Integração do SUMO e Traffic3D com o framework de agentes MASPY

Gabriel Galvan Neres[1], Rafael C. Cardoso[2], André P. Borges[1], Gleifer V. Alves[1]

[1]Universidade Tecnológica Federal do Paraná (UTFPR)
Ponta Grossa, PR, Brasil

[2]University of Aberdeen
Aberdeen, United Kingdom

neres@alunos.utfpr.edu.br, rafael.cardoso@abdn.ac.uk

{apborges,gleifer}@utfpr.edu.br

Abstract. *This work addresses the integration of the MASPY agent framework with two graphical traffic simulators: SUMO and Traffic3D in Unity. The goal is to provide a graphical visualization for MASPY and expand its range of available tools, achieved through socket communication and JSON message exchange to connect MASPY to the simulators. For SUMO, an intermediate server was implemented to convert JSON messages into the TraCI protocol. In the case of Traffic3D, the integration was done directly without the need for an intermediate server. The results show that the approach enables graphical visualization of the simulations, broadening MASPY's applicability in real-world scenarios. Finally, the work discusses the ease and limitations encountered, as well as suggestions for future expansions.*

Resumo. *Este trabalho aborda a integração do framework de agentes MASPY com dois simuladores gráficos de tráfego: SUMO e Traffic3D na Unity. O objetivo é fornecer uma visualização gráfica para a MASPY e aumentar sua gama de ferramentas disponíveis, realizando isso por meio de comunicação via socket e troca de mensagens JSON para conectar o MASPY aos simuladores. Para o SUMO, foi implementado um servidor intermediário que converte as mensagens JSON para o protocolo TraCI. Já para o Traffic3D, a integração ocorreu diretamente sem necessitar de um servidor intermediário. Os resultados mostram que a abordagem permite visualização gráfica das simulações, ampliando as possibilidades de uso do MASPY em cenários reais. Por fim, discute-se a facilidade e limitações encontradas, além de sugerir futuras expansões.*

1. Introdução

Nas últimas décadas, os Sistemas Multiagentes (SMA) têm ganhado destaque devido à sua capacidade de representar, simular e resolver problemas nos mais diversos cenários. Com a criação de novas tecnologias para desenvolvimento de agentes, entre as ferramentas criadas, destaca-se a MASPY [Mellado et al. 2023], um framework desenvolvido em Python visando facilitar a programação de agentes BDI (*Belief-Desire-Intention*), tornando o desenvolvimento de SMA mais acessível, principalmente para usuários iniciantes.

Durante os testes realizados com a MASPY, identificou-se uma limitação importante: a ausência de recursos gráficos para visualização das simulações, visto que as

simulações ocorrem exclusivamente via interface de linha de comando, dificultando a visualização dos eventos da simulação. Este trabalho demonstra a viabilidade de ester o MASPY a simuladores gráficos com o intuito de superar essa limitação e de realizar uma expansão do framework, para ter acesso a novas ferramentas, foram desenvolvidas duas integrações com simuladores gráficos, utilizando conexões via *Socket* e troca de mensagens no formato JSON. Os simuladores escolhidos para essa integração foram o SUMO [Lopez et al. 2018], voltado à simulação de tráfego urbano, e o Traffic3D [Garg et al. 2019] sendo utilizado na plataforma Unity [Unity Technologies 2023].

Para a realização desse trabalho foi realizado um estudo de trabalhos prévios que exploram integrações entre diferentes simuladores, como o estudo apresentado em [Engelmann et al. 2021], em [Van Haare Heijmeijer and Vaz Alves 2018] e [Soares et al. 2013]. A escolha do cenário utilizado para esta integração foi baseada no trabalho de [Izidorio and Alves 2024]), que propõe a simulação de um cruzamento urbano controlado por um semáforo.

Este artigo está estruturado da seguinte forma: a Seção 2 apresenta as ferramentas utilizadas; a Seção 3 descreve a metodologia empregada para o desenvolvimento da integração; a Seção 4 detalha a implementação das integrações com cada simulador; a Seção 5 discute os resultados obtidos; e por fim, a Seção 6 apresenta as conclusões e sugestões para trabalhos futuros.

2. Referencial Teórico

Esta seção apresenta uma visão geral sobre frameworks BDI em Python, trabalhos relacionados à integração de SMA com simuladores gráficos e os simuladores utilizados neste projeto.

2.1. Frameworks de Agentes BDI em Python

Os frameworks BDI (*Belief-Desire-Intention*) têm como objetivo facilitar a implementação de agentes, permitindo que eles tomem decisões com base em crenças, desejos e intenções. Com o intuito de facilitar o desenvolvimento de SMA para novos programadores, o framework MASPY [Mellado et al. 2023] foi desenvolvido em Python para a programação de agentes BDI, podendo criar ambientes com múltiplos agentes que comunicam entre si e com o próprio ambiente.

2.2. Integrações de SMA com Simuladores Gráficos

A integração entre SMA e simuladores gráficos é uma estratégia que visa representar cenários de forma visual e interativa. Simuladores gráficos permitem observar o comportamento dos agentes, o que facilita a validação das regras implementadas e a comunicação de resultados.

Trabalhos como o apresentado em [Engelmann et al. 2021] mostram a integração com o framework JaCaMo, permitindo o controle em ambientes simulados por agentes BDI. Já em [Van Haare Heijmeijer and Vaz Alves 2018], é apresentada uma integração entre JaCaMo e o simulador SUMO. E [Soares et al. 2013] criam uma plataforma de simulação de tráfego baseada em agentes, usando SUMO e JADE.

2.3. Uso de SUMO e Traffic3D e sistemas multiagente

Para o trabalho em questão foram escolhidas duas ferramentas, que possuem caractéristicas próprias de simulações de cenários de tráfego urbano, pois se encaixar no cenário simulado.

O **SUMO** (***S**imulation of **U**rban **MO**bility*) [Lopez et al. 2018] é um simulador de código aberto que permite ao usuário desenvolver simulações de cenários de tráfego urbano. A ferramenta possibilita a criação de cenários de tráfego via definição de ruas, interseções, semáforos, rotas e comportamentos individuais de veículos, sendo configurável e compatível com diversos formatos de dados e ferramentas externas. Além de oferecer recursos avançados para o controle de tráfego urbano, permite também a visualização de maneira gráfica da simulação.

O **Traffic3D** [Garg et al. 2019] é uma ferramenta de simulação de tráfego, a qual foi desenvolvida para aplicações em sistemas de transporte inteligente e tráfego urbano, utilizada em projetos científicos que envolvam aprendizado por reforço e tarefas de detecção e segmentação de objetos como, por exemplo [Garg et al. 2020] e [Alfikri and Kaliski 2024] que utilizam o simulador.

3. Metodologia

Para a realização desse trabalho foi implementado um cenário da MASPY como base. O cenário em questão consta como um cruzamento de duas vias, uma na horizontal e outra na vertical, onde os carros surgem na parte inferior da rua vertical, e se movem rumo ao topo da mesma. Na interseção das duas vias, consta um semáforo que, em um primeiro momento se encontra na cor vermelha. Quando o primeiro carro chega à encruzilhada e verifica que o sinal está vermelho, ele então opta por aguardar. O sinal é modificado para verde, permitindo assim o cruzamento do veículo. Os próximos veículos que chegaram ao cruzamento verificam que o sinal já consta na cor verde, assim então atravessando-o diretamente em vez de aguardar a mudança de cor.

Para a integração do código no framework MASPY, foram selecionados dois simuladores, SUMO e Traffic3D. Para realização dessas integrações era fundamental a utilização de um mesmo código MASPY para conectar-se com os dois simuladores, por conta disso, foi implementada uma solução via conexão *socket*, tornando o código MASPY um cliente e criando os códigos dos simuladores atuando como Servidores, e utilizar formato JSON por se tratar de um formato leve e eficiente para a troca de mensagens entre Cliente e Servidor.

Para realizar uma análise mais precisa dos resultados obtidos com a integração, foram definidos alguns critérios de avaliação. Esses parâmetros visam identificar os pontos fortes e fracos de cada solução integrada, bem como as dificuldades enfrentadas durante o processo. Os critérios avaliados são: (**i.**) a facilidade de integração; (**ii.**) compatibilidade com o framework MASPY e com o ambiente Python; (**iii.**) facilidade de uso. Com base nesses critérios, será possível avaliar se a integração mostrou-se adequada ou não.

4. Integrando simuladores com o framework MASPY

Para definir a melhor forma de realizar a integração, o principal objetivo foi garantir que o mesmo código MASPY pudesse ser utilizado com ambas as ferramentas. Com isso

em mente, optou-se por implementar a comunicação por meio de conexões via *socket*, configurando o código MASPY como cliente e os simuladores como servidores. A troca de informações entre eles é realizada por meio de mensagens no formato JSON.

Para viabilizar o envio das mensagens JSON aos simuladores, foi implementada uma função genérica em Python chamada `send_json` mostrada no Listing 1. Essa função estabelece uma conexão via protocolo TCP com o servidor especificado, utilizando por padrão o endereço *localhost* e a porta 5000. Caso o servidor não esteja disponível no momento da conexão, a função trata e exibe uma mensagem informando que o servidor não está acessível. Essa abordagem garante uma comunicação direta e eficiente entre o código MASPY e os servidores dos simuladores, permitindo o envio contínuo de mensagens durante a simulação.

```
import socket
import json

def send_json(data, host='localhost', port=5000):
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM)
           as s:
            s.connect((host, port))
            s.sendall(json.dumps(data).encode('utf-8'))
    except ConnectionRefusedError:
        print("Server not available")
```

Listing 1. Função de envio de mensagens JSON via socket

A integração do simulador SUMO exigiu uma abordagem indireta, pois o SUMO não possui suporte nativo para JSON nem para comunicação via *Socket*. Uma vez que o SUMO utiliza comunicação via TraCI (Interface de Controle de Tráfego). Para contornar essa limitação, foi implementado um servidor intermediário em Python, esse servidor recebe as mensagens em formato JSON enviados pela MASPY, interpreta e converte esses comandos e as informações para TraCI, enviando-os para o SUMO, que apresenta gráficamente os enventos que ocorrem simultaneamente na MASPY. Foi adicionado também, um *delay* geral na simulação, para que seja possível ver os eventos. A Fig. 1 representa o fluxo na troca de mensagens desta integração.

Figura 1. Diagrama de funcionamento da integração ao SUMO

Já no caso da integração ao Traffic3D ocorreu de forma direta, uma vez que a Unity oferece suporte nativo tanto a conexões por *Socket* quanto a receber mensagens via JSON. Dessa forma, o MASPY atua como cliente, conectando-se diretamente ao servidor implementado dentro da Unity e enviando as mensagens JSON que descrevem os estados

e eventos que estão ocorrendo na simulação em tempo real. Na Unity é possível visualizar o que está ocorrendo na simulação MASPY, porém por se tratar de uma visualização gráfica dos eventos, foi necessário adicionar um *delay* geral na simulação, pois sem ele é visualmente impossível visualizar os eventos. A Fig. 2 demonstra o diagrama de como funciona a troca de mensagens.

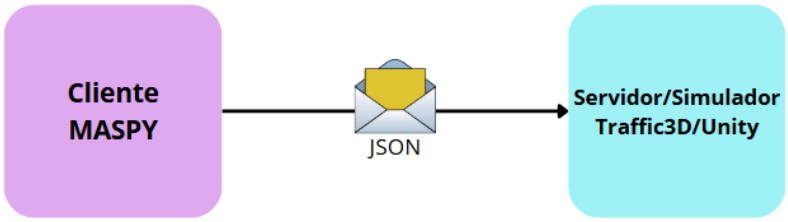

Figura 2. Diagrama de funcionamento da integração ao Traffic3D

Para o cenário implementado, foram programadas quatro diferentes mensagens JSON enviadas aos servidores. O formato interno principal de cada JSON é o *event* referente ao evento ocorrendo na simulação, como a criação de veículos, com a quantidade total que foram criados passado em conjunto. O segundo é referente a cor do semáforo, podendo ser vermelha ou verde, os dois ultimos indicam o estado de cada veículo, podendo ele estar parado (*waiting*) ou estar andando (*crossing*).

```
send_json({"event": "new_vehicle", "agent": num_veiculos})
send_json({"event": "traffic_light", "color": aux})
send_json({"event": "crossing", "agent": src})
send_json({"event": "waiting", "agent": src})
```

Após o recebimento da mensagem JSON, o código do simulador vai interpretar cada uma delas, e dependendo de qual tipo de evento da mensagem, o simulador vai fazer a interpretação e mostrar na tela as alterações da simulação de maneira gráfica. Ambos os simuladores possuem um *delay*, que diminui a velocidade da simulação de maneira geral, para possibilitar uma visualização adequada do que está ocorrendo.

5. Resultados

Após ter realizado as duas integrações, foi feita a avaliação baseada nos critérios propostos anteriormente, a primeira avaliação é da integração ao SUMO, ver Fig. 3, apesar do simulador não possuir suporte JSON e *Socket* nativo, é completamente viável uma integração ao criar um servidor intermediário. A Fig. 3 mostra o momento em que um veículo, representado pelo triangulo amarelo, está fazendo a travessia do cruzamento, onde consta com a sinaleira, representada pela barra colorida, está verde.

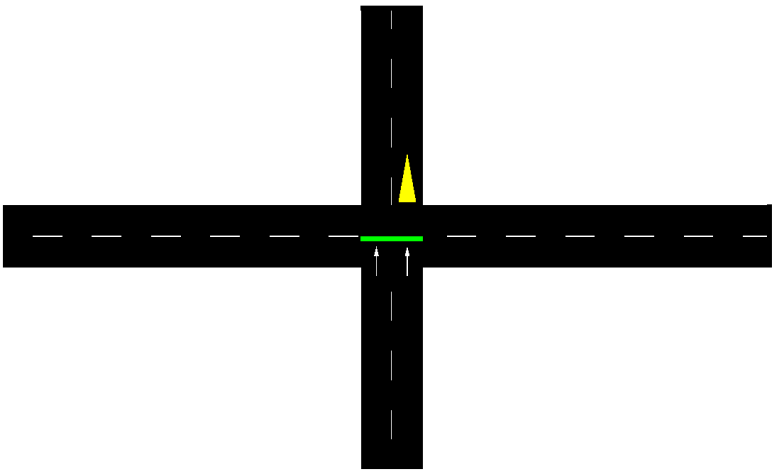

Figura 3. Tela SUMO durante a execução do experimento

A integração ao Traffic3D na Unity, ver Fig. 4, possui compatibilidade JSON e *Socket*, permitindo assim uma integração direta a MASPY, sem necessidade de nenhum servidor intermediário, foram usados os *Assets* disponibilizados pelo simulador, de veículos, ruas, calçadas e do semáforo. A Fig. 4 está no momento em que um veículo, representado pelo modelo de um carro, está fazendo a travessia do cruzamento, a sinaleira está representada pelo objetivo cinza na parte inferior da figura.

Figura 4. Tela Unity durante a execução do experimento

Após duas integrações da MASPY foi possível concluir fazendo uma avaliação do framework a facilidade de integrá-lo a algum simulador. O framework possui capacidade de se conectar via *Socket* com servidores distintos, mantendo o mesmo código. As alterações necessárias no código MASPY é somente uma função de conexão ao servidor, e a adição de linhas de código para mandarem JSON ao servidor.

Na Tabela 1 foi elencada uma comparação direta dos dois simuladores, onde os três métodos de avaliação foram avaliados, a facilidade da integração, se ela ocorreu de maneira direta ou indireta, a compatibilidade com a MASPY, se o servidor conseguiu receber as informações do SMA vindo da MASPY e conseguiu demonstrar em tela e, por

fim, a facilidade de uso, onde foi analisado se o simulador é relativamente fácil de se usar sem demandar nenhum conhecimento avançado de maneira externa.

Tabela 1. Análise dos simuladores

	SUMO	Traffic3D
Facilidade de integração	Não possui compatibilidade para JSON e Socket	Possui compatibilidade para JSON e Socket
Compatibilidade com a MASPY	Possui compatibilidade	Possui compatibilidade
Facilidade de uso	Possui facilidade de uso	Possui facilidade de uso

6. Conclusão

Pela necessidade de visualização gráfica do framework MASPY, e para expandir a gama de ferramentas disponíveis ao utilizar o framework, foram realizadas duas integrações com dois diferentes simuladores, SUMO e Traffic3D/Unity, cada uma delas apresentou diferentes dificuldades. Após uma análise dos resultados foi possível identificar que é possível integrar o framework em simuladores. Tendo facilidades e dificuldades com sua compatibilidade *socket* e JSON, os resultados da integração podem ser considerados positivos, apesar das atuais limitações de não generalização dos experimentos, e de uma necessidade do servidor possuir as compatibilidades.

A integração implementada consiste em um experimento inicial, visando testar a capacidade do framework de se conectar a simuladores, identificando as limitações e as possibilidades possíveis. No futuro, pretende-se expandir as integrações, possibilitando realizá-las de forma genérica em vez de um cenário específico, além de buscar integrações com outros simuladores.

Agradecimentos: Este trabalho tem financiamento do projeto: 444568/2024-7, CNPq/MCTI/FNDCT Nº 22/2024, *Programa Conhecimento Brasil – Apoio a Projetos em Rede com Pesquisadores Brasileiros no Exterior.*

Referências

Alfikri, M. D. and Kaliski, R. (2024). Real-time pedestrian detection on iot edge devices: a lightweight deep learning approach. *arXiv preprint arXiv:2409.15740*.

Engelmann, D., Damasio, J., Krausburg, T., Borges, O., Cezar, L. D., Panisson, A. R., and Bordini, R. H. (2021). Dial4jaca – a demonstration. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Social Good. The PAAMS Collection: 19th International Conference, PAAMS 2021, Salamanca, Spain, October 6–8, 2021, Proceedings*, page 346–350, Berlin, Heidelberg. Springer-Verlag.

Garg, D., Bugajski, C., Mount, S., Vogiatzis, G., and Chli, M. (2019). Traffic3d: A rich 3d traffic environment to train intelligent agents.

Garg, D., Chli, M., and Vogiatzis, G. (2020). Multi-agent deep reinforcement learning for traffic optimization through multiple road intersections using live camera feed. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8. IEEE.

Izidorio, F. M. and Alves, G. V. (2024). Implementação de agentes com técnicas de aprendizagem por reforço. In *Anais do XIV Seminário de Extensão e Inovação XXIX Seminário de Iniciação Científica e Tecnológica da UTFPR - SEI/SICITE*, Francisco Beltrão, Paraná, Brasil. UTFPR.

Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., and Wießner, E. (2018). Microscopic traffic simulation using sumo. In *Proceedings [...]*, pages 2575–2582, Maui, USA. IEEE International Conference on Intelligent Transportation Systems, IEEE.

Mellado, A. L. L., Fidler, I. G., Borges, A. P., and Alves, G. V. (2023). MASPY: Towards the Creation of BDI Multi-Agent Systems. In *Proceedings [...]*, pages 106–117, Pelotas, RS, Brasil. Anais do XVII Workshop Escola de Sistema de Agentes seus Ambientes e Aplicações WESAAC 2023, Zenodo.

Soares, G., Macedo, J., Kokkinogenis, Z., and Rossetti, R. (2013). An integrated framework for multi-agent traffic simulation using sumo and jade. In *SUMO2013, The first SUMO user conference*, pages 15–17.

Unity Technologies (2023). Unity. `https://unity.com/`. Game development platform.

Van Haare Heijmeijer, A. and Vaz Alves, G. (2018). Development of a middleware between sumo simulation tool and jacamo framework. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, 7(2):5–15.