde o Sistema Multiagentes abriga o servidor e o ambiente de simulação abrange o cliente, a troca de informações entre os dois lados é realizada através de *strings* no formato JSON[4]. Esse formato foi escolhido devido à disponibilidade de bibliotecas de codificação e decodificação em ambas as linguagens utilizadas. As informações enviadas do cliente para o servidor contêm dados relativos aos agentes e outros objetos. Enquanto as instruções destinadas às aeronaves da simulação, definidas pelos agentes do Sistema Multiagentes, são transmitidas do servidor para o cliente.

O protocolo TCP foi selecionado para a implementação,

[4]É um acrônimo de *JavaScript Object Notation*, um formato compacto, de padrão aberto independente, de troca de dados simples e rápida entre sistemas [29]

pois a perda de uma instrução/ação do Sistema Multiagentes poderia causar falta de sincronia entre os dois lados e resultar em erros que poderiam levar a colisões ou acidentes na simulação.

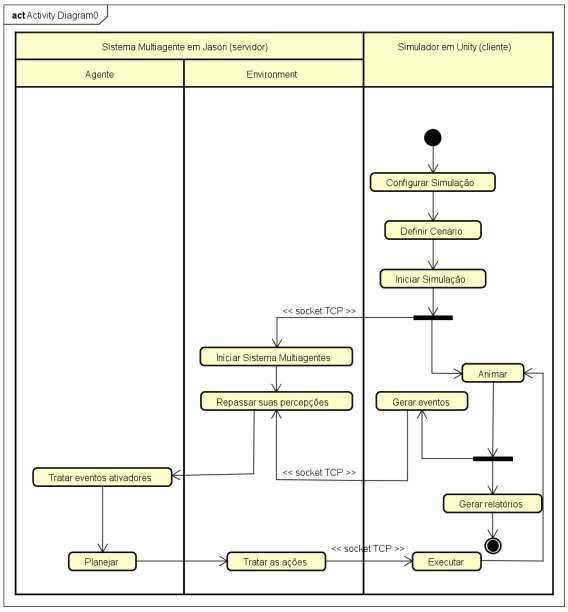

Figura 3. Diagrama de atividades com a ideia geral de funcionamento e integração do simulador [28].

Com isso, através deste sistema de comunicação largamente utilizado e de fácil implementação, garantiu-se a sincronização adequada entre os dois sistemas, minimizando assim o risco de erros e garantindo uma interação fluida e segura durante a execução da simulação.

E. Resultados e Discussões

Foram desenvolvidos três cenários para a realização de testes com o SMA. O primeiro cenário foi projetado para testar o desempenho do sistema com um número limitado de aviões (6 aeronaves), das quais duas precisavam realizar a decolagem, enquanto as demais simularam o processo de pouso. Para garantir a consistência na avaliação, os níveis de combustível foram ajustados para representar situações de risco controlado (aviões com combustível mínimo), conforme ilustrado na Figura 4. O segundo teste/cenário foi realizado com todos os aviões possuindo escala, ou seja, precisavam decolar e com a quantidade de combustível idêntica do primeiro cenário (Figura 5). O terceiro teste representou um cenário totalmente aleatório em relação ao número de aviões e suas características de combustível e escala, conforme apresentado na Figura 6. Registra-se, contudo, que os índices de combustível não aparecem na animação, somente nas informações de agentes exibidas na verbalização da simulação do SMA (Figura 7).

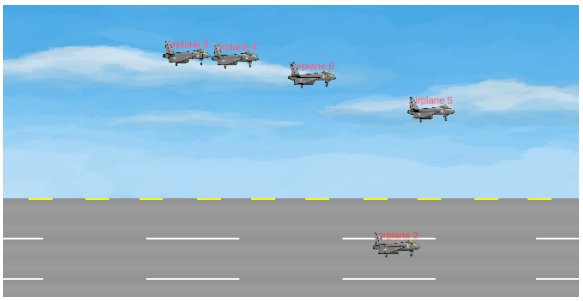

Figura 4. Cenário 1 de testes no Unity [30].

Nos três cenários de teste, o Sistema Multiagente executou todos os planos conforme o esperado, sem apresentar problemas na comunicação entre os agentes/aviões ou entre o servidor e o cliente. As trocas de mensagens entre os agentes, mediadas pelo *controller*, ocorreram de forma sincronizada graças à definição de ciclos temporais específicos, denominados como: *tempo1*, *tempo2*, e etc. Esses ciclos, executados a cada 3 segundos, representavam 1 minuto em uma escala de tempo ajustada para a simulação. Nos cenários 1 e 2, foi possível realizar o pouso de 3 aviões em um tempo equivalente a 1 minuto, o que representa um desempenho superior ao esperado em cenários tradicionais de tráfego aéreo. A capacidade de gerenciar múltiplos pousos em um curto intervalo indica que o sistema conseguiu otimizar o fluxo de tráfego, reduzindo o tempo de espera na pista e maximizando a eficiência da operação. Na Figura 7, é possível observar a saída textual da simulação durante os dois primeiros ciclos de execução. No cenário 3, onde o número de aviões e suas características eram aleatórios, a visualização gráfica foi prejudicada pela complexidade e pela falta de clareza nos dados apresentados na interface. Embora a saída textual tenha fornecido informações cruciais para entender a execução do sistema, a interface gráfica não conseguiu acompanhar o volume e a dinâmica da simulação, indicando que o sistema pode precisar de uma otimização na representação visual de cenários mais complexos conforme exibido na Figura 6.

Figura 5. Cenário 2 de testes no Unity

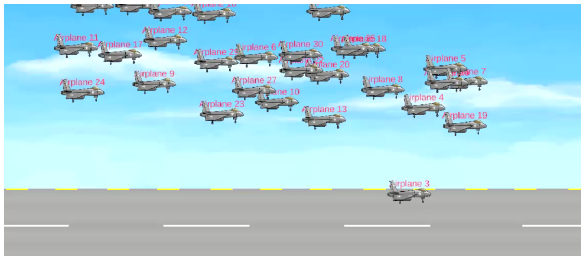

Figura 6. Cenário 3 de testes no Unity [31, 32].

```
[controller] Iniciando operacoes sem aguardar conexao da simulacao.
[controller] Iniciando negociacoes no tempo 1
[controller] CFP enviado para avioes no tempo 1
[controller] Recebida proposta de plane3 com gasolina: 2 no tempo 1
[controller] Recebida proposta de plane1 com gasolina: 1 no tempo 1
[controller] Recebida proposta de plane2 com gasolina: 0 no tempo 1
[controller] Aguardand vencedor para o tempo 1
[controller] Aviao selecionado para pousar: plane2 com gasolina: 0 no tempo 1
[controller] plane2 recebeu autorizacao para pousar no tempo 1
[controller] Aguardando mais pousos antes de iniciar a decolagem.
[controller] AviÃµes no ar no tempo TempoAtual<no-value>: []
[controller] Atualizacao enviada para a simulacao: Tempo=2, PousosRestantes=5, AvioesPousados=["plane2"], AvioesParaDecolar=[], AvioesNoAr=[]
[controller] Iniciando negociacoes no tempo 2
[controller] CFP enviado para avioes no tempo 2
[controller] Recebida proposta de plane4 com gasolina: 3 no tempo 2
[controller] Recebida proposta de plane3 com gasolina: 2 no tempo 2
[controller] Recebida proposta de plane5 com gasolina: 1 no tempo 2
[controller] Recebida proposta de plane1 com gasolina: 1 no tempo 2
[controller] Aguardand vencedor para o tempo 2
[controller] plane2 esta aguardando autorizacao para decolar.
[controller] Aviao selecionado para pousar: plane1 com gasolina: 1 no tempo 2
[controller] plane1 recebeu autorizacao para pousar no tempo 2
[controller] Autorizado decolagem para plane2
[controller] Aguardando mais pousos antes de iniciar a decolagem.
```

Figura 7. Saída verbal dos planos no cenário 1 de testes [28, 33]

A arquitetura do SMA segue uma abordagem ponto a ponto, permitindo que os aviões/agentes interajam diretamente, o que se alinha com o objetivo de promover um controle descentralizado. No entanto, a negociação entre os agentes, mediada pelo agente controller.asl, mostrou-se competente para otimizar as decisões de pouso e decolagem, mas também revelou a necessidade de ajustes para gerenciar cenários mais dinâmicos e com maior volume de tráfego. Esse agente permite que os agentes do SMA troquem mensagens e gerenciem suas demandas. No *Listing* 8, observa-se o comportamento dessas funções, nas quais ocorrem negociações e comparações para identificar o avião mais adequado para pouso (linhas 2, 4 e 11), organizadas conforme um processo de análise de prioridades. Já no arquivo airplane.asl, conforme ilustrado no *Listing* 6, ocorrem as chamadas dessas funções utilizando o protocolo *Call for Proposal*s (CFP)[5] das linhas 1 a 6 é onde recebido do controller.asl a proposta, e das linhas 8 a 19 vem a resposta das propostas do vencedor (quem irá pousar). Dessa forma, os agentes seguem as filas de pouso e decolagem de maneira estruturada, otimizando a eficiência das operações.

Uma das situações em que as particularidades da ferramenta Unity foram benéficas foi no desenvolvimento do mecanismo de detecção de obstáculos ao redor de cada agente. A estrutura de posicionamento e detecção de sobreposição do Unity facilitou a análise dos objetos ao redor de cada agente, permitindo identificar facilmente quais estavam próximos, na pista ou no céu. Outro aspecto facilitado pela ferramenta foi a integração multiplataforma, realizada por meio de mensagens JSON, novamente utilizando métodos nativos da plataforma. No entanto, no código referente ao SMA, foi necessária a utilização de uma biblioteca externa denominada ORG JSON [35] para a leitura e processamento das informações.

Por outro lado, a comunicação via *socket* não é nativamente suportada pela plataforma Unity de forma que atenda aos propósitos deste trabalho. Para suprir essa necessidade, utilizou-se a biblioteca *System.Net.Sockets* da linguagem C# [36]. Já no SMA, foi possível utilizar ferramentas nativas da linguagem Java para estabelecer a comunicação, diferentemente do trabalho de Gabriel Dal Forno [24], que implementou uma interface de comunicação byte a byte. Esse processo de comunicação é ilustrado tanto na função de recebimento de informações da simulação (linha 9), apresentada no *Listing* 1, quanto na função de envio de dados (linha 45), conforme *Listing* 3. A utilização de *socket* no lado do servidor, conforme exibido no *Listing* 2, na linha 4.

```
1  private void listenForUnityMessages() {
2      new Thread(() -> {
3          String line;
4          try {
5              while ((line = in.readLine()) != null) {
6                  JSONObject receivedJson = new JSONObject(line);
7              }
8          } catch (IOException e) {
9              System.err.println("Erro ao ler dados da Unity: " + e.getMessage
                   ());
10         }
11     }).start();
12 }
```

Listing 1. Função do arquivo Communicator.java, para ouvir mensagens da Unity [37].

```
1      private void startServer() {
2          new Thread(() -> {
3              try {
4                  serverSocket = new ServerSocket(12345);
5                  System.out.println("Servidor iniciado na porta 12345.
                       Aguardando conexes...");
6
7                  clientSocket = serverSocket.accept();
8                  System.out.println("Conexo estabelecida com Unity.");
9
10                 OutputStream outputStream = clientSocket.getOutputStream();
11                 out = new PrintWriter(outputStream, true);
12
13                 InputStreamReader inputStream = new InputStreamReader(
                           clientSocket.getInputStream(), StandardCharsets.UTF_8);
14                 in = new BufferedReader(inputStream);
15
16                 listenForUnityMessages();
17
18             } catch (IOException e) {
19                 System.err.println("Erro ao iniciar o servidor: " + e.
                       getMessage());
20             }
21         }).start();
22     }
```

Listing 2. Função do arquivo Communicator.java para iniciar o servidor SMA [37].

```
1  @OPERATION
2      void sendUpdateToSimulation(int tempoAtual, int pousosRestantes, Object
           [] avioesPousadosArray, Object[] avioesParaDecolarArray, Object
           [] avioesNoArArray) {
3          try {
4              String content = new String(Files.readAllBytes(Paths.get("src/env
                   /planesData.json")), StandardCharsets.UTF_8);
5              JSONObject jsonObj = new JSONObject(content);
6              JSONArray planesArray = jsonObj.getJSONArray("planes");
7
8              JSONArray avioesNoAr = new JSONArray();
9              for (int i = 0; i < planesArray.length(); i++) {
10                 JSONObject planeObj = planesArray.getJSONObject(i);
11                 int tempoChegada = planeObj.getInt("tempo");
12
13                 if (tempoChegada <= tempoAtual) {
14                     String planeId = planeObj.getString("id");
15                     if (!arrayContains(avioesPousadosArray, planeId)) {
16                         avioesNoAr.put(planeId);
17                     }
```

[5]O protocolo CFP no JaCaMo permite que um agente solicite propostas de outros agentes para realizar uma tarefa específica. Os agentes respondem com uma proposta ou recusam, e o agente solicitante escolhe a mais adequada [34].

```
18              }
19          }
20
21          JSONArray avioesPousados = new JSONArray();
22          for (Object o : avioesPousadosArray) {
23              avioesPousados.put(o.toString());
24          }
25
26          JSONArray avioesParaDecolar = new JSONArray();
27          for (Object o : avioesParaDecolarArray) {
28              avioesParaDecolar.put(o.toString());
29          }
30
31          JSONObject json = new JSONObject();
32          json.put("tempo_atual", tempoAtual);
33          json.put("pousos_restantes", pousosRestantes);
34          json.put("avioes_pousados", avioesPousados);
35          json.put("avioes_para_decolar", avioesParaDecolar);
36          json.put("avioes_no_ar", avioesNoAr);
37
38          writeJsonToFile(json);
39
40          if (out != null) {
41              out.println(json.toString());
42              out.flush();
43          }
44
45          signal("dataSent", json);
46
47      } catch (IOException e) {
48          failed("Erro ao carregar o arquivo planesData.json: " + e.
                getMessage());
49      } catch (Exception e) {
50          failed("Erro ao processar dados: " + e.getMessage());
51      }
52  }
```

Listing 3. Trecho de código do arquivo Communicator.java, enviando os dados vindos do arquivo controller.asl para a simulação [37].

```
1  +!enviar_atualizacao_simulacao <-
2      ?avioes_pousados(AvioesPousados);
3      ?avioes_para_decolar(AvioesDecolagem);
4      ?tempo_atual(T);
5      ?pousos_restantes(N);
6      !calcular_avioes_no_ar;
7      ?avioes_no_ar(AvioesNoAr);
8      !converter_lista_para_strings(AvioesPousados, AvioesPousadosStr);
9      !converter_lista_para_strings(AvioesDecolagem, AvioesDecolagemStr);
10     !converter_lista_para_strings(AvioesNoAr, AvioesNoArStr);
11     sendUpdateToSimulation(T, N, AvioesPousadosStr, AvioesDecolagemStr,
           AvioesNoArStr);
12     .println("Atualizacao enviada para a simulacao: Tempo=", T, ",
           PousosRestantes=", N, ", AvioesPousados=", AvioesPousadosStr, ",
            AvioesParaDecolar=", AvioesDecolagemStr, ", AvioesNoAr=",
           AvioesNoArStr).
```

Listing 4. Trecho de código do arquivo controller.asl, enviando os dados do sistema para o .java [37].

Na função *sendUpdateToSimulation* (*Listing* 3), das linhas 31 a 36 ocorrem a recepção dos eventos enviados pelo arquivo *controller.asl* (*Listing* 4). Adicionalmente, a função *listenForUnityMessages* (*Listing* 1) realiza a recepção do JSON proveniente do simulador, seguido da conversão necessária para o envio aos agentes do SMA, conforme ilustrado na linha 9.

Após geradas as percepções e enviadas aos agentes, tornou-se necessária a construção dos planos que processam essas percepções, ou seja, que realizam o raciocínio sobre os eventos provenientes do simulador e devolvem ações às aeronaves. No *Listing* 7, na linha 7, observa-se a recepção dos dados, enquanto das linhas 8 a 13 os dados são definidos para cada agente. No *Listing* 8 é apresentada a comparação e negociação entre os aviões. Já no *Listing* 6, das linhas 9 a 13 visualiza-se a recepção da aprovação para pouso. No *Listing* 9, na linha 5, é enviado um *broadcast* para todos os agentes com a mensagem de solicitação de pouso, e na linha 13 aguarda-se a confirmação de todos para dar continuidade à simulação. Assim, esses códigos apresentam os planos de comunicação entre os agentes para negociação baseada no nível de combustível, bem como o controle de aviões com escala que precisam pousar e decolar.

```
1  +!notificar_avioes(T) <-
2      .broadcast(cfp, ["quem_quer_pousar", T]); //tempo atual junto com o CFP
3      .println("CFP enviado para avioes no tempo ", T).
4
5  +!aguardar_todas_propostas(T) <-
6      .println("Aguardando vencedor para o tempo ", T);
7      .wait(2000).
```

Listing 5. Trecho de código do arquivo controller.asl iniciando negociação de pouso [37].

```
1  +!kqml_received(controller, cfp, ["quem_quer_pousar", T], MsgID) <-
2      +pending_cfp(MsgID);
3      .println("Mensagem CFP recebida de ", controller, " e armazenada.");
4      -proposta_enviada(false);
5      .println("Resetando proposta_enviada para false apos receber CFP.");
6      !processar_mensagens_pendentes.
7
8  +!kqml_received(controller, tell, pouso_aprovado, _) <-
9      .my_name(AgentName);
10     .println(AgentName, " esta iniciando o pouso");
11     .wait(3000);
12     .println(AgentName, " pousou com sucesso.");
13     -+pousado(true);
14     if (escala(1)) {
15         !esperar_para_decolar; //se escala = 1, ele precisa decolar depois
16     } else {
17         .println(AgentName, " nao possui escala. Encerrando agente.");
18         .kill_agent(AgentName); //termina o agente
19     }.
20     .println(AgentName, " nao enviara mais propostas.").
```

Listing 6. Trecho de código do arquivo airplane.asl, enviando solicitação de pouso [37].

```
1  +!obterDados <-
2      .my_name(AgentName);
3      .println("Agente: ", AgentName);
4      .println("Obtendo dados do artefato...");
5      getData(AgentName, Dados);
6      .println("Dados recebidos: ", Dados);
7      Dados = dados(Id, Altitude, Posicao, Escala, Gasolina, Tempo);
8      +id(Id);
9      +altitude(Altitude);
10     +posicao(Posicao);
11     +escala(Escala);
12     +gasolina(Gasolina);
13     +tempo_chegada(Tempo);
14     .println("Crencas atualizadas: id(", Id, "), altitude(", Altitude, "),
           posicao(", Posicao, "), escala(", Escala, "), gasolina(", Gasolina
           , "), tempo_chegada(", Tempo, ").");
15     .wait(4000);
16     !aguardar_tempo_chegada.
```

Listing 7. Trecho de código do arquivo controller.asl, obtendo os dados vindos do .java [37].

```
1  +!encontrar_melhor_aviao(T) : proposta(MelhorAviao, MenorC, T) <-
2      !comparar_propostas(MelhorAviao, MenorC, T).
3
4  +!encontrar_melhor_aviao(T) : not proposta(_, _, T) <-
5      .println("Nenhuma proposta recebida no tempo ", T);
6      !avancar_tempo.
7
8  +!comparar_propostas(AtualAviao, AtualC, T) : proposta(Aviao, C, T) & C <
       AtualC <-
9      !comparar_propostas(Aviao, C, T).
10
11 +!comparar_propostas(MelhorAviao, MenorC, T) : not (proposta(_, C, T) & C <
       MenorC) <-
12     +melhor_aviao(MelhorAviao);
13     .println("Aviao selecionado para pousar: ", MelhorAviao, " com gasolina:
           ", MenorC, " no tempo ", T).
```

Listing 8. Trecho de código do arquivo controller.asl, encontrando melhor avião para pousar [37].

```
1  +!start <-
2      joinWorkspace(w);
3      lookupArtifact(simulationInterface, SimID);
4      focus(SimID);
5      .broadcast(tell, tempo_atual(1));
6      !iniciar_pouso.
7
8  +!iniciar_pouso : tempo_atual(T) & pousos_restantes(N) <-
9      if (N > 0) {
10         .println("Iniciando negociacoes no tempo ", T);
11         !notificar_avioes(T);
12         .wait(4000);
13         !aguardar_todas_propostas(T);
14         !encontrar_melhor_aviao(T);
15         ?melhor_aviao(Aviao);
16         .send(Aviao, tell, pouso_aprovado);
17         .println(Aviao, " recebeu autorizacao para pousar no tempo ", T);
18         !atualizar_pousos_restantes(Aviao);
19         .wait(3000);
20         !verificar_decolagem;
21     } else {
22         .println("Todos os pousos foram realizados.");
23         !verificar_decolagem;
24     }.
```

```
25
26  +!kqml_received(Aviao, tell, decolagem_solicitada, _) <-
27      ?avioes_para_decolar(ListaDecolagem);
28      ListaNova = [Aviao | ListaDecolagem];
29      -avioes_para_decolar(ListaDecolagem);
30      +avioes_para_decolar(ListaNova);
31      .println(Aviao, " esta aguardando autorizacao para decolar.").
```

Listing 9. Trecho de código do arquivo controller.asl, iniciando a operação de pousar e comunicando o agente que irá realizar o pouso [37].

IV. Conclusões

Este trabalho propôs o desenvolvimento de um sistema de simulação de tráfego aéreo utilizando sistemas multiagentes e inteligência emergente. O sistema demonstrou que, ao aplicar uma abordagem descentralizada com negociação entre os agentes, é possível criar um fluxo de tráfego mais fluídos, especialmente em cenários de alto tráfego, minimizando as filas e otimizando o uso da pista de pouso e decolagem. Esses resultados têm implicações diretas para a implementação de sistemas de controle autônomo em aeroportos reais, podendo contribuir para a redução da sobrecarga nos controladores de tráfego aéreo. Além disso, os agentes do sistema aprenderam a interagir com outros agentes, de forma autônoma, proativa e flexível.

Durante o desenvolvimento, foram identificados vários sistemas, de simulação de tráfego e sistemas multiagentes com inteligência emergente. No entanto, a implementação de uma rede de aviões interconectados diferencia este trabalho dos demais, representando um conceito atual para a aplicação de tecnologias distintas. Para o desenvolvimento do sistema, foi enfatizada a importância do conceito de Inteligência de Enxame, inspirado em colônias de insetos, que demonstra a coordenação de indivíduos para a emergência natural de um sistema inteligente. Esse conceito foi aplicado aos agentes da simulação, já que a rede de aeronaves possui similaridades com a ideia de enxames. Este trabalho contribuiu significativamente para o campo da simulação de tráfego aéreo autônomo ao desenvolver um sistema multiagente completo. Os agentes foram projetados para negociar entre si de maneira satisfatória, priorizando decisões baseadas em parâmetros críticos como combustível e necessidades emergenciais. Este tipo de abordagem descentralizada tem o potencial de transformar a maneira como o tráfego aéreo é gerido, oferecendo soluções inovadoras para aeroportos congestionados e aumentando a segurança e eficiência das operações. As negociações foram implementadas para priorizar aviões com menor combustível e necessidades emergentes, e a simulação visual foi realizada com Unity, onde os agentes são animados em 2D. A comunicação entre o SMA e a simulação foi estabelecida de forma consistente, utilizando TCP/IP para troca de dados entre os sistemas e garantindo a sincronização dos eventos.

Para aprimorar o sistema, várias melhorias podem ser implementadas. A transição para animações em 3D poderia aumentar significativamente o realismo da simulação, proporcionando uma experiência mais imersiva. Além disso, a implementação de detecção de falhas emergenciais e a introdução de algoritmos para lidar com situações de emergência em voo seriam cruciais para tornar a simulação mais robusta. A otimização das animações de pouso e decolagem, incluindo uma abordagem mais suave e detalhada, também pode melhorar a percepção dos usuários sobre o desempenho do sistema. Esses aprimoramentos poderão permitir uma simulação ainda mais rica e uma experiência de uso mais detalhada e imersiva.

Referências

[1] Gabriel Benevides. *Aeroporto de Guarulhos recebeu 41,3 milhões de passageiros em 2023*. Jan. de 2024. URL: https://www.aeroflap.com.br/aeroporto-de-guarulhos-recebeu-413-milhoes-de-passageiros-em-2023/.

[2] Elmira Zohrevandi et al. "Design and evaluation study of visual analytics decision support tools in air traffic control". Em: *Computer Graphics Forum*. Vol. 41. 1. Wiley Online Library. 2022, pp. 230–242.

[3] Alyssa Martin et al. "Air traffic control (ATC) technical training collaboration for the advancement of global harmonization". Em: *Journal of Air Transport Management* 89 (2020), p. 101794.

[4] Averill M. Lei. "Como construir modelos de simulação válidos e confiáveis". Em: *Conferência de Simulação de Inverno 2022 WSC*. IEEE, 2022, pp. 1283–1295.

[5] Adelinde M. Uhrmacher e Danny Weyns. *Multi-Agent Systems: Simulation and application*. Computational analysis, synthesis, and design of dynamic models series. Boca Raton, FL, USA: CRC Press, 2009.

[6] Mir Salim Ul Islam, Ashok Kumar e Yu-Chen Hu. "Context-aware scheduling in Fog computing: A survey, taxonomy, challenges and future directions". Em: *Journal of Network and Computer Applications* 180 (2021), p. 103008.

[7] Yoav Shoham e Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2008.

[8] Wenhui Fan et al. "Multi-Agent Modeling and Simulation in the AI Age". Em: *TSINGHUA SCIENCE AND TECHNOLOGY* (2021). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9409754.

[9] Supachai Vongbunyong et al. "Simulation of autonomous mobile robot system for food delivery in Inpatient ward with unity". Em: *2020 15th International Joint Symposium on Artificial Intelligence and Natural Language Processing (iSAI-NLP)*. IEEE. 2020, pp. 1–6.

[10] Unity. "Shader Graph". Em: (2018). Disponível em <https://unity.com/pt/features/shader-graph>.

[11] Stuart J Russell e Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

[12] Nick R Jennings. “Coordination techniques for distributed artificial intelligence”. Em: (1996).

[13] Sam Newman. *Monolith to microservices: evolutionary patterns to transform your monolith*. O’Reilly Media, 2019.

[14] Alexandre Zamberlan et al. “Multi-Agent Systems, Simulation and Nanotechnology”. Em: *Multi Agent Systems-Strategies and Applications*. IntechOpen, 2020.

[15] Rafael H Bordini e Jomi F Hübner. “A Java-based interpreter for an extended version of AgentSpeak”. Em: *University of Durham, Universidade Regional de Blumenau* 256 (2007).

[16] Jomi Fred Hübner, Rafael Heitor Bordini e Renata Vieira. “Introdução ao desenvolvimento de sistemas multiagentes com inteligência col”. Em: *XII Escola de Informática da SBC* 2 (2004), pp. 51–89.

[17] Eric Bonabeau, Marco Dorigo e Guy Theraulaz. *From Natural to Artificial Swarm Intelligence*. USA: Oxford University Press, Inc., 1999. ISBN: 0195131584.

[18] Rafael H. Bordini, Jomi Fred Hübner e Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using coletiva*. Ed. por Michael Wooldridge. Wiley, 2007.

[19] Jomi F. Hübner e Rafael H. Bordini. *Jason Description*. 2007. URL: https://sourceforge.net/projects/jason/ (acesso em 02/05/2024).

[20] Olivier Boissier et al. *JaCaMo*. Nov. de 2023. URL: https://jacamo-lang.github.io/.

[21] Jomi F. Hubner e Cleber Jorge Amaral. *JaCaMo*. https://jacamo-lang.github.io/doc. Acessado em 18 de novembro de 2024. 2023. URL: https://jacamo-lang.github.io/doc (acesso em 20/03/2024).

[22] Douglas Rocha Mendes. *Redes de computadores: teoria e prática*. Novatec Editora, 2020.

[23] AB Pawar et al. “Efficacy of TCP/IP Over ATM Architecture Using Network Slicing in 5G Environment”. Em: *Smart Data Intelligence: Proceedings of ICSMDI 2022*. Springer, 2022, pp. 79–93.

[24] Gabriel Dal Forno, Rafael Heitor Bordini e Alexandre Zamberlan. *Simulação de tráfego autônomo com Inteligência Coletiva e Emergente*. Pimenta Cultural - Upgrade: jogos, entretenimento e cultura, 2024. URL: https://www.pimentacultural.com/livro/upgrade-jogos-2/ (acesso em 20/05/2024).

[25] Carlos Eduardo Pantoja e Tielle da Silva Alexandre. “Um Agente Inteligente para Simulação de Voo Usando Jason e X-Plane”. Em: *8th Software Agents, Environments and Applications School (WESAAC)* (2014).

[26] X-PLANE 12. *X-Plane | The world’s most advanced flight simulator*. X-PLANE. 2024. URL: https://www.x-plane.com/.

[27] Robert K Yin. *Estudo de Caso-: Planejamento e métodos*. Bookman editora, 2015.

[28] Bernardo Viero. *images-tfg*. https://github.com/bernardoviero/TFG2/tree/main/images-tfg. Acessado em 18 de novembro de 2024. 2024. URL: https://github.com/bernardoviero/TFG2/tree/main/images-tfg (acesso em 18/11/2024).

[29] Douglas CROCKFORD. *Introdução ao JSON*. 2000.

[30] Bernardo Viero. *Airplane - Cenário 1 (animação gráfica)*. Acessado em 18 de novembro de 2024. 2024. URL: https://youtu.be/ezznI5X3z2k (acesso em 18/11/2024).

[31] Bernardo Viero. *Airplane - Cenário 3 (animação gráfica)*. https://www.youtube.com/watch?v=ctESk1XNKN4. Acessado em 18 de novembro de 2024. 2024. URL: https://www.youtube.com/watch?v=ctESk1XNKN4 (acesso em 18/11/2024).

[32] Bernardo Viero. *Airplane - Cenário 3 (sma)*. https://www.youtube.com/watch?v=ctESk1XNKN4. Acessado em 18 de novembro de 2024. 2024. URL: https://youtu.be/HZghZ2J8_Hs (acesso em 18/11/2024).

[33] Bernardo Viero. *Airplane - Cenário 1 (sma)*. https://www.youtube.com/watch?v=ctESk1XNKN4. Acessado em 18 de novembro de 2024. 2024. URL: https://youtu.be/9_9SeDxYa3o (acesso em 18/11/2024).

[34] Jomi Hübner. *JaCaMo - CFP*. https://jason-lang.github.io/jason/tutorials/jason-jade/readme.html. Acessado em 11 de julho de 2024. 2024. URL: https://jason-lang.github.io/jason/tutorials/jason-jade/readme.html (acesso em 11/07/2024).

[35] JSON. *Introducing JSON*. https://www.json.org/json-en.html. Acessado em 11 de junho de 2024. 2024. URL: https://www.json.org/json-en.html (acesso em 11/06/2024).

[36] Microsoft. *Socket Classe C#*. https://learn.microsoft.com/pt-br/dotnet/api/system.net.sockets.socket?view=net-8.0. Acessado em 11 de junho de 2024. 2024. URL: https://learn.microsoft.com/pt-br/dotnet/api/system.net.sockets.socket?view=net-8.0 (acesso em 11/06/2024).

[37] Bernardo Viero. *codes*. https://github.com/bernardoviero/TFG2/tree/main. Acessado em 18 de novembro de 2024. 2024. URL: https://github.com/bernardoviero/TFG2/tree/main (acesso em 18/11/2024).