Simulação de tráfego autônomo com inteligência coletiva e emergente

Gabriel Dal Forno Mello[1], **Alexandre Zamberlan**[1]

[1]Universidade Franciscana (UFN)
Ciência da Computação
Santa Maria – RS

{mello.gabriel,alexz}@ufn.edu.br

Abstract. *This paper presented the design, model and implementation of a multi-agent system based on autonomous traffic, which makes use of swarm intelligence. For this purpose, we study several tools and methodologies for developing simulations using multi-agent systems. Finally, the simulation architecture was based on a simulator (Unity and the C# language) and a multi-agent system (built in the JASON). Thus, the multi-agent system behaved as a server and the simulator as a client, communicating via TCP/IP model protocols. The results were significant with regard to simulation animation, modeling of the multi-agent system, communication between the animation environment and the agents' reasoning environment.*

Resumo. *O trabalho buscou projetar, modelar e implementar um sistema multiagente baseado em tráfego autônomo, que faz uso de swarm intelligence. Para tal, foram estudadas ferramentas e metodologias para o desenvolvimento de simulações utilizando sistemas multiagentes. A arquitetura da simulação foi baseada em um simulador (ambiente Unity e linguagem C#) e um sistema multiagente (ambiente JASON). Dessa forma, o sistema multiagente comportou-se como um servidor e o simulador como um cliente, comunicando-se via protocolos do modelo TCP/IP. Os resultados foram promissores no que se refere à animação da simulação, modelagem do sistema multiagente, comunicação entre ambiente de animação e o ambiente de raciocínio dos agentes.*

1. Introdução

A gestão de tráfego veicular urbano por meio de simulação é amplamente estudada em áreas como Inteligência Artificial, Modelagem e Simulação, Otimização Computacional. Nesta área de estudo, sempre existe a intenção de otimizar tráfego e adquirir um maior fluxo de veículos, seja pelo uso de pedágios, semáforos ou controle de veículos na via por pistas adicionais.

Em geral, simulações de tráfego urbano tentam validar e otimizar comportamentos de semáforos e pontos em que são necessárias pistas adicionais. O objetivo final é que as vias fiquem o mínimo congestionadas possíveis, levando em consideração o número de veículos, seus tamanhos, velocidades de arranque e tempo dos semáforos para sinais verdes e vermelhos. Este trabalho portanto visa pesquisar, implementar e avaliar um sistema multiagente em que os veículos sejam agentes autônomos que colaborem entre si

e os semáforos, a fim de melhor coordenar ações de descolamento, diminuindo o congestionamento das vias. Dessa forma, este trabalho contempla a área de Inteligência Artificial, mais especificamente a subárea de Sistemas Multiagentes (SMA). A teoria de Sistemas Multiagentes tem foco no projeto e na implementação de sistemas autônomos, proativos, adaptáveis (flexíveis) e com habilidade social (comunicação entre agentes). Um dos estudos de SMA é o de Sistemas Inteligentes Emergentes (*Swarm Intelligence*) [Wooldridge 2001], que recebe esse nome devido ao fenômeno visto onde diversas criaturas de comportamentos simples agem em conjunto para realizar ações inteligentes, como ocorre em colônias de formigas. [Hübner and Bordini 2010].

O objetivo geral do trabalho foi projetar, modelar e implementar um sistema de simulação multiagente com *swarm intelligence* capaz de representar os movimentos de veículos autônomos no tráfego urbano. Os objetivos específicos foram: i)estudar simulações de tráfego e projetos já existentes; ii) definir e aprender a utilizar ferramentas de modelagem e simulação; iii) modelar os agentes do sistema; e definir e aprender a utilizar ferramentas de modelagem visual.

Uma simulação de tráfego veicular urbano usando sistemas multiagentes, em geral, apresenta um processo de simulação focado nos semáforos como os gerentes de trânsito. Nessas gestões, os semáforos comunicam-se entre si trocando informações de tempo de cada sinal, quantidade de veículos percebidos na via (em frente ao cruzamento), entre outros. E a partir desses parâmetros traçam planos coletivos ou individuais para que o trânsito flua sem congestionamentos. Uma das técnicas mais conhecidas é a implementação de onda verde, em que os semáforos entram em sincronismo para que uma quantidade de veículos possam atravessar uma via, com inúmeros semáforos/cruzamentos, sem parar no trajeto [Ferreira et al. 2007]. Entretanto, essa solução ainda é incompleta, uma vez que os cruzamentos/semáforos adjacentes muitas vezes não entram no processo de gestão de fluidez de trânsito.

Portanto, este trabalho justifica-se, pois os agentes inteligentes foram também os veículos, e que foram capazes de se auto-organizar (conceito de *swarm intelligence)* para permitir o maior fluxo possível em uma via. Eles também puderam lidar com obstáculos e situações comuns, tais como faixa de pedestre e pedestres. Assim sendo, neste trabalho, diferente dos demais, a gestão ou organização do tráfego não foi coordenada especificamente por semáforos, como se fosse um controle centralizado. Buscou-se descentralizar o controle do tráfego, de forma que eles pudessem se comunicar entre si (veículos com veículos, semáforos com veículos e semáforos com semáforos) e se organizar no trânsito, justificando o uso da teoria de *Swarm Intelligence* no trabalho.

2. Revisão bibliográfica

Nesta seção, são apresentados conceitos e alguns trabalhos relacionados que são relevantes para a compreensão da proposta e objetivo deste trabalho.

2.1. Inteligência Artificial

De acordo com [Chollet 2017], uma definição concisa do campo de Inteligência Artificial (IA) é "o esforço para automatizar tarefas intelectuais geralmente performadas por humanos". Sendo assim, a área da IA engloba uma grande quantia de aplicações, desde

os primeiros programas de solução de xadrez, que lidavam com regras definidas e estritas, até aplicações mais modernas, que lidam com problemas de lógica difusa, tais como classificação de imagens.

Devido a abrangência desse campo de estudo, existem inúmeras maneiras de implementar algo definido como IA. O principal paradigma era focado na criação de um grande número de regras explicitas, que era conhecido como IA Simbólica (*symbolic AI*). No entanto, a partir de 1990, novas soluções passaram a se popularizar, tal como a utilização de aprendizado e o uso de interações entre componentes de um sistema, a fim de alcançar soluções naturalmente emergentes [Chollet 2017].

Em um contexto prático, técnicas de IA tem acarretado avanços em inúmeras áreas não necessariamente relacionadas, e isso se dá graças a sua grande aplicabilidade e flexibilidade. Um exemplo é o trabalho realizado por [Topol 2019], que mostra algumas formas em que a IA auxilia processos no campo da Medicina, como a interpretação rápida e precisa de imagens, facilitando diagnósticos e reduzindo o potencial de erro médico.

2.2. Sistemas multiagentes - SMA

Na área de Inteligência Artificial, há a subárea conhecida como Inteligência Artificial Distribuída, em que problemas são resolvidos de maneira descentralizada por alguns elementos responsáveis por tarefas específicas. Esses elementos são conhecidos como agentes e estão inseridos em sociedades (sistemas), interagindo e colaborando para resolver problemas em comum [Bordini et al. 2007]. Os agentes são modelados e implementados para terem comportamentos autônomos, proativos, adaptáveis e de interação social (comunicação entre agentes) [Hübner and Bordini 2010]. Por estarem inseridos numa sociedade, compartilham o ambiente, objetivos e recursos, por isso a sociedade multiagente é tida com comportamento emergente de inteligência [Zamberlan et al. 2014].

A teoria SMA está presente em muitas aplicações, como simulação computacional, jogos eletrônicos e sistemas autônomos robóticos, por exemplo. Essas aplicações têm em comum componentes ou elementos (software ou hardware) que precisam atuar e perceber o ambiente, sem que sejam controlados por algo ou alguém (autonomia), que executem ações ou tarefas de forma proativa, adaptáveis e flexíveis às situações inesperadas do ambiente. Além disso, precisam estar em constante comunicação entre si, coordenando tarefas ou ações junto ao ambiente.

2.2.1. Swarm Intelligence: inteligência coletiva e emergente

Conforme [Bonabeau et al. 1999], *Swarm Intelligence* refere-se a uma maneira de modelar sistemas multiagentes inteligentes, que se originou da forma em que colônias de insetos interagem. Sendo assim, os principais pontos a serem considerados no desenvolvimento de sistema coletivos são: eficiência, flexibilidade e robustez. A ideia de *Swarm Intelligence* teve início em Autômatos Celulares, sistemas em que agentes buscam ocupar espaços adjacentes e se auto-organizam para formar padrões complexos [Bonabeau et al. 1999].

É possível destacar que *Swarm Intelligence* é um conceito que surge naturalmente em sistemas multiagentes amplamente populados por agentes, com alta interação social

(comunicação), em que esses agentes, em termos computacionais, não registram grandes capacidades de processamento, mas que em conjunto atingem objetivos comuns, gerando inteligência coletiva [Ferreira et al. 2007].

A forma em que os sistemas de inteligência coletiva funcionam, baseia-se em *loops* de *feedback* positivo, ou seja, os agentes devem ser programados de tal maneira que comportamentos individualmente bons também sejam bons para o sistema coletivo, levando a uma auto-coordenação. Outro ponto relevante, deve ser a capacidade de comunicação entre os agentes, que evita que eles tomem ações desnecessárias, assim melhorando a eficiência geral do sistema.

Dentre as justificativas para se usar técnicas de inteligência coletiva, há potencial para simular casos em que situações complexas emergem devido a interação de diferentes agentes. Isso torna tais sistemas adequados para a resolução de problemas dinâmicos, cujas características exatas podem mudar durante a sua execução.

2.2.2. Simulação

Simulação, como descrito por [Shannon and Johannes 1976], é o processo de desenvolver o modelo de um sistema real e conduzir experimentos com tal modelo, para compreender comportamentos ou avaliar estratégias para operação dentro do sistema. Assim, [Uhrmacher and Weyns 2009] determina que a simulação pode ser considerada uma ferramenta computacional para desenvolver, testar e estudar teorias a fim de compreender sistemas reais, e desenvolver sistemas computacionais.

Seguindo essa definição, se conclui que há um grande espaço de intersecção entre os conceitos de simulação e de sistemas multiagentes, em que ambos geram benefícios ao outro. O uso de simulações pode servir como uma maneira de experimentar e desenvolver sistemas multiagentes de forma eficiente e controlada. Por outro lado, os sistemas multiagentes fornecem uma plataforma excelente para a construção de simulações complexas com inúmeras partes envolvidas, tais como sistemas sociais ou sociedades artificiais. Toda simulação ocorre sobre uma perspectiva de ambiente com elementos (objetos ou agentes) que interagem entre si, respondem a eventos e obedecem a restrições impostas pelo ambiente.

O grande espaço compartilhado por essas áreas significa que muitas vezes, quando houve avanços em uma delas, isso foi causado ou levou diretamente a avanços na outra [Uhrmacher and Weyns 2009]. Como exemplo, pode-se citar que quando paradigmas baseados em agentes começaram a ser desenvolvidos, consequentemente levou à criação de simulações de comportamentos animais e humanos de precisão. Dessa maneira, a utilização de sistemas multiagentes para o propósito de uma simulação não é apenas prático, mas também uma conclusão naturalmente encontrada graças aos conceitos apresentados.

2.3. Ferramentas, ambientes e metodologias de construção de Sistemas Multiagentes

Há inúmeras ferramentas ou ambientes de simulação multiagentes, aqui são listadas as apresentadas e discutidas em [Zamberlan 2018]: FLAME, JASON, MASON, Netlogo,

Repast e SWARM. Os ambientes ou ferramentas de especificação e de programação de sistemas multiagentes possuem uma variedade de características e funcionalidades, como por exemplo ambiente de desenvolvimento integrado, linguagem de programação específica ou genérica, sistema operacional, suporte ao usuário (manuais e exemplos), integração com outras bibliotecas, possibilidade de executar o sistema com visualização 3D, propriedade de visualização de cenários.

De acordo com [Bordini and Hübner 2007], o *Java-based interpreter for an extended version of AgentSpeak*, ou JASON, é um interpretador baseado em Java para a linguagem AgentSpeak(L) e suas extensões. A linguagem AgentSpeak(L) mais o interpretador JASON formam um *framework* para o desenvolvimento de agentes no paradigma BDI (*Belief, Desire, Intention*), que é baseado em crenças (informações do ambiente, de outros agentes e de si mesmo), desejos (planos planejados) e intenções (planos em execução, ou seja, com recursos alocados) [Bordini et al. 2007]. Dessa maneira, o JASON serve como uma ferramenta para simular o sistema multiagente desenvolvido em AgentSpeak(L) a um contexto maior [Bordini et al. 2007].

O JASON possui ampla documentação e recursos de suporte, tanto ao AgentSpeak(L), quanto aos pacotes utilizados para a sua implementação. Além disso, existem inúmeros trabalhos na área de simulação via sistemas multiagentes desenvolvidos por meio de seu uso, provando sua flexibilidade e confiabilidade. Os recursos existentes para a ferramenta tornam mais simples o processo de desenvolvimento de qualquer sistema que for fazer uso dessa.

2.3.1. Metodologia PROMETHEUS e *framework* JaCaMo

Uma metodologia para o projeto e a implementação do SMA deve obedecer às orientações da Engenharia de Software Orientada a Agentes (*Agent-Oriented Software Engineering* - AOSE). Assume-se que uma metodologia de projeto e de construção de SMA deve conter um conjunto de métodos, processos e ferramentas para o desenvolvimento de um sistema baseado em agentes, sendo que a modelagem deve contemplar objetivos do sistema, papéis e interações dos seus elementos [Padgham and Winikoff 2004]. A metodologia PROMETHEUS possui as etapas: i) especificação do sistema: descrição dos objetivos e cenários do sistema. Para isso, elenca as principais funcionalidades; ii) projeto arquitetural: definição da Visão Geral do sistema, em que os agentes são organizados em papéis e são descritas suas relações; iii) projeto detalhado: detalha-se, então, todos os agentes e seus papéis, em que são descritos eventos, planos e dados.

Segundo em [Boissier et al. 2021], JaCaMo é um *framework* para programação multiagente e que integra o interpretador JASON (para a programação dos agentes), ambiente Cartago (para modelar artefatos[1] de ambiente de programação) e plataforma Moise (para programar organizações multiagentes). Essa combinação garante uma cobertura de todos os níveis de abstrações que são necessários para o desenvolvimento de sistemas multiagentes sofisticados e/ou complexos.

Um projeto em JaCaMo integra três dimensões: a dos agentes (JASON), a do ambiente (Cartago) e da organização (Moise). Por essa razão que o JaCaMo é ideal para

[1] Um artefato é um elemento que compõe um ambiente de simulação

sistemas complexos. O principal benefício apresentado pelo JaCaMo é a sinergia entre estas diferentes tecnologias, que utilizadas em conjunto são capazes de criar sistemas complexos e robustos muito mais facilmente do que quando sendo utilizadas separadamente.

2.4. Simuladores de Tráfego

Em [Al Barghuthi and Togher 2020], são apresentados dois ambientes de simulação desenvolvidos para tráfego urbano veicular. Um desses ambientes, em forma de *frameworks*, é o *Multi-Agent- Based Traffic and Environment Simulator* (MATES), um ambiente *open-source*, que foi utilizado em planos de extensão na cidade de Okayama, Japão, e permite interpretação visual por meio de uma visão *top-down*, representando a cena em duas dimensões. O segundo ambiente de simulação, também em forma de *framework*, foi chamado de ArchiSim. Esse ambiente foi utilizado para simular intersecções e rótulas com a intenção de propor soluções para diminuir o número de acidentes ocorridos. Ele apresenta uma visão 3D das simulações.

Apesar das potenciais vantagens da utilização de simuladores especificamente desenvolvidos para o tráfego, existe uma preocupação quanto a dificuldade de encontrar documentação e exemplos de seu uso, já que foram encontrados poucos trabalhos associados. Outra preocupação está no potencial nível de liberdade disponibilizada pelos *frameworks*, já que este trabalho busca apresentar inteligência emergente. Dessa forma, é vital que os agentes sejam customizáveis e programáveis.

2.5. Trabalhos relacionados

Nesta seção, busca-se discutir alguns trabalhos relacionados que de alguma forma utilizaram simulação e/ou sistemas multiagentes, e que de alguma maneira trazem pontos relevantes que contextualizem o estudo realizado.

No trabalho apresentado em [Zamberlan et al. 2020], houve destaque para o uso da teoria de Sistemas Multiagentes em simulações computacionais. No caso, as simulações eram em ambientes com partículas poliméricas nanoestruturadas (contendo fármaco e um polímero como invólucro), em que partículas-agentes agiam de forma autônoma, interagindo entre si e com o ambiente. Nesse trabalho, usou-se agentes e sistemas multiagentes com arquitetura reativa e toda a simulação buscava verificar visualmente se as partículas tinham comportamento de aglomeração ou não (devido a carga elétrica de cada e o pH do ambiente). Todo o comportamento do sistema obedecia as regras do sistema de colisão da teoria Browniana de movimento. Outro ponto do trabalho, é que o sistema multiagente proposto garantia que os eventos de restrição e de interação fossem os mais parecidos com a realidade. A metodologia de desenvolvimento do sistema foi *Feature-Driven Development* (FDD), com uso da linguagem de programação JAVA, da biblioteca de simulação multiagente JASON, do pacote de computação científica *algs4* [Sedgewick and Wayne 2011].

O trabalho realizado por [Doniec et al. 2008] apresenta um modelo multiagente para simulação de tráfego urbano, especialmente adequado em lidar com junções na estrada. As simulações realizadas foram testadas e comparadas a tráfego real, assim garantindo sua precisão e fidelidade. Os agentes utilizados no trabalho foram modelados com base na coordenação entre os motoristas e dois tipos de comportamento, antecipatório e

oportunista. Para as simulações, foi utilizado o ArchiSim, uma ferramenta de simulação de tráfego comportamental ainda em desenvolvimento, mas que possui a capacidade de realizar suas simulações em conjunto a um simulador de direção. O comportamento dos agentes foi baseado, primariamente, em regras simples de prioridade em intersecções. No entanto, essas regras podem ser quebradas pelo agente dependendo de seu comportamento e outros fatores, como velocidade e posições de veículos. Graças a essas características, os agentes são cognitivos, já que tomam ações diferentes dependendo de fatores internos e não apenas externos. Finalmente, há uma camada antecipatória aos agentes, que permite que se coordenem a fim de evitar engarrafamentos. O sistema foi validado de diversas maneiras, passando por testes comparativos a outros modelos e a dados reais, após os quais, foram feitas mudanças a fim de reduzir engarrafamentos e oscilações. O sistema também foi testado através de simuladores de direção com rodovias reais, e seus agentes apresentaram comportamentos próximos ao mundo real.

O trabalho de [Vester et al. 2011] apresenta uma abordagem para o desenvolvimento de sistemas multiagentes, incluindo *design*, análise e tecnologias. Mais especificamente, é descrito o processo para simulação de pastoreamento de animais em fazendas, que tanto animais quanto pastoreados (condutores dos animais). Esse sistema foi apresentado e testado durante o *Multi-Agent Programming Contest* 2010. Para a programação dos agentes, foi utilizada a linguagem de especificação e de implementação AgentSpeak(L) interpretada em JASON, que por sua vez é uma biblioteca de simulação de sistemas multiagentes como *plugin* do ambiente de programação Eclipse. O sistema de simulação multiagente possui 4 agentes que são capazes de coletar informações do ambiente e utilizá-los para informar suas decisões, compartilhar informações com outros agentes. Para isso, foram utilizados certos recursos do interpretador JASON (diretivas de comunicação entre agentes, por exemplo, *send()*, *broadcast()*, et.). Os agentes são: animal, pastor, *scout* (batedor) e sabotador. O *scout* observa e aprende o melhor caminho até o destino de entrega dos animais a partir de onde os animais se encontram. Registra-se que os animais se espalham na área de uma fazenda e muitas vezes ficam aglomerados (*cluster*) em pequenos grupos. Outro ponto interessante, por o sistema ter sido desenvolvido em uma competição, também foram projetados agentes para sabotar os agentes adversários, atrapalhando suas metas. A equipe desenvolvedora do sistema concluiu que a utilização do JASON facilitou de maneira significativa o desenvolvimento, graças aos recursos do interpretador. Também concluiu-se que em um projeto futuro, seria melhor prover aos agentes mais autonomia, e deixá-los menos acoplados a papéis ou funções fixas como foi projetado e implementado.

O trabalho realizado por [Liu et al. 2011] apresenta um *framework* multiagente baseado em aprendizado por reforço para simulação de grupos de pedestres (MARL-Ped). O objetivo do trabalho é apresentar comportamentos emergentes adaptados aos requisitos de movimentação de pedestres. A construção dos comportamentos dos agentes foi feita por meio de uma estratégia que utiliza dois modos, o de aprendizado e o de simulação. Durante o modo de aprendizado, os agentes do sistema otimizam seus movimentos e geram uma função que serve como centro de suas tomadas de decisão. Durante o modo de simulação, os agentes são testados e os dados sobre seus movimentos são gravados a fim de serem inseridos em uma *engine* gráfica. O módulo que contempla comportamentos físicos, como movimentação, no MARL-Ped é uma versão da biblioteca de Física *Open Dynamic Engine* (ODE), que modela os cenários com base em forças externas e internas

em relação aos agentes. O módulo responsável pelo aprendizado e desenvolvimento dos agentes foi o algoritmo Sarsa. Os agentes foram testados em alguns cenários diferentes, incluindo cruzamentos em corredores, labirintos e bifurcações. Os resultados encontrados pelo trabalho mostram a resolução dos problemas de navegação no nível estratégico, tático e funcional, e expõem uma alternativa a modelos de simulação de pedestres anteriormente existentes. Finalmente, concluiu-se também que a utilização de aprendizagem por reforço apresentou certas vantagens, mas também dificultou a fase de animação e controle preciso da simulação.

Analisando os trabalhos, destaca-se que o primeiro trabalho [Zamberlan et al. 2020] há um ambiente animado das simulações, diferente do que ocorre no trabalho dois [Doniec et al. 2008], que é difícil visualizar o comportamento dos agentes. Já em relação ao terceiro trabalho, pode-se destacar o uso de uma abordagem para coordenação de grupos, como ocorre na gestão de semáforos, e o uso de uma ferramenta flexível, multiplataforma, pertencente ao paradigma lógico-declarativo[2], integrada à linguagem Java, com consolidação na comunidade científica e uma vasta documentação e exemplos. O destaque do trabalho quatro é justamente a modelagem e a implementação de comportamento emergente, como da teoria de *Swarm Intelligence*.

3. Estudo de caso

Nesta seção, são explicados a metodologia e os passos seguidos para o projeto, implementação e execução do estudo de caso, que busca avaliar o uso de *Swarm Intelligence* no universo de Tráfego Urbano. Na Figura 1, o mapa mental ilustra os conceitos e suas relações no trabalho.

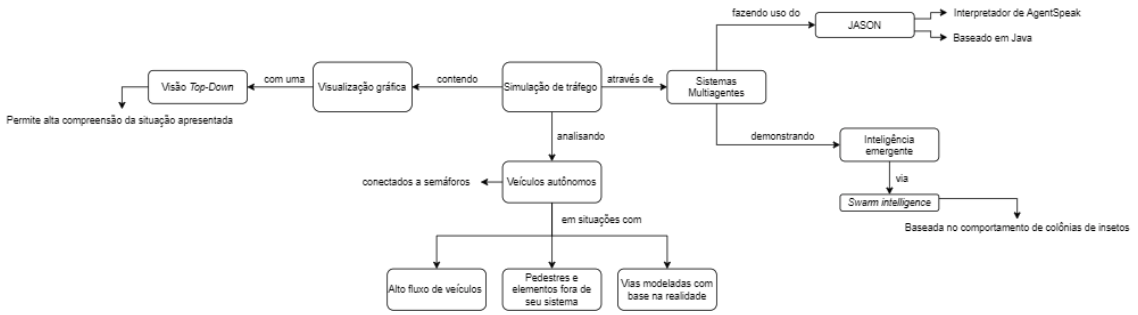


Figura 1. Mapa mental da proposta geral do trabalho.

3.1. Materiais e métodos

Em relação à realização da pesquisa, a metodologia *Scrum* [Silveira et al. 2012] foi utilizada, bem como a técnica *Kanban* para a gestão das atividades assumidas. Os *sprints* foram semanais tendo como referência as funcionalidades mapeadas e inseridas na ferramenta Trello para *Kanban*.

Para o projeto do SMA foi definida a utilização do interpretador JASON, com as linguagens AgentSpeak(L) e Java, a metodologia PROMETHEUS. O código desenvolvido foi armazenado em um repositório na ferramenta GitHub. Para a simulação, foi escolhida a utilização do software de desenvolvimento Unity (no qual a linguagem de

[2]Facilidade para a especificação e para a implementação.

programação usada é C#), devido a existência e suporte a varias classes úteis a criação do ambiente, tais como colisores (*colliders*), vetores de movimento e paralelismo de fácil implementação.

Finalmente, foi definida a utilização de *socket* para comunicação, via protocolo TCP, entre a aplicação de simulação e a aplicação do SMA, por ser facilmente implementável em ambas as linguagens utilizadas.

3.2. Ambiente de simulação

O ambiente de simulação projetado e implementado contém, cruzamentos, ruas, avenidas, semáforos, pedestres, e veículos. A intenção foi gerar um número de cenários que seja suficiente para os agentes desenvolverem comportamentos generalizados e úteis em situações inesperadas ou previamente não vistas. Buscou-se também estudar situações de congestionamento para analisar maneiras de reduzir a incidência dessas situações, e uma otimização geral do tráfego em cidades.

Para a realização dos processos de simulação, como já mencionado, foi usada a plataforma Unity, que permitiu a execução do ambiente em 3D de maneira eficiente e efetiva para os propósitos do trabalho. A plataforma também possui mecanismos de tratamento da Física, assim permitindo que o desenvolvimento fosse focado no controle dos agentes. Além disso, a vasta documentação e quantia de aplicações previamente desenvolvidos na plataforma simplificaram o processo de programação.

Foi definido que a simulação ocorreria em um espaço tridimensional pois isso se demonstrou um diferencial comparado a grande parte dos trabalhos de simulação de tráfego encontrados, que em geral apresentavam visualização bidimensional. Além do ponto mencionado, o ambiente tridimensional também proporciona ao usuário maior liberdade para observar a simulação, aumentando assim a interação com o sistema.

É importante ressaltar que o ambiente do Unity existe apenas como uma parte do sistema, formado também pelo sistema multiagente, que determina as ações a serem tomadas pelos agentes principais da simulação, e o subsistema de comunicação, responsável por coordenar estas partes. Sendo de tal forma, apesar dos veículos (os agentes principais) terem maior parte de sua composição implementada neste subsistema, eles são apenas representações dos agentes no SMA, que são responsáveis por determinar seu comportamento durante o tempo de execução, de maneira qual cada agente do SMA pode ser considerado um ”motorista”do veículo na simulação. A Figura 2 também ilustra a situação descrita.

3.3. Comunicação

O diagrama da Figura 2 ilustra o fluxo dos processos necessários (modelagem funcional do simulador) para a execução da simulação e a relação do usuário com o simulador e o sistema multiagente, tendo início com a definição dos parâmetros de uma simulação por um usuário, até o eventual início dessa. Já a Figura 3 mostra a dinâmica entre o simulador e o SMA no que se refere à comunicação entre as tecnologias.

O sistema de comunicação definido foi baseado em *socket*, onde o SMA contém o servidor, e o ambiente de simulação contém um cliente. A troca de informação entre os dois lados é feita através de *strings* no formato JSON. O formato foi escolhido devido

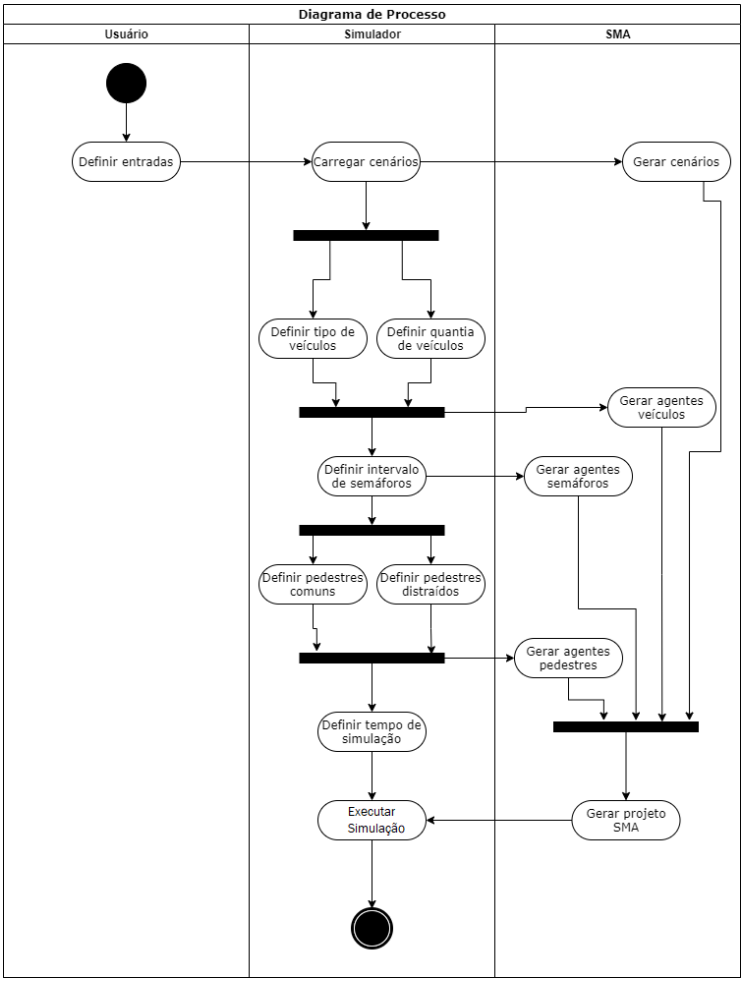


Figura 2. Diagrama de processo do simulador.

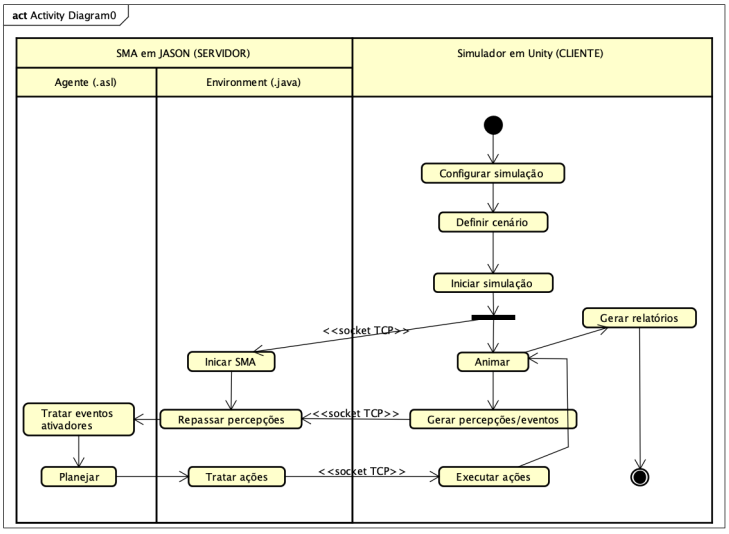


Figura 3. Diagrama de atividades com a ideia geral de funcionamento e integração do simulador.

a disponibilidade de bibliotecas de codificação e decodificação em ambas as linguagens utilizadas. A informação passada do cliente para o servidor contém informações referentes aos agentes e outros objetos. Por outro lado, a informação passada do servidor possui instruções para os veículos da simulação, definidos pelos agentes do SMA.

Definiu-se o protocolo TCP para a implementação. O motivo disto é que a perda de uma instrução do SMA pode causar falta de sincronia entre os dois lados e gerar erros que poderiam resultar em colisões ou acidentes na simulação.

A Figura 4 demonstra a estrutura dos objetos enviados pelo cliente e pelo servidor respectivamente, em que pode ser visto que o cliente envia grande parte das informações, o objeto JSON mostrado é enviado para cada agente presente na simulação. Esse objeto também contém informações sobre os agentes ou obstáculos ao redor do agente primário nos vetores *seen*, representando aqueles diretamente na frente, e *around*, representando aqueles ao redor. O objeto enviado pelo servidor por outro lado, é relativamente esparso, contendo apenas o *id* do agente relevante e o comando a ser executado.

Figura 4. Estrutura das mensagens JSON enviadas

3.4. Sistema Multiagente

Em uma dimensão organizacional, devem existir agentes veículos, pedestres e semáforos. Isso define as 3 diferentes categorias de agentes existentes no sistema. Já na dimensão do ambiente (uso de artefatos), devem existir cruzamentos, vias e faixas de pedestres.

O funcionamento dos agentes deve ser avaliado por meio de três cenários diferentes. Primeiro, um cenário contém uma representação de um cruzamento, para que o desempenho dos agentes possa ser avaliado quando precisam se coordenar em rotas diferentes. Um segundo cenário possui faixa de segurança e passagem de pedestres, para analisar como os agentes lidam com elementos fora de seu sistema. E um terceiro cenário deve analisar como os agentes se comportam em relação a um semáforo.

A Figura 5 mostra a modelagem da metodologia PROMETHEUS representativa do sistema multiagente a ser desenvolvido, contendo os principais agentes (representados em laranja), suas percepções ou crenças (representadas em vermelho), seus planos (representados em verde) e protocolos de comunicação com outros agentes (representados em azul). Especificamente, as percepções representam eventos que envolvem aquele agente, e que, por consequência, disparam um plano (ou seja, uma ação que o agente deve tomar)

ou um protocolo de comunicação, que tem função de enviar alguma informação a outro agente presente no no sistema.

É importante ressaltar que devido as restrições deste trabalho, foi definido que os agentes devem seguir um caminho pré-definido, e o SMA deve controlar sua aceleração e desaceleração. O motivo para isso é evitar a introdução de complexidade desnecessária ao sistema já relativamente complexo, que foi desenvolvido inicialmente como uma maneira de testar a funcionalidade de um sistema distribuído com SMA Jason e ambiente de simulação separados.

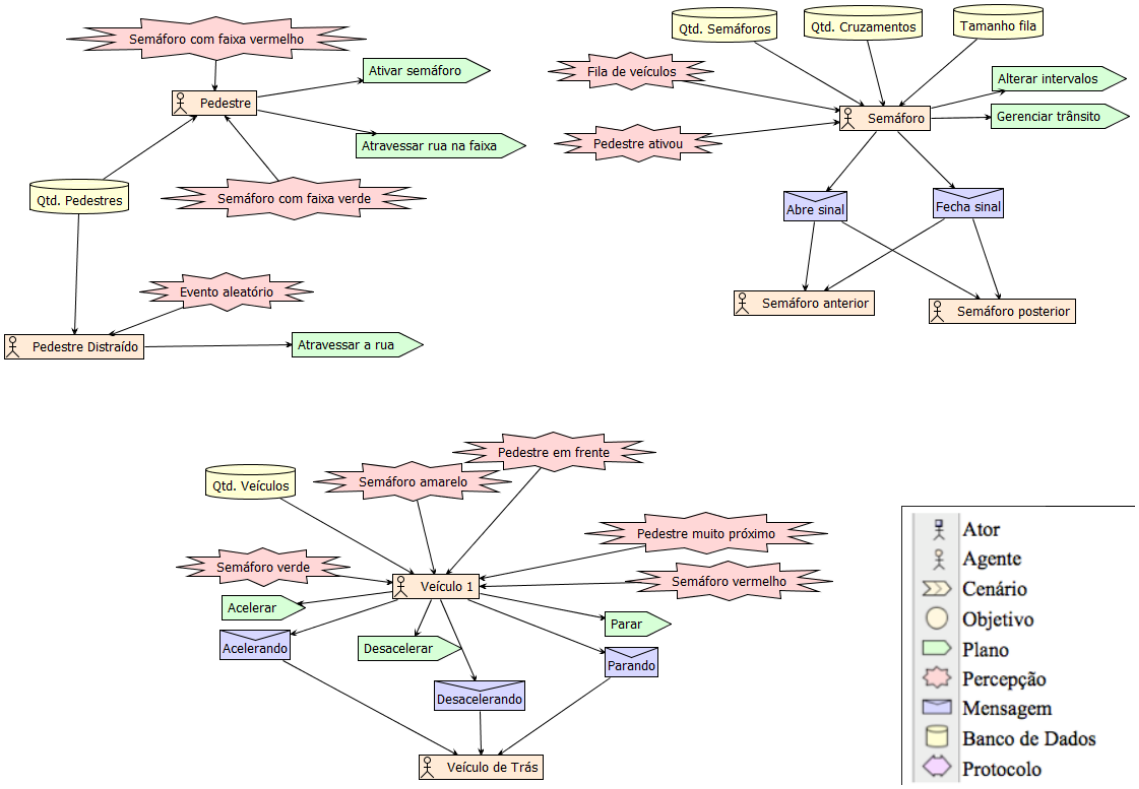


Figura 5. Diagrama de visão geral do sistema multiagente.

4. Resultados e discussões

Na Figura 6, podem ser vistos os três cenários desenvolvidos para realização de testes com o SMA. O primeiro sendo um teste de cruzamento, o segundo um teste de comportamento com semáforos, e o terceiro um teste de resposta a pedestres.

Uma das situações na qual as particularidades da ferramenta Unity ajudaram foi no desenvolvimento do mecanismo de detecção de obstáculos ao redor de cada agente, que foi facilitada devido a estrutura de posicionamento, ângulos e detecção de sobreposição do Unity, que permitiram facilmente analisar quais objetos estavam ao redor de cada agente, e quais estavam diretamente em sua frente.

Outro ponto facilitado pela ferramenta foi a integração multiplataforma, feita através de mensagens JSON, novamente utilizando métodos nativos da plataforma. Por outro lado, no código referente ao SMA, foi necessária a utilização de uma biblioteca externa chamada de GSON para leitura e processamento das informações.

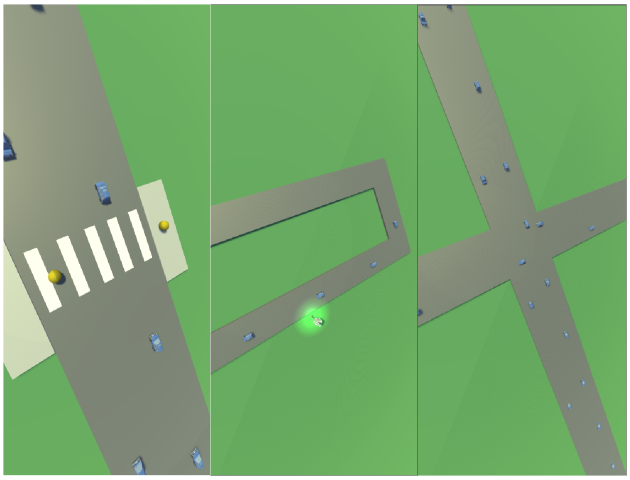


Figura 6. Capturas de tela dos cenários de teste do sistema.

A comunicação através de *socket* por outro lado, não é suportada nativamente pela plataforma Unity da maneira que era necessária para os propósitos do trabalho. Portanto, foi utilizada uma biblioteca da linguagem C# para alcançar este fim. Também foi definida a criação de duas classes, uma de envio e uma de escuta, que facilitou o paralelismo da comunicação, já que no Unity os *scripts* executam paralelamente. O SMA por outro lado, permitiu o uso de ferramentas nativas da linguagem Java para alcançar a comunicação. Na Figura 7, podem ser vistos os códigos para envio de mensagens por *socket* de ambos os lados, que apesar das diferenças de linguagem, são similares em funcionamento.

```
44  public void Send(String message)
45  {
46      try
47      {
48          // Sending
49          int toSendLen = System.Text.Encoding.ASCII.GetByteCount(message);
50          byte[] toSendBytes = System.Text.Encoding.ASCII.GetBytes(message);
51          byte[] toSendLenBytes = System.BitConverter.GetBytes(toSendLen);
52          socket.Send(toSendLenBytes);
53          socket.Send(toSendBytes);
54      }
55      catch (Exception e)
56      {
57          Console.WriteLine(e);
58      }
59  }
60  }
61
```

```
31  public static void sendMessageTCP(Socket s, String msg) {
32      OutputStream os;
33      try {
34          os = s.getOutputStream();
35          byte[] toSendBytes = msg.getBytes();
36          int toSendLen = toSendBytes.length;
37          byte[] toSendLenBytes = new byte[4];
38          toSendLenBytes[0] = (byte)(toSendLen & 0xff);
39          toSendLenBytes[1] = (byte)((toSendLen >> 8) & 0xff);
40          toSendLenBytes[2] = (byte)((toSendLen >> 16) & 0xff);
41          toSendLenBytes[3] = (byte)((toSendLen >> 24) & 0xff);
42          os.write(toSendLenBytes);
43          os.write(toSendBytes);
44
45      } catch (IOException e1) {
46          // TODO Auto-generated catch block
47          e1.printStackTrace();
48      }
49  }
```

Figura 7. Códigos envio de mensagens via *socket*. Lado esquerdo C# e lado direito Java.

Do lado do SMA, mais precisamente no arquivo *.java* do JASON, há toda recepção e conversão de eventos em percepções para os agentes (Figura 8). Neste caso, o código está envolto em uma *Thread* para manter a comunicação não bloqueante com o simulador. Além disso, há a recepção do JSON vindo do simulador (linha 83), execução do *parser* JSON para Orientação a Objetos (linhas 86, 87 e 88) e a conversão em percepções (entre as linhas 90 e 120) a serem enviadas aos agentes do SMA (linhas 102, 120 e como ilustra a caixa em destaque).

Uma vez geradas as percepções e enviadas aos agentes, foi necessário a construção dos planos que tratam dessas percepções, ou seja, raciocinam sobre os eventos vindos do simulador e devolvem ações aos veículos. Na Figura 9 há os planos de eventos externos para lidar com a proximidade de um semáforo e de um pedestre. Se a distância for inferior a 20, a um envio de comando ou ação *break(ID)* (linha 15) ao simulador, para efetuar uma

```
private void waitClientsTCP() {
    try {
        client = server.accept();
        logger.info("Server MAS received a conection with Unity Simulator");
        new Thread() {
            public void run() {
                String jsonStringReceived = new String();
                logger.info("All threads to handle percepts are online!!");
                try {
                    while (true) {
                        jsonStringReceived = Communicator.receiveMessageTCP(client);
                        logger.info("A socket json received: " + jsonStringReceived);
                        //convert json into object
                        Agent agent = JsonUtil.gson.fromJson(jsonStringReceived, Agent.class);
                        java.lang.reflect.Type listType = new TypeToken<LinkedList<Agent>>(){}.getType();
                        LinkedList<Agent> listObstaclesSeen = JsonUtil.gson.fromJson(agent.seen, listType);
                        //build perceptions and send to all agents
                        StringBuffer agentPercept = new StringBuffer();
                        agentPercept.append(agent.name + "(");
                        agentPercept.append(agent.id);
                        agentPercept.append(",");
                        agentPercept.append(agent.position_x);
                        agentPercept.append(",");
                        agentPercept.append(agent.position_y);
                        agentPercept.append(",");
                        agentPercept.append(agent.speed);
                        agentPercept.append(",");
                        agentPercept.append(agent.facing);
                        agentPercept.append(")");
                        addPercept(ASSyntax.parseLiteral(agentPercept.toString()));
                        StringBuffer perceptListObstaclesSeen = new StringBuffer();
                        for (Agent i : listObstaclesSeen) {
                            perceptListObstaclesSeen.append("seen(");
                            perceptListObstaclesSeen.append(agent.id);
                            perceptListObstaclesSeen.append(",");
                            perceptListObstaclesSeen.append(i.name);
                            perceptListObstaclesSeen.append(",");
                            perceptListObstaclesSeen.append(i.position_x);
                            perceptListObstaclesSeen.append(",");
                            perceptListObstaclesSeen.append(i.position_y);
                            perceptListObstaclesSeen.append(",");
                            perceptListObstaclesSeen.append(i.facing);
                            perceptListObstaclesSeen.append(",");
                            perceptListObstaclesSeen.append(i.speed);
                            perceptListObstaclesSeen.append(",");
                            perceptListObstaclesSeen.append(i.distance);
                            perceptListObstaclesSeen.append(")");
                            addPercept(ASSyntax.parseLiteral(perceptListObstaclesSeen.toString()));
                        }
```

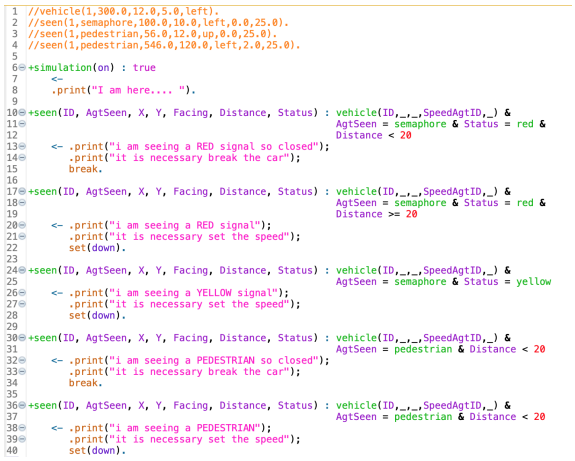
```
JSON received:
 {
  "id": 1,
  "name": "vehicle",
  "position_x": 300.0,
  "position_y": 12.0,
  "facing": "left",
  "speed": 5.0,
  "distance": 0.0,
  "state": "",
  "seen": "[\n   {\n    \"id\": 100,\n    \"name\": \"semaphore\"
  "around": ""
}

Agents sent:
vehicle(1,300.0,12.0,5.0,left)

List of seen:
seen(1,semaphore,100.0,10.0,left,0.0,25.0)
seen(1,pedestrian,56.0,12.0,up,0.0,25.0)
seen(1,pedestrian,546.0,120.0,left,2.0,25.0)
```

Figura 8. Código SMA do JASON para tratar os eventos vindos do Unity e convertê-los em percepções aos agentes do ambiente.

parada rápida do veículo. Já a Figura 10 mostra como esse comando ou ação vai para o simulador (linha 40).



```
1  //vehicle(1,300.0,12.0,5.0,left).
2  //seen(1,semaphore,100.0,10.0,left,0.0,25.0).
3  //seen(1,pedestrian,56.0,12.0,up,0.0,25.0).
4  //seen(1,pedestrian,546.0,120.0,left,2.0,25.0).
5
6  +simulation(on) : true
7      <-
8      .print("I am here.... ").
9
10 +seen(ID, AgtSeen, X, Y, Facing, Distance, Status) : vehicle(ID,_,_,SpeedAgtID,_) &
11                                                     AgtSeen = semaphore & Status = red &
12                                                     Distance < 20
13     <- .print("i am seeing a RED signal so closed");
14        .print("it is necessary break the car");
15        break.
16
17 +seen(ID, AgtSeen, X, Y, Facing, Distance, Status) : vehicle(ID,_,_,SpeedAgtID,_) &
18                                                     AgtSeen = semaphore & Status = red &
19                                                     Distance >= 20
20     <- .print("i am seeing a RED signal");
21        .print("it is necessary set the speed");
22        set(down).
23
24 +seen(ID, AgtSeen, X, Y, Facing, Distance, Status) : vehicle(ID,_,_,SpeedAgtID,_) &
25                                                     AgtSeen = semaphore & Status = yellow
26     <- .print("i am seeing a YELLOW signal");
27        .print("it is necessary set the speed");
28        set(down).
29
30 +seen(ID, AgtSeen, X, Y, Facing, Distance, Status) : vehicle(ID,_,_,SpeedAgtID,_) &
31                                                     AgtSeen = pedestrian & Distance < 20
32     <- .print("i am seeing a PEDESTRIAN so closed");
33        .print("it is necessary break the car");
34        break.
35
36 +seen(ID, AgtSeen, X, Y, Facing, Distance, Status) : vehicle(ID,_,_,SpeedAgtID,_) &
37                                                     AgtSeen = pedestrian & Distance < 20
38     <- .print("i am seeing a PEDESTRIAN");
39        .print("it is necessary set the speed");
40        set(down).
```

Figura 9. Planos para agentes *vehicles*, tratando eventos com os agentes *semaphore* e *pedestrian*.

```
35    @Override
36    public boolean executeAction(String agName, Structure action) {
37        if (agName.equals("vehicle")) {
38            try {
39                String message = action.getFunctor() + ":"+ action.getTerm(0).toString();
40                Communicator.sendMessageTCP(client, message);
41                Thread.sleep(200);
42            } catch (Exception e) {
43                e.printStackTrace();
44            }
45        }
46        return true; // the action was executed with success
47    }
```

Figura 10. Código Java que recebe a ação do agente e envia para o simulador.

5. Conclusões

Este trabalho apresentou o projeto para um sistema de simulação de tráfego autônomo via sistemas multiagentes e inteligência emergente. Mais especificamente, o sistema modelado assumiu que veículos são agentes autônomos capazes de comunicação e auto-organização, com o intuito de gerar um fluxo de tráfego eficiente. Além disso, os agentes do sistema também deveriam aprender a lidar com outros agentes independentes, que tomam a forma de pedestres e semáforos, ilustrando capacidade de lidar com situações fora de sua rede. Durante o estudo realizado, foram encontrados sistemas de simulação de tráfego e sistemas multiagentes com inteligência emergente. No entanto, a implementação de uma rede de veículos interconectados diferencia este trabalho dos demais e representa um conceito interessante para o uso de tecnologias diferentes, pois também inovou na comunicação entre Unity e a ferramenta JASON.

Para o desenvolvimento do sistema, foi definida a importância do conceito de *Swarm Intelligence* que, inspirado em colônias de insetos, mostrando a coordenação de indivíduos para a emergência natural de um sistema inteligente. Ao longo da modelagem e das implementações foi utilizado esse conceito nos agentes da simulação, visto que a rede de veículos tem semelhança com a ideia original.

Os resultados foram promissores, uma vez que: i) toda animação da simulação foi finalizada no ambiente Unity; ii) a comunicação entre simulador e SMA (via *socket* e uso de JSON) também foi concluída; iii) o tratamento dos eventos vindos do simulador por JSON e convertidos em percepções foi terminada; iv) foi realizada a implementação dos planos que tratam os eventos ativadores dos agentes em JASON.

Finalmente, há trabalho futuro a ser realizado, como refinar os planos dos agentes (seu raciocínio) e realizar testes mais efetivos de desempenho, para verificar se o planejamentos das ações dos agentes foi satisfatório. Destaque-se esse último ponto, pois o sistema está operacional e poderá ser utilizado para testar diferentes modelagens e abordagens de tratamento de tráfego urbano, seja por agentes veículos, pedestre e/ou semáforos.

Referências

Al Barghuthi, N. B. and Togher, M. (2020). Analysis of frameworks for traffic agent simulations. In *International Symposium on Intelligent Computing Systems*, pages 44–54. Springer.

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2021). JaCaMo project. http://jacamo.sourceforge.net/.

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *From Natural to Artificial Swarm Intelligence*. Oxford University Press, Inc., USA.

Bordini, R. H. and Hübner, J. F. (2007). A java-based interpreter for an extended version of agentspeak. *University of Durham, Universidade Regional de Blumenau*, 256.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*. Wiley.

Chollet, F. (2017). *Deep Learning with Python*. Manning.

Doniec, A., Mandiau, R., Piechowiak, S., and Espié, S. (2008). A behavioral multi-agent model for road traffic simulation. *Engineering Applications of Artificial Intelligence*, 21(8):1443–1454.

Ferreira, P. R., Boffo, F. S., and Bazzan, A. L. (2007). Using swarm-gap for distributed task allocation in complex scenarios. In *International conference on autonomous agents and multiagent systems*, pages 107–121. Springer.

Hübner, J. F. and Bordini, R. H. (2010). Using agent- and organisation-oriented programming to develop a team of agents for a competitive game. *Ann. Math. Artif. Intell.*, 59(3-4):351–372.

Liu, H., Yu, H., Li, Y., and Sun, Y. (2011). A role modelling approach for crowd animation in a multi-agent cooperative system. In *Proceedings of the 2011 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 304–310.

Padgham, L. and Winikoff, M. (2004). *Developing Intelligent Agent Systems: A practical guide.* John Wiley & Sons.

Sedgewick, R. and Wayne, K. (2011). *Algorithms*. Addison-Wesley Professional, Boston, 4th edition.

Shannon, R. and Johannes, J. D. (1976). Systems simulation: The art and science. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(10):723–724.

Silveira, P., Silveira, G., Lopes, S., Moreira, G., Steppat, N., and Kung, F. (2012). *Introdução à arquitetura e Design de Software*. Elsevier Editora.

Topol, E. J. (2019). High-performance medicine: the convergence of human and artificial intelligence. *Nature medicine*, 25(1):44–56.

Uhrmacher, A. M. and Weyns, D. (2009). *Multi-Agent Systems: Simulation and application*. Computational analysis, synthesis, and design of dynamic models series. CRC Press, Boca Raton, FL, USA.

Vester, S., Boss, N. S., Jensen, A. S., and Villadsen, J. (2011). Improving multi-agent systems using jason. *Annals of Mathematics and Artificial Intelligence*, 61(4):297–307.

Wooldridge, M. (2001). *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA.

Zamberlan, A. (2018). *Sistema Multiagente para avaliação do efeito de aglomeração em nanopartículas poliméricas*. PhD thesis, Universidade Franciscana - UFN, Santa Maria.

Zamberlan, A., Bordini, R., Kurtz, G. C., and Fagan, S. B. (2020). Multi-agent systems, simulation and nanotechnology. In *Multi Agent Systems-Strategies and Applications*. IntechOpen.

Zamberlan, A., Perozzo, R., Kurtz, G., Librelotto, G., and Fagan, S. (2014). Integrando agentes AgentSpeak(L) em ambientes pervasivos educacionais. In *WESAAC - Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações*, pages 1–13, Porto Alegre. SBC.