UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

VÍCTOR MUNIZ DOS SANTOS

INTERLIGAÇÃO DO FRAMEWORK JADE COM A FERRAMENTA SUMO EM UM DOMÍNIO DE SMART PARKING BASEADO EM AGENTES

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2020

VÍCTOR MUNIZ DOS SANTOS

INTERLIGAÇÃO DO FRAMEWORK JADE COM A FERRAMENTA SUMO EM UM DOMÍNIO DE SMART PARKING BASEADO EM AGENTES

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação do Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. André Pinz Borges

PONTA GROSSA

2020

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa

Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Bacharelado em Ciência da Computação

TERMO DE APROVAÇÃO

INTERLIGAÇÃO DO FRAMEWORK JADE COM A FERRAMENTA SUMO EM UM DOMÍNIO DE SMART PARKING BASEADO EM AGENTES
por

VÍCTOR MUNIZ DOS SANTOS

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 27 de outubro de 2020 como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

____
Prof. Dr. André Pinz Borges
Orientador(a)

____
Prof. Dr. Gleifer Vaz Alves
Membro titular

____
Prof. Dr. André Koscianski
Membro titular

____
Prof. MSc. Geraldo Ranthum
Responsável pelo Trabalho de Conclusão de Curso

____
Profª. Dra. Mauren Louise Sguario
Coordenador do curso

AGRADECIMENTOS

Muitas pessoas passaram por minha vida ao longo de toda essa caminha ao decorrer do curso. Não irei me atentar a nomes, mas cada uma dessas pessoas marcou minha vida e levam um pouco de responsabilidade por eu ter chego até aqui. Por isso, deixo um enorme agradecimento a cada uma dessas pessoas.

Devo deixar um agradecimento ao meu orientador Prof. Dr. André Pinz Borges por me guiar até a conclusão deste projeto.

De forma alguma eu poderia deixar de dar os agradecimentos especiais a toda a minha família. Mesmo de longe, nunca deixei de receber apoio dele para que eu conseguisse chegar a esse ponto.

RESUMO

MUNIZ, Víctor. **Interligação do framework JADE com a ferramenta SUMO em um domínio de Smart Parking baseado em agentes**. 2020. 91 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2020.

O Termo “Cidade Inteligente” tem sido cada vez mais empregado nos dias de hoje. A tecnologia tem sido inserida em diversos campos de uma cidade. O campo abordado nesse trabalho é o da *Smart Mobility*, mais especificamente os *Smart Parkings,* onde são vistos temas da utilização de sistemas multiagentes para a resolução de problemas de mobilidade urbana com a utilização de *Smart Parkings* desenvolvidos utilizando o Framework JADE e comparação com resultados já obtidos utilizando outros *frameworks*. Este trabalho tem como objetivo o desenvolvimento de um protótipo de um sistema multiagente para um Smart Parking utilizando framework JADE e também a ferramenta de simulação de trânsito SUMO para análise gráfica dos cenários propostos no projeto. A integração entre as ferramentas utilizadas se demonstrou viável para a criação de um *Smart Parking* produzindo, no decorrer de diversos cenários, resultados satisfatórios sem problemas demasiadamente conflitantes a este trabalho.

Palavras-chave: Estacionamento Inteligente. JADE. Sistema Multiagente. Cidade Inteligente. Agente.

ABSTRACT

MUNIZ, Víctor**. Interconnection of JADE framework with SUMO tool in an agent-based Smart Parking Domain**. 2020. 92 p. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Federal University of Technology - Paraná. Ponta Grossa, 2020.

The term "Smart Parking" has been increasingly employed these days. The high technology has been inserted in several fields of a city. The field covered in this work is that of Smart Mobility, more specifically Smart Parkings. We discuss the use of multi-agent systems to solve problems of urban mobility using Smart Parkings developed using the JADE Framework and the comparison with results already obtained using other frameworks. This work aims to develop a prototype of a multi-agent system for a Smart Parking using the JADE framework and also the SUMO traffic simulation tool for graphical analysis of the proposed scenarios. The integration between the tools used proved to be feasible for the creation of a Smart Parking, producing, in the course of several scenarios, satisfactory results without presenting problems that are too conflicting to this work.

Keywords: Smart Parking. JADE. Multiagent System. Smart City. Agent.

LISTA DE ILUSTRAÇÕES

Figura 1 - Um agente interagindo com o seu ambiente por meio de sensores e atuadores 21
Figura 2 - Interface gráfica do JADE 36
Figura 3 - Utilização do Sniffer para a visualização de troca de mensagens entre os agentes 38
Figura 4 - Utilização do sniffer na comunicação de 5 agentes 39
Figura 5 - Exemplo de simulação de trânsito no SUMO 41
Figura 6 - Estrutura de arquivos de entrada necessários para o SUMO 46
Figura 7 - Simulação de estacionamento utilizando CityMobil 48
Figura 8 - Interligação entre o JADE e o SUMO 51
Figura 9 - Simulação de estacionamento utilizando todas as vagas 62
Figura 10 - Diagrama de atividades do projeto 63
Figura 11 - Cenário 1 da simulação 70
Figura 12 - Simulação do cenário 2 73
Figura 13 - Simulação do cenário 3 76
Figura 14 - Simulação do cenário 4 79
Figura 15 - Comunicação de agentes no sniffer 81

LISTA DE TABELAS

Tabela 1 - Relação entre o conjunto de veículos e os setores da simulação 69

LISTA DE QUADROS

Quadro 1 - Relação entre a cor e o valor de avaliação 56

LISTA DE GRÁFICOS

Gráfico 1 - Percentual utilizado do estacionamento do cenário 1.............................71
Gráfico 2 - Quantidade de veículos na fila de espera do cenário 1 72
Gráfico 3 - Percentual utilizada do estacionamento no cenário 2.............................74
Gráfico 4 - Quantidade de veículos na fila de espera do cenário 2...........................75
Gráfico 5 - Percentual utilizada do estacionamento no cenário 3.............................77
Gráfico 6 - Quantidade de veículos na fila de espera no cenário 3...........................78
Gráfico 7- Percentual utilizada do estacionamento para o cenário 4 80
Gráfico 8 - Quantidade de veículos na fila de espera do cenário 4...........................80

LISTA DE CÓDIGOS

Código 1 - Desenvolvimento de um agente em JADE 35
Código 2 - Criação de agentes utilizando o framework JADE 35
Código 3 - Criação e envio de mensagem em JADE 37
Código 4 - Exemplo da criação de nós da simulação..42
Código 5 -Exemplo da criação de vias da simulação 42
Código 6 - Exemplo de códigos dos tipos na simulação 43
Código 7 - Exemplo da criação de conexões da simulação 43
Código 8 - Exemplo de utilização da ferramenta NETCONVERT 44
Código 9 - Exemplo de criação de rotas para a simulação 44
Código 10 - Exemplo de arquivo de configuração do SUMO 45
Código 11 - Rotas de chegada e saída de uma vaga da simulação 49
Código 12 - Tipos de veículos utilizados na simulação 50
Código 13 – Criação dos agentes da simulação 52
Código 14 - Informações importantes à simulação 53
Código 15 – Alocação de vagas...54
Código 16 - Troca de mensagens 55
Código 17 - Gerenciadores de tempo e fila de espera 55
Código 18 - Solicitação de vaga pelo driver 57
Código 19 - Inserção de veículo pela biblioteca TraaS 57
Código 20 - Remoção de veículos 59
Código 21 - Inserção do veículo na fila de espera 60
Código 22 - Função de inserção na fila de espera 61
Código 23 - Gerenciador da fila de espera...61

SUMÁRIO

1 INTRODUÇÃO 14
1.1 OBJETIVOS 15
1.1.1 Objetivo Geral 15
1.1.2 Objetivos específicos 16
1.2 JUSTIFICATIVA 16
2 TRABALHOS RELACIONADOS 18
2.1 CONSIDERAÇÕES FINAIS 20
3 AGENTES E SISTEMAS MULTIAGENTES 21
3.1 PRINCIPAIS CARACTERÍSTICAS DOS AGENTES 22
3.1.1 Autonomia 22
3.1.2 Habilidade Social 23
3.1.3 Reatividade 23
3.1.4 Pró-Atividade 23
3.2 ARQUITETURA DOS AGENTES 25
3.2.1 Arquitetura Reativa 25
3.2.2 Arquitetura Cognitiva 25
3.3 AMBIENTES 26
3.4 SISTEMA MULTIAGENTES 27
3.4.1 Sistemas Multiagentes Reativos 28
3.4.2 Sistemas Multiagentes Cognitivos 28
3.5 COMUNICAÇÃO ENTRE AGENTES 29
3.6 CONSIDERAÇÕES FINAIS 29
4 SMART CITY E SMART PARKING 30
4.1 SMART PARKING 31
4.1.1 Projeto Proposto por Di Napoli *et. al.* (2014) 31
4.1.2 Projeto Proposto por Castro (2015) 32
4.1.3 Considerações Finais 32
5 FRAMEWORK JADE 34
6 VISÃO GERAL DO SUMO 40
6.1 SUMO SIMULATOR 40
6.1.1 *TraCI* 46
6.2 CONSIDERAÇÕES FINAIS 47
7 INTERLIGAÇÃO ENTRE O SUMO E O FRAMEWORK JADE 48
7.1 MODELO DE ESTACIONAMENTO 48
7.2 INTERLIGAÇÃO COM O PROJETO 50
7.2.1 Visão geral 52
7.2.2 Alocação de veículos 56
7.2.3 Remoção de veículos 58

7.2.4 Inserção de veículos na fila de espera....59
7.3 CONSIDERAÇÕES FINAIS....64
8 EXPERIMENTOS....65
8.1 CONFIGURAÇÕES DE AMBIENTE E FERRAMENTAS....65
8.1.1 Configurações do Sistema....65
8.1.2 Informações Sobre os Agentes....66
8.1.3 Criação dos cenários....67
8.2 RESULTADOS OBTIDOS....69
8.2.1 Cenário 1....70
8.2.2 Cenário 2....73
8.2.3 Cenário 3....76
8.2.4 Cenário 4....79
8.2.5 Comunicação entre os agentes....81
8.3 RESULTADOS OBTIDOS....82
8.4 CONSIDERAÇÕES FINAIS....83
9 CONCLUSÃO....84
10 REFERÊNCIAS....86

1 INTRODUÇÃO

Um dos maiores problemas encontrados por motoristas nos grandes centros urbanos é a procura por uma vaga de estacionamento. Motoristas despendem, desnecessariamente, tempo buscando vagas para estacionar. Combustível é gasto, o que leva a um aumento dos índices de poluição, além do estresse gerado para os motoristas durante a busca por uma vaga, o que resulta, muitas vezes, até em acidentes de trânsito (DEUTSCHE TELEKOM, 2017). Talvez uma solução possível seria a criação de novos estacionamentos nas cidades, mas nem sempre isso é viável.

Com o uso da tecnologia é possível facilitar a busca por uma vaga de estacionamento. Uso de sensores de veículos, monitoração de estacionamentos e comunicação wireless são alguns exemplos de solução para melhorar as estratégias de mobilidade numa cidade inteligente. Utilizando dessas e outras tecnologias, é possível a criação de *Smart Parkings* (Di Napoli *et. al.,*2014). Num No *Smart Parking*, o objetivo é acabar com os empecilhos trazidos pelo estacionamento comum. Existem diversos modelos implementados de *Smart Parkings*. Neste protótipo, existe um agente gerenciador de todas as vagas de estacionamento disponíveis numa cidade. Dessa maneira, qualquer motorista poderá saber a quilômetros de distância onde se encontram vagas de estacionamento disponíveis. O motorista pode requisitar uma vaga de estacionamento ao gerenciador. Esse gerenciador inicia um processo de negociação levando em conta diversos parâmetros para decidir se deve ou não alocar a vaga. Disponibilidade da vaga, avaliação com o motorista, por exemplo, são fatores que serão levados em conta para alocar a vaga ao motorista.

Os resultados pretendidos fazem com que o motorista não perca mais tempo desnecessariamente em busca de uma vaga, pois muito antes de chegar na mesma, ele já sabe que aquela vaga está disponível para ele. Indiretamente isso acarreta na diminuição do tráfego de carros nas cidades, diminuição da emissão de gás carbônico na atmosfera etc.

Com a utilização de técnicas de um Sistema MultiAgente (SMA) para a criação de soluções de um *Smart Parking*, foi criado o Projeto MAPS (*Multiagent Parking System*). Ele está contido no GPAS (Grupo de Pesquisa em Agentes de Software – UTFPR – PG) com os primeiros resultados obtidos por Castro (2015) que desenvolveu o JaCaMo utilizando Multiagentes apresentando uma solução para um sistema de alocação de vagas de estacionamento.

Assim como o trabalho proposto por Castro (2015), Heijimeijer (2016) também propôs a construção de um sistema de *Smart Parking* com a utilização do JaCaMo, porém fazendo a interligação deste com a ferramenta de simulação SUMO possibilitando a análise visual dos cenários descritos.

Neste trabalho foi desenvolvido um protótipo de um Sistema Multiagente para um *Smart Parking* utilizando o *framework* JADE, que é um *framework* totalmente desenvolvido em JAVA e que facilita a criação de sistemas multiagentes (JADE, 2000). O JADE ainda conta com uma ferramenta visual para o gerenciamento dos agentes conseguindo obter informações como a comunicação que está sendo feita entre os agentes, por exemplo.

Para a parte gráfica, foi feito uso da ferramenta SUMO que é um simulador de tráfego. Com ele, é possível a criação de diversos tipos de veículos, estradas, rotas etc, para uma simulação de tráfego além de ser uma ferramenta *open source* (SUMO. 2001).

Foram desenvolvidos agentes condutores de veículos e um agente gerenciador de um estacionamento. Cabe ressaltar que o foco deste trabalho está na integração dos frameworks, logo, questões como mecanismos de negociação, otimização dos espaços do estacionamento, fluxo de veículos não foram abordadas com profundidade. São implementados apenas mecanismos necessários para a integração dos frameworks. Pretende-se, com este protótipo, realizar uma comparação com resultados obtidos anteriormente por Heijimeijer (2016), que utilizou a ferramenta JaCaMo. A integração entre as ferramentas utilizadas se mostrou viável para a criação de um *Smart Parking* produzindo, no decorrer de diversos cenários, resultados satisfatórios sem problemas como falha na interligação ou comunicação.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

O objetivo deste trabalho é a interligação do *framework* JADE com a ferramenta de simulação SUMO em um domínio de *Smart Parking* baseado em agentes.

1.1.2 Objetivos específicos

Os seguintes objetivos específicos deverão ser alcançados neste trabalho:

- Analisar o funcionamento do framework JADE utilizado no projeto;
- Adaptação da simulação do *City Mobil* utilizado nesse projeto;
- Criação do arquivo de rotas da simulação;
- Interligar o *framework* JADE com a ferramenta SUMO utilizando a biblioteca TraaS;
- Implementar os agentes que estarão presentes no sistema utilizando o *framework* JADE;
- Definir os cenários de teste;
- Analisar os resultados obtidos para verificar se foi possível garantir viabilidade do projeto.

1.2 JUSTIFICATIVA

O projeto visa o desenvolvimento de um protótipo de Sistema Multiagente de *Smart Parking* utilizando o *framework* JADE. Esse sistema já foi anteriormente desenvolvido no projeto MAPS por Castro (2015), porém fazendo a utilização da ferramenta JaCaMo. O intuito desse sistema de *Smart Parking* é a existência de dois tipos de agentes: *Manager* e *Driver*. O agente *Manager*, como o nome já diz, é o gerenciador do estacionamento. Ele deve ter todo o controle das vagas disponíveis e ocupadas em todo o sistema e também deve realizar toda a negociação e comunicação com os agentes *Driver*. Os agentes *Driver* apenas requisitam a vaga. Requisitando a vaga, informações como distância do *Driver* da vaga, o tempo de estadia na vaga, a avaliação que o agente *Driver* tem no sistema, etc.

Este trabalho objetiva a interligação do *framework* JADE com a ferramenta SUMO em um domínio de *Smart Parking* baseado em agentes para a obtenção de resultados gráficos da simulação para que seja realizada melhor análise sobre os resultados obtidos e também possíveis melhorias no sistema para trabalhos futuros.

Além disso, foi escolhida a utilização da ferramenta SUMO para a simulação gráfica desse projeto. A escolha dessa ferramenta deve-se a ela ser gratuita e de código-fonte aberto, o que permite que usuário consiga extrair maior proveito a

ferramenta e ainda possibilitando a manutenção ou criação de melhorias para a mesma para que sejam disponibilizadas à comunidade.

2 TRABALHOS RELACIONADOS

Como mencionado, este trabalho é relacionado ao proposto por Castro (2015), mas outros anteriores já demostraram a viabilidade da criação de um ambiente de simulação interligando a ferramenta SUMO com um sistema multiagentes Jason. Um desses foi descrito por Batista Júnior e Coutinho (2013), onde foi desenvolvido um sistema multiagentes afim de aumentar a eficiência de cruzamentos de vias arteriais. Além desse, foi descrito por Vincent Baines e Julian Padget (2014), uma interligação entre a ferramenta SUMO e também o Jason para o gerenciamento dos agentes presentes numa simulação de trânsito.

O principal trabalho relacionado a este foi descrito por Heijimeijer (2016), a interligação da ferramenta SUMO com as linguagens Jason e Cartago (JaCa) por meio de um *middleware* para que, dessa forma, seja possível o gerenciamento da simulação de um estacionamento. O desenvolvimento dos cenários do presente trabalho está baseado nos desenvolvidos por Heijimeijer (2016), que utilizou a ferramenta JaCamo, porém utilizando o *framework* JADE. O cenário de simulação escolhido foi uma adaptação do projeto *City Mobil* que está contido nos arquivos fontes da instalação da ferramenta SUMO. Foram definidas cores para representar cada intervalo da avaliação dos agentes e melhor visualização no SUMO.

Soares (2014) propôs a criação de um sistema multiagente desenvolvido com o *framework* JADE e utilizando o simulador SUMO para alcançar soluções de tráfego. Análises foram feitas utilizando mapas reais para otimizar a escolha de rotas, por exemplo.

O trabalho de Azevedo *et. al.* (2014), assim como Soares (2014), também apresentou a criação de um *Smart Parking* utilizando o *framework* JADE integrado ao simulador SUMO. Mas, seu foco foi sobre as relações dos agentes num ambiente com controle de semáforos.

No trabalho de Ducheiko (2018) é proposto um modelo de negociação descentralizado para o sistema multiagente do *smart parking*. Foram desenvolvidas um conjunto de regras para que fosse possível a tomada de decisões no sistema. Não sendo este do foco do presente trabalho, o modelo de negociação proposto funciona de forma muito mais simples.

Fernandes *et. al.* (2017) teve como foco de seu trabalho os veículos autônomos. Nele foram propostas soluções em *software* para a tomada de decisões

dos veículos autônomos. O projeto está definido em três partes. Na primeira devem ser definidos os planos de ações do agente através da linguagem de programação para agentes Gwendolen. Na segunda etapa é desenvolvido o ambiente do projeto desenvolvido em Java. Na terceira etapa são definidas propriedades dos agentes.

Mellado *et. al.* (2019) produziu um trabalho com foco na análise de sistemas multiagentes para o desenvolvimento de um *Smart Parking.* Foram estudas diversas propriedades do sistema como comunicação entre agentes, hierarquia etc para que fosse possível a comparação de características de diferentes soluções de um sistema de *Smart Parking*.

O trabalho de Botelho *et. al.* (2019) é uma proposta de implantação de uma arquitetura em *Raspberry Pi* e agentes ESP-12e para que seja possível extrair informações de um estacionamento inteligente. Essa proposta visa a integração dos com o *framework* JADE, também utilizado neste trabalho, com o sistema abordado.

Sakurada *et. al* (2019) propõe o desenvolvimento sistemas de estacionamento com agentes baseados em *Cyber-Physical Systems*. Além disso, é proposta a interligação desses sistemas com recursos físicos através da Internet das Coisas. Diferentemente dos trabalhos anteriores, que são focados apenas em carros, esse trabalho faz um estudo sobre carros e bicicletas.

Este trabalho se assemelha aos citados ao produzir um sistema de *Smart Parking* utilizando determinada ferramenta, porém neste optou-se pela utilização do *framework* JADE integrado ao simulador SUMO para a obtenção dos resultados. Para viabilizar essa integração, foi necessária a utilização da biblioteca TraaS para o desenvolvimento na linguagem Java, que também é um diferencial para este trabalho.

A implementação desse projeto de *Smart Parking* tem como objetivo reduzir o tráfego nas estradas dos centros urbanos, consequentemente, reduzindo as emissões de gás carbônico. Esse projeto cumpriria com as propostas previstas às cidades inteligentes, que é facilitar e ajudar na vida da população. Também tem como objetivo diminuir o nível de estresse dos motoristas nas estradas, pois todos os dias muitas pessoas gastam tempo desnecessário nas estradas em busca de uma vaga de estacionamento.

Já existem no mercado algumas ferramentas para a simulação de trânsito sejam elas pagas ou gratuitas. São elas:

- Aimsun Live: essa é uma ferramenta que permita a simulação de grandes áreas em tempo real. É possível a visualização das condições do tráfego para

que seja possível a implantação de estratégias de trânsito para o gerenciamento de congestionamentos. Dessa forma torna-se possível a disseminação de informações sobre o tráfego rapidamente ao público (AIMSUN, 2008);

- Paramics: essa é uma ferramenta de simulação de trânsito microscópica. Ela permite a visualização 3D do tráfego. Ela permite desde a construção, edição e visualização até a simulação de redes de tráfego para simulação (PARAMICS, 1990);
- PTV VISSIM: essa ferramenta permite uma simulação precisa dos padrões de trânsito. Ela é capaz de simular transportes privados, públicos, de mercadoria, ferroviário além de pedestres e ciclistas. Com a simulação da interação de todos esses componentes, essa ferramenta apresenta uma visão mais realista de um ambiente de trânsito (PTV VISSIM, 2018).

2.1 CONSIDERAÇÕES FINAIS

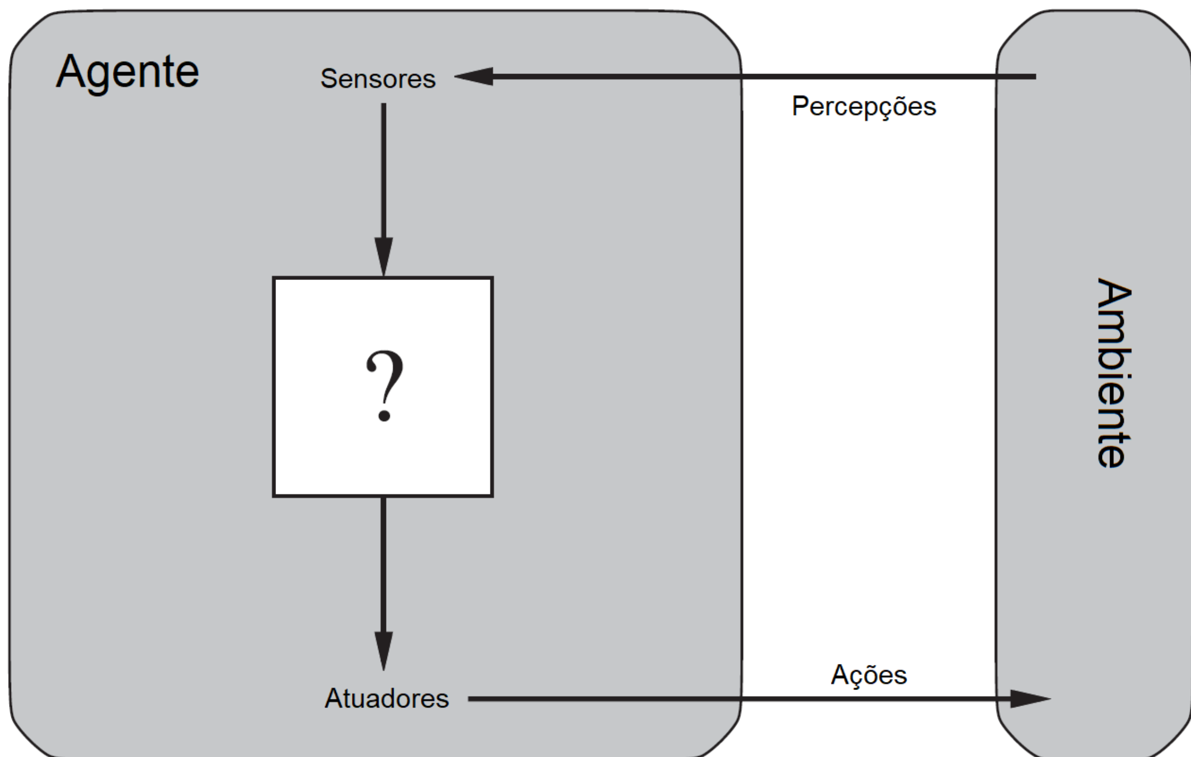
O projeto MAPS dispõe de vários trabalhos com propostas com diversos focos na área de *Smart Parking* como negociação, utilização da ferramenta JaCaMo etc. O que diferencia este trabalho é a interligação do *framework* JADE com o simulador SUMO para a construção de um *Smart Parking*. Para isso é necessária a utilização da biblioteca *TraaS* desenvolvida na linguagem Java.

3 AGENTES E SISTEMAS MULTIAGENTES

O termo “inteligente” na computação está muitas vezes ligado à Inteligência Artificial, onde estão contidos os Agentes Inteligentes. Neste trabalho, para a criação do estacionamento inteligente, serão utilizados agentes que, resumidamente, nada mais são que sistemas computacionais capazes de realizar ações autonomamente. Os agentes têm a capacidade de perceber seu ambiente, utilizar sensores e, sem a necessidade de intervenção humana, tomar decisões para a resolução de determinado problema (JUNCHEM E BASTOS, 2011).

Um exemplo, muito conhecido, de um agente é o de um aspirador de pó autônomo (ver Figura 1). Esse aspirador tem a capacidade perceber o ambiente e identificar se o piso está sujo ou não. Ele pode decidir se deve aspirar ou deslocar-se para algum lado do ambiente. Formas diferentes para a implementação desse agente podem ser escolhidas, mas no final, esse aspirador deve conseguir, autonomamente, realizar a limpeza de todo o ambiente.

Figura 1 - Um agente interagindo com o seu ambiente por meio de sensores e atuadores



Fonte: Adaptado de Russel (2010)

Fazendo uso de uma linguagem de computação qualquer orientada a objetos, cada função é designada a realizar determinada atividade em específico. Essa função, sempre realizará a mesma tarefa e permanecerá estática, pois ela não possui autonomia. Essa é uma grande diferença de um objeto para um agente. Um agente tem autonomia para perceber seu ambiente e realizar determinada ação por si só. Ele não necessita da intervenção de um usuário e realiza suas tarefas dinamicamente para que se consiga obter sucesso em seu objetivo.

SMA nada mais é do que diversos agentes atuando num mesmo ambiente (JUNCHEM E BASTOS, 2011). Num SMA todos os agentes coexistentes no ambiente devem ser capazes de realizar interação entre si para que possam atingir um objetivo final em específico. Então todos os agentes realizam individualmente suas percepções e ações, podendo trocar informações uns com os outros tornando o sistema muito mais dinâmico.

3.1 PRINCIPAIS CARACTERÍSTICAS DOS AGENTES

De acordo, com Wooldridge (2009), os agentes podem ser descritos utilizando as seguintes quatro propriedades: autonomia, habilidade social, reatividade e pro-atividade.

3.1.1 Autonomia

Essa é a principal característica que define os agentes. Um agente deve ser capaz de agir autonomamente, ou seja, não há intervenção humana durante sua atuação. Diferentemente dos sistemas computacionais comuns numa linguagem qualquer orientada a objetos, por exemplo. As ações dos sistemas são estáticas. Cada funcionalidade tem o objetivo de resolver determinada função em específico que não se altera. Um agente pode mudar suas ações constantemente. Dessa forma, um agente deve tomar suas decisões autonomamente a fim de conseguir atingir seu objetivo final (WOOLDRIDGE, 2009).

Num SMA, pode ser gerado um problema caso vários agentes tomem suas decisões autonomamente, pois vários conflitos podem ocorrer. Por isso, a cooperação torna-se necessária.

3.1.2 Habilidade Social

Num SMA, vários agentes coexistem num mesmo ambiente a fim de atingir um objetivo em comum. É um ponto crucial que tais agentes trabalhem em conjunto para que obtenham sucesso alcançando seu objetivo. Para tal feito, é necessário que tais agente consigam comunicar-se entre si para maior eficácia em determinadas tarefas (WOOLDRIDGE, 2009).

A comunicação pode ser feita através da passagem de *strings* por exemplo. Um agente pode enviar uma mensagem para outro e obter uma resposta podendo haver cooperação entre os agentes e tornando o ambiente muito mais dinâmico.

3.1.3 Reatividade

Um agente deve ser capaz de interagir com o seu ambiente. Como dito, os agentes possuem sensores para perceber o ambiente. Com base nessas percepções, o agente deve identificá-las e realizar determinada ação autonomamente (WOOLDRIDGE, 2009).

Essa pode tornar-se uma tarefa difícil sendo que o agente deve tomar as decisões autonomamente e que, às vezes, o ambiente pode ser muito complexo e/ou passar por alterações constantemente.

3.1.4 Pró-Atividade

As ações que um agente realiza não devem ser simplesmente aleatórias. Um agente deve almejar um objetivo a ser alcançado e suas ações devem ser baseadas para atingi-lo.

Essa é uma visão superficial sobre os agentes, mas de acordo com Wooldridge (1999), a definição de agentes vai mais a fundo. Estudos caracterizam como sistemas computacionais que possuem, além das características identificadas anteriormente, conceitos que são mais geralmente aplicados a humanos. Nesse meio de pesquisa, agentes são desenvolvidos com a implementação de propriedades humanas tais como conhecimento, crença, intenção e obrigação (SHOHAM, 1993).

Alguns estudos mais aprofundados sobre agentes chegaram a desenvolver agentes que possuem emoções.

Além das principais características mencionadas anteriormente sobre os agentes. Existem vários outros atributos que podem ser estudados. Por exemplo:

- Veracidade: um agente nunca deverá, propositalmente, comunicar uma informação falsa;
- Benevolência: assume que os agentes não deverão ter conflito para ter sucesso em suas metas. Cada agente irá realizar a tarefa à qual a ele foi agregada;
- Mobilidade: uma agente deve ter capacidade de se movimentar pela rede de computadores;
- Racionalidade: assume que um agente agirá de forma que ele consiga atingir suas metas e não tomará decisões que impedirão de alcançá-las e menos alguma ação vá contras suas crenças.

Humanos possuem crenças e desejos e, com base em tais, eles podem tomar diversas decisões a fim de obter sucesso em algum objetivo. Por exemplo, uma pessoa dirigindo um veículo pode optar por determinado caminho para que ela consiga chegar em seu destino mais rápido. Essa pessoa pode conhecer algum atalho ou saber que o tráfego está congestionado em determinada área e, dessa maneira, optar por um caminho mais viável para chegar ao seu destino. Essas noções, no ramo da psicologia, são chamadas de noção intencionais (DENNETT, 1987).

Um sistema em primeira instância pode possuir crenças e desejos, mas não pode possuir crenças e desejos sobre crenças e desejos. Já um sistema em segunda instância possui crenças e desejos e também crenças e desejos sobre outras crenças e desejos (DENNETT, 1987).

Um agente pode possuir atitudes de informação, que são o conhecimento e crenças, e também as pró-atitudes, que são desejos, obrigações, intenções, escolhas etc. As atitudes de informação são as noções que o agente tem do ambiente ao qual ele pertence. Já as pró-atitudes são as que guiarão o agente na tomada de decisões para realizar determinada ação (WOOLDRIDGE, 1999).

3.2 ARQUITETURA DOS AGENTES

A arquitetura exemplifica como ele pode ser decomposto em vários módulos componentes que devem interagir entre si. Esses módulos com suas intenções irão receber as informações das percepções obtidas do ambiente e definirão como o agente deverá agir. A forma de como as percepções do agente serão tratadas para definir a ação dependerá da arquitetura que o agente estará utilizando (WOOLDRIDGE, 1999).

Geralmente a arquitetura dos agentes são classificadas em dois grupos: arquitetura reativa e arquitetura cognitiva. No entanto, também existe uma arquitetura híbrida que reúne propriedades das arquiteturas reativa e cognitiva.

3.2.1 Arquitetura Reativa

Estruturalmente, os agentes reativos são simples. Nessa arquitetura, os agentes não são capazes de realizar raciocínios complexos. O agente reativo toma suas decisões baseando-se somente no estado do ambiente atual. Ele não possui um histórico, onde é possível que se aprenda com ações anteriores. Um agente reativo deve ser capaz de realizar ações isoladas para obter sucesso em seu objetivo. Seu comportamento inteligente está baseado nas ações que ele produz sob as percepções obtidas do ambiente (WOOLDRIDGE, 1999).

3.2.2 Arquitetura Cognitiva

Os agentes cognitivos (ou agentes deliberativos) têm uma representação simbólica de seu ambiente e tem informações sobre o mesmo em sua base de conhecimentos. Esses possuem histórico de suas ações já realizadas. Dessa forma, as ações futuras que o agente cognitivo tomar deverão estar baseadas em suas ações já realizadas. Portanto um agente dessa arquitetura consegue tomar ações fundamentado em sua base de conhecimentos (WOOLDRIDGE, 1999).

De acordo com Wooldridge (1999), para desenvolver um agente nessa arquitetura, pelo menos dois importantes problemas deverão ser resolvidos:

1. O ambiente no qual o agente estará contido deverá ser traduzido, de forma simbólica, para que o mesmo se torne usável;
2. Deve-se achar um meio de como representar simbolicamente as informações de um ambiente complexo de maneira que um agente possa ter determinado raciocínio baseado nessa informação em bom tempo tornando-se usável.

3.3 AMBIENTES

Os ambientes proveem informações às quais o agente pode percebê-las, por meio de seus sensores, e realizar determinada função baseado nelas.

Os ambientes podem ser classificados baseado em suas propriedades. De acordo com Russel e Norvig (2010) são elas:

- Acessível ou Inacessível: um ambiente acessível é aquele no qual o agente consegue obter informações sobre o mesmo. Caso não seja possível, ele é inacessível
- Estático ou Dinâmico: Um ambiente estático é aquele que permanece inalterado até que o agente execute alguma ação. Exemplo: tabuleiro de xadrez. Um ambiente dinâmico pode alterar seu estado independente das ações do agente. Exemplo: ruas de uma cidade;
- Determinístico ou Não-determinístico: num ambiente determinístico, tem-se conhecimento do estado final de ambiente depois de determinada ação de um agente. Num ambiente não-determinístico não se sabe o estado em que o ambiente se encontrará depois da ação do agente;
- Discreto ou contínuo: Um ambiente discreto é aquele que possui um número finito de estados possíveis. Num ambiente contínuo não é possível determinar o número de estados possíveis;
- Episódico ou Não-episódico: Num ambiente episódico, a experiência de um agente é dividida em episódios. Em cada episódio o agente perceberá seu ambiente

e realizará suas ações. Nesse tipo de ambiente, as ações tomadas pelo agente no episódio atual não farão efeito em episódios futuros assim como o episódio atual não é dependente das ações tomadas em episódios anteriores.

3.4 SISTEMA MULTIAGENTES

Um Sistema-Multiagente (SMA) é constituído de um ambiente que pode ter um espaço definido ou não. Nesse sistema não existe apenas um, mas vários agentes dispersos num mesmo ambiente. Esses agentes agem autonomamente, porém devem ter a capacidade de se relacionar um com os outros a fim de cooperar para obter sucesso de um objetivo final (FERBER, 1996).

A concepção de um SMA apresenta as seguintes características (ALVARES E SICHMAN, 1997):

- Os agentes num SMA devem ser desenvolvidos tendo em mentes eles não devem resultar em apenas um sistema capaz de resolver somente um determinado problema alvo. Nessa concepção os agentes devem ser capazes de resolver problemas que também se encontram em outros contextos;
- Essa mesma concepção também se aplica nas interações entre os agentes. As interações não devem ser desenvolvidas com o objetivo de resolver um problema alvo, mas criando protocolos de interação genéricos que possam ser reutilizados em outros contextos;
- Esse modelo também deve ser aplicado ao projeto da organização. Deve-se selecionar as funcionalidades que realmente deverão ser implementadas aos agentes;
- Os agentes têm o dever de fazer as interações, de acordo com o protocolo, a fim de atingir determinado objetivo. Dessa forma, não é possível ter controle do problema, pois ele se encontra descentralizado entre os agentes.

3.4.1 Sistemas Multiagentes Reativos

O conceito de agentes reativos já foi estudado anteriormente. Esse conceito também pode ser aplicado aos SMA. Vários agentes desenvolvidos deverão coexistir num mesmo ambiente. Agentes reativos não conseguem resolver problemas demasiadamente complexos e que exigem muito raciocínio. Eles não possuem um ambiente definido e nem armazenam um histórico das ações realizadas anteriormente. Eles tomam suas ações baseado somente em suas percepções do estado atual do ambiente (DEMAZEAU, 1994).

Um SMA reativo poderia ser utilizado, por exemplo, na busca de informações altamente específicas numa rede de computadores. Tratando-se de informações específicas, os agentes não precisarão realizar tarefas complexas. E contanto que as informações sejam independentes, não será necessário que os agentes tenham um histórico de suas ações já realizadas.

3.4.2 Sistemas Multiagentes Cognitivos

Como estudado anteriormente, os agentes cognitivos mantêm explicitamente uma demonstração simbólica do ambiente em que coexistem. Esses agentes também armazenam um histórico das ações já realizadas para que essas possam influenciar na tomada de decisões futuras. Tratando-se de um SMA, esses agentes são capazes de realizar interações uns com os outros. Essa interação é feita por troca de mensagens. Como existem diversos agentes, a execução do processo se faz de forma mais dinâmica já que podem existir agentes que resolvem conflitos de negociação, que, com representações mútuas, criam um esquema de alocação de tarefas etc (DEMAZEAU, 1994).

Um exemplo de SMA cognitivo é sistema de *Smart Parking*. Um sistema que gerencia vagas de estacionamento, onde carros são agentes em busca de uma vaga. Esses agentes devem ter conhecimento do seu ambiente e possuem um histórico de suas ações já realizadas. Esses agentes devem comunicar-se e cooperar para que consigam alcançar o seu objetivo: conseguir uma vaga de estacionamento.

3.5 COMUNICAÇÃO ENTRE AGENTES

Num SMA, os agentes devem cooperar a fim de alcançar determinado objetivo. Dessa forma, eles podem interagir entre si por meio de mensagens. Existem algumas linguagens para a formulação de tais mensagens (BELLIFEMINE *et. al.*, 2007).

- KQML (*Knowledge and Query Manipulation Language*): é uma linguagem que trabalha com ações como atos de fala como *ask-if, tell perform, reply* etc. Ela define uma cobertura para formatar as mensagens que determinar o significado resultante da mensagem. Ela não está preocupada com o conteúdo da mensagem, mas sim com a formatação da informação para o entendimento do conteúdo (MAYFIELD *et. al.,* 1996);
- KIF (*Knowledge Interchange Format*): é uma linguagem que tem como objetivo representar o conhecimento sobre um domínio de discurso em específico. Ela foi criada para definir o conteúdo das mensagens criadas em KQML (GENESERETH e KETCHPEL, 1994).
- ACL (*Agent Communication Language*): é uma linguagem, semelhante ao KQML, explicada mais profundamente no decorrer deste trabalho. Ela é uma linguagem de comunicação externa e não requer a utilização de linguagem alguma para o conteúdo (LABROU *et. al.*, 1999).

3.6 CONSIDERAÇÕES FINAIS

Neste capítulo foram abordadas diversas características sobre os agentes. Conhecimentos sobre seus tipos, ambientes, arquitetura etc são importantes para o entendimento deste trabalho que utiliza de um sistema multiagente para a simulação do *Smart Parking*.

4 SMART CITY E SMART PARKING

Cidades inteligentes ou *Smarty Cities* tem sido frequentemente descritas como constelações de dispositivos de várias escalas interligados por meio de uma rede de computadores que provêm dados continuamente a respeito do movimento de materiais e das pessoas sobre suas decisões em meio à cidade (BATTY *et. al.*, 2012).

O conceito de uma cidade inteligente começou a ser utilizado nos últimos anos. Surgiu com a ideia de unir a alta tecnologia em conjunto com a comunicação em rede para que se pudesse aprimorar o funcionamento de uma cidade em vários aspectos, tais como eficiência, melhorar a competitividade, facilitar a vida dos cidadãos etc.

Mesmo com tecnologia avançada, desenvolver uma cidade inteligente torna-se uma tarefa muito complexa, visto que diversas áreas de uma cidade devem ser abordadas. Por isso, dentro de uma *Smart City*, de acordo com Batty *et. al.* (2012), existem várias subáreas de estudo, como mostradas a seguir:

- *Economy*: inteligência sendo inserida nas decisões econômicas de uma cidade;
- *People*: inteligência participando da vida pública das pessoas;
- *Governance*: participação da inteligência nas decisões públicas, tornar o governo mais transparente etc;
- *Mobility*: a inteligência podendo lidar com questões de congestionamento, acessibilidade, transportes sustentáveis etc.
- *Environment*: inteligência trabalhando para e proteção do meio ambiente por exemplo;
- *Home*: inteligência dentro das casas dos cidadãos de uma cidade.

O enfoque deste trabalho está na *Smart Mobility*. Vários problemas, tais como congestionamentos e veículos altamente causadores de poluição, estão presentes todos os dias na vida de cidadãos de uma cidade comum. Numa *Smart City*, tem-se como objetivo tornar todo esse processo mais eficiente a fim de tornar a vida das pessoas mais feliz e menos estressante. Vários campos de estudo como carros autônomos, *Smart Parkings* etc, têm sido abordados.

4.1 SMART PARKING

Os estacionamentos inteligentes ou *Smart Parkings* estão contidos no campo da *Smart Mobility* de uma cidade inteligente. São utilizadas diversas tecnologias como sensores de veículos, gerenciadores de estacionamentos, sistema autônomo para o pagamento de *tickets* a fim de melhorar o funcionamento das estratégias de mobilidade de uma cidade inteligente (DI NAPOLI *et. al.,*2014).

Um dos grandes problemas encontrados nas ruas dos grandes centros urbanos é para encontrar uma vaga de estacionamento disponível. Quando um motorista deseja uma vaga de estacionamento em determinada área, ele não sabe onde exatamente ele encontrará um local para alocar seu carro. Com isso, diversos motoristas acabam trafegando desnecessariamente pelas ruas da cidade em busca dessa vaga. Dessa forma, o número de carros nas ruas aumenta causando congestionamentos, combustível é gasto prejudicando o meio ambiente, além do estresse causado aos motoristas que poderá ocasionar acidentes de trânsito.

A seguir são descritos os dois principais projetos envolvendo *Smart Parkings* utilizados e seguidos pelo grupo de pesquisa MAPS. O primeiro (Di Napoli *et. al. 2014)* foi utilizado como base para a criação do projeto de Castro (2015). O segundo foi utilizado por trabalhos do grupo pela sua eficiência e disponibilização do código fonte.

4.1.1 Projeto Proposto por Di Napoli *et. al.* (2014)

Na Itália, Di Napoli *et. al.* (2014) fez uma proposta de um sistema de *Smart Parking* baseado em negociações. Neste *Smart Parking* existem dois tipos de agentes: o agente motorista (*Driver*) e o agente gerenciador (*Manager*). Existe apenas um *Manager* no sistema e ele é responsável pelo gerenciamento de todas as vagas de estacionamento do *Smart Parking*. O agente *Driver* representa todos os outros agentes do sistema os quais eles apenas requisitam uma vaga de estacionamento ao *Manager*.

Um *Driver* pode requisitar uma vaga de estacionamento ao *Manager* passando a ele diversas propriedades, tais como região da vaga, preço que se dispõe

a pagar, tempo que ficará na vaga etc. O *Manager* é responsável por fazer a negociação dessa vaga. Ele avalia as informações recebidas e também deve ser capaz de lidar com conflito quanto mais de um *Driver* requisita uma mesma vaga. Depois da negociação o *Manager* deve responder ao *Driver* se ele conseguir ou não a vaga. Caso positivo, o *Driver* poderá dirigir-se à vaga para seu carro. Caso negativo, o *Driver* pode requisitar novamente uma outra vaga.

4.1.2 Projeto Proposto por Castro (2015)

O projeto desenvolvido por Castro (2015) segue a mesma linha de pensamento do desenvolvido por Di Napoli *et. al.* (2014). Um motorista pode requisitar uma vaga de estacionamento para um agente gerenciador e este negocia a vaga para que se decida se o motorista conseguirá utilizá-la ou não.

Castro (2015) utilizou a ferramenta JaCaMo para o desenvolvimento de seu sistema. A ferramenta JaCaMo consiste na junção de três ferramentas:

- Jason: ferramenta utilizada para a modelagem dos agentes;
- Cartago: ferramenta utilizada para o desenvolvimento do ambiente onde os agentes coexistirão;
- Moise: ferramenta para criar as regras do ambiente criado.

Um problema que ocorre nos dois projetos vistos diz respeito a que os dois sistemas são centralizados. O agente *Manager* é responsável por realizar negociação de todas as vagas de estacionamento que forem requisitadas. Dessa forma, caso ocorra algum problema nesse agente, todo o sistema será comprometido.

4.1.3 Considerações Finais

Este trabalho e o proposto por Castro (2015) seguem um objetivo em comum, o desenvolvimento de um sistema de *Smart Parking*, mas indo por caminhos distintos. Enquanto Castro (2015) optou pelo desenvolvimento utilizando a ferramenta JaCaMo, este trabalho utiliza o *framework* JADE para a construção dos agentes, suas

interações e gerenciamento da comunicação que ocorre entre eles. Além disso, é feita a integração do JADE com o simulador SUMO para que seja possível fazer análise visual sobre a simulação do estacionamento.

5 FRAMEWORK JADE

Existem diversos *frameworks* para o desenvolvimento de projetos orientados a agentes. O escolhido para este trabalho foi o JADE (Java Agent DEvelopment, 2000). JADE é um framework para o desenvolvimento de agentes e está nos padrões FIPA e foi desenvolvido pela Telecom Itália em parceria com a Universidade de Parma. JADE também é um projeto *open source* com licença LGPL (*Lesser General Public Licence*).

O JADE é totalmente escrito em Java e suas funcionalidades atuam na camada de *middleware*. Ele pode ser executado em diversos tipos de dispositivos que podem variar desde servidores até *smartphones*.

Algumas das características do JADE são:

- É um sistema totalmente distribuído habitado por agentes. Cada agente no JADE é executado separadamente por *threads;*
- Totalmente de acordo com os padrões FIPA;
- Possui transporte de mensagens assíncronas;
- Os agentes possuem um ciclo de vida. Quando um agente é criado, ele obtém um identificador único para referenciá-lo;
- Possui suporte para *agente mobility*. Um agente consegue se locomover entre processos. Essa migração é transparente;
- Possui um conjunto de ferramentas numa interface gráfica para que se faça monitoramento do funcionamento dos agentes em tempo real;
- Suporte para ontologias e linguagens de conteúdo como, por exemplo, o XML;
- Uma biblioteca de protocolos de interação com padrões de comunicação para serem utilizados com os agentes;
- Integração com ferramentas de desenvolvimento web, tais como JSP, servlet etc.

O Código 1 mostra a criação de um agente utilizando JADE:

Código 1 - Desenvolvimento de um agente em JADE

```
1 public class Vendedor extends Agent {
2
3     Vendedor(){ }
4
5     @Override
6     public void setup(){
7
8     }
9 }
```

Fonte: Autoria Própria

No Código 1 é mostrado o desenvolvimento simples de um agente. A função *setup*, linha 6, é chamada quando um agente instanciado.

Código 2 mostra a inicialização de agentes utilizando o framework JADE na IDE Netbeans:

Código 2 - Criação de agentes utilizando o framework JADE

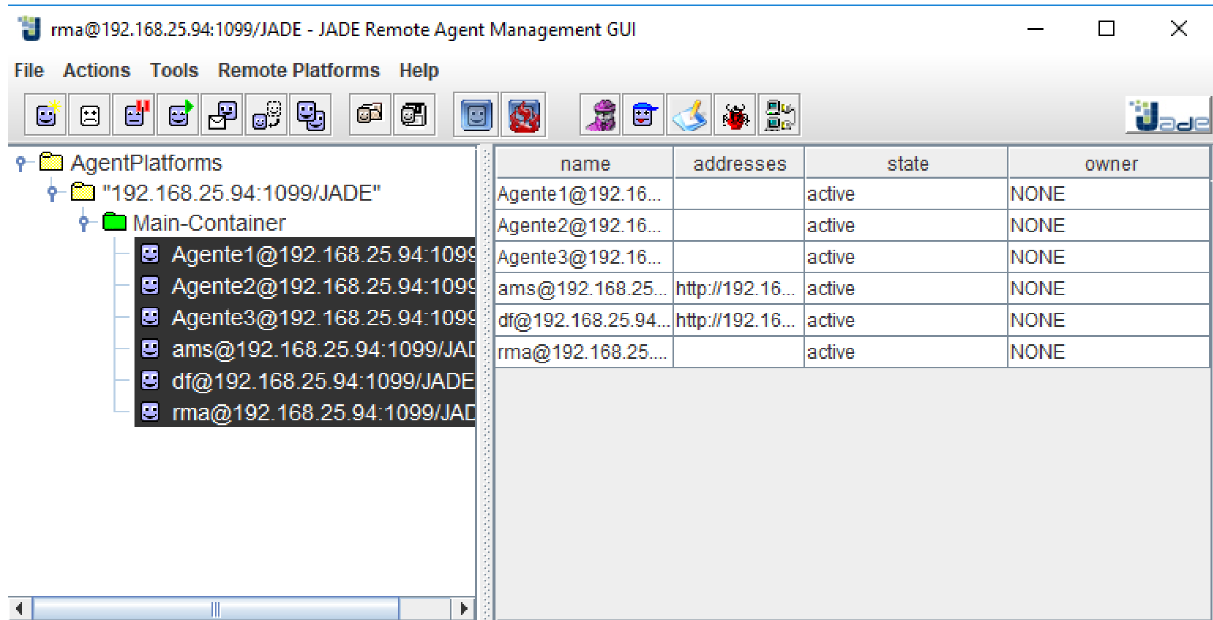
```
1         Runtime runtime = Runtime.instance();
2         Profile profile = new ProfileImpl();
3         ContainerController controller = runtime.createMainContainer(profile);
4         AgentController rma;
5         AgentController agente1;
6         AgentController agente2;
7         AgentController agente3;
8
9         try{
10            rma = controller.createNewAgent("rma", "jade.tools.rma.rma", null);
11            rma.start();
12
13            agente1 = controller.acceptNewAgent("Agente1", new Vendedor());
14            agente1.start();
15
16            agente2 = controller.acceptNewAgent("Agente2", new Vendedor());
17            agente2.start();
18
19            agente3 = controller.acceptNewAgent("Agente3", new Comprador());
20            agente3.start();
21         } catch(Exception e){
22            System.out.println(e);
23         }
```

Fonte: Autoria Própria

Nesse trecho de código foram declarados quatro agentes do tipo *AgentController* nas linhas de 4 a 7. Os agentes no JADE existem dentro de containers e dentro deles eles realizam suas atividades. Foi, então, criado o *container* controller e, dentro dele, foram criados quatro agentes: agente1, agente2, agente3 e rma. O agente rma (*Remote Monitoring Agent*) é um agente específico do JADE, conforme

Figura 2. Ele é responsável pela interface gráfica do JADE e também pela visualização dos agentes dentro dos containers e dos próprios containers.

Figura 2 - Interface gráfica do JADE



Fonte: Autoria Própria

O JADE é um sistema distribuído e os agentes que coexistem num mesmo ambiente tem as mensagens assíncronas como forma de comunicação. A linguagem ACL (*Agent Communication Language*) é responsável pela estrutura das mensagens. Esse tipo de mensagem possui propriedades como tempo limite de aguardo e contexto da mensagem. Os agentes possuem um nome único pelo qual ele é reconhecido pelos outros componentes do sistema. Por esse nome é possível realizar a identificação e realizar o envio e recebimento de mensagens.

A linguagem ACL está de acordo com as especificações FIPA (2002) a qual possui uma série de parâmetros para e estrutura de uma mensagem. Esses parâmetros servem para aumentar e eficiência na troca de mensagens entre agentes numa aplicação. Estão disponíveis para uso diversos parâmetros, mas serão citados a seguir alguns dos principais utilizados:

- *Sender*: especifica o agente que irá enviar a mensagem;
- *Receiver*: especifica o agente que irá receber a mensagem;
- *Content*: descreve o conteúdo da mensagem que será enviada;
- *Language*: especifica o idioma do conteúdo a mensagem;
- *Performative*: especifica o tipo da mensagem enviada.

Este último parâmetro citado, Performative, descreve qual é o tipo da mensagem ACL que é enviada. Baseada nessa informação, é tomada determinada ação. Existe uma vasta quantidade de tipos de mensagem que pode ser enviada e esses tipos são descritos pela FIPA Communicative Act Library Specification (2000). Não é necessário demonstrar todas as performativas da especificação FIPA, desta forma, a seguir serão descritas algumas das performativas presentes neste trabalho:

- *Inform*: Descreve determinada informação a outro agente. O agente emissor deve estar tomando a mensagem enviada como sendo verdadeira. Essa é a performativa mais utilizada pela especificação FIPA;
- *Request*: O agente emissor solicita determinada ação ao agente receptor. Geralmente, ao utilizar esse tipo de performativa, o agente emissor está esperando alguma resposta. Dessa forma, poderá haver continuação da comunicação entre os agentes;
- *Agree*: essa performativa aceita uma mensagem que geralmente foi recebida por outra da performativa *request*.
- *Cancel*: assim como a performativa *Agree*, essa também é uma continuação de uma mensagem com performativa *request*, porém essa é do tipo de cancelamento;
- *Failure*: Também uma continuação de uma mensagem de performativa *request*, essa performativa avisa uma falha ocorrida na comunicação.

Código 3 - Criação e envio de mensagem em JADE

```
 1 @Override
 2     public void setup(){
 3        addBehaviour(new VenderCarro());
 4     }
 5
 6    private class VenderCarro extends OneShotBehaviour {
 7        @Override
 8        public void action() {
 9            ACLMessage mensagem = new ACLMessage(ACLMessage.INFORM);
10            mensagem.addReceiver(new AID("Comprador", AID.ISLOCALNAME));
11            mensagem.setLanguage("portugues");
12            mensagem.setContent("Você deseja comprar o(a) " +
13                    carros.get(0).getNome() + "?");
14            send(mensagem);
15        }
16    }
```

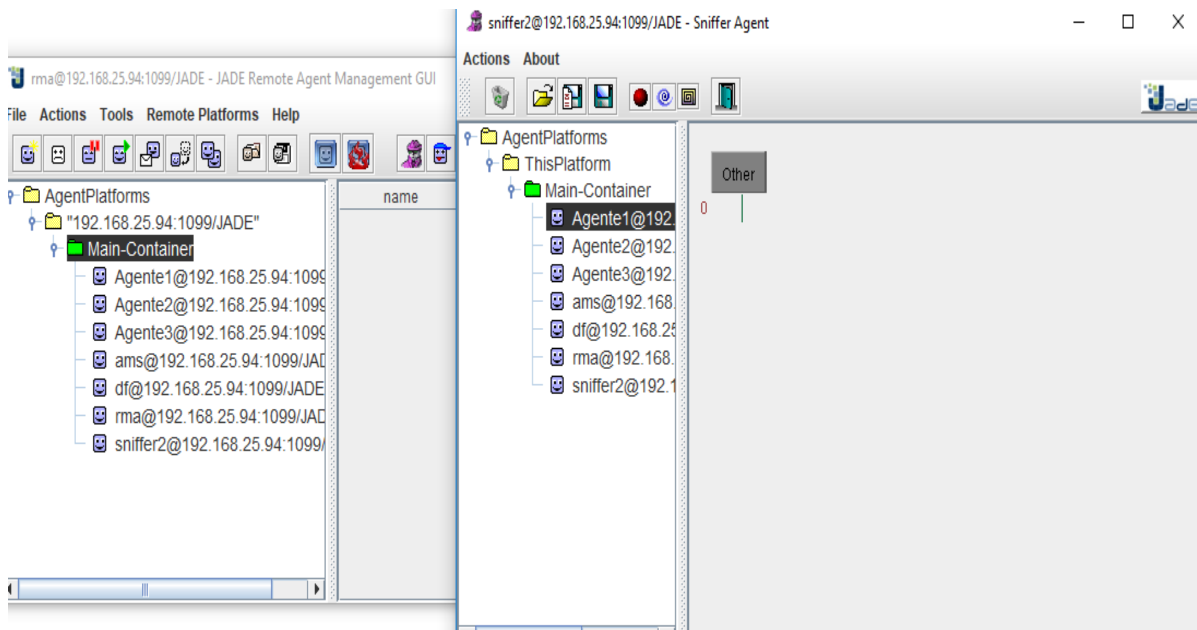
Fonte: Autoria Própria

No Código 3 é mostrada a criação de uma mensagem ACL. Conforme a linha 9, é criada uma mensagem de performativa *Inform*. As três linhas seguintes definem,

respectivamente, quem irá receber a mensagem, seu idioma e o conteúdo da mesma. Enfim, pode-se enviar a mensagem conforme método *send* da linha 14.

É possível fazer a visualização do envio e recebimento de mensagens entre os agentes na interface gráfica do próprio JADE. Para isso deve-se utilizar o *Sniffer*. Ele possibilita a observação da troca de mensagens entre agentes em tempo real de execução do SMA.

Figura 3 - Utilização do Sniffer para a visualização de troca de mensagens entre os agentes

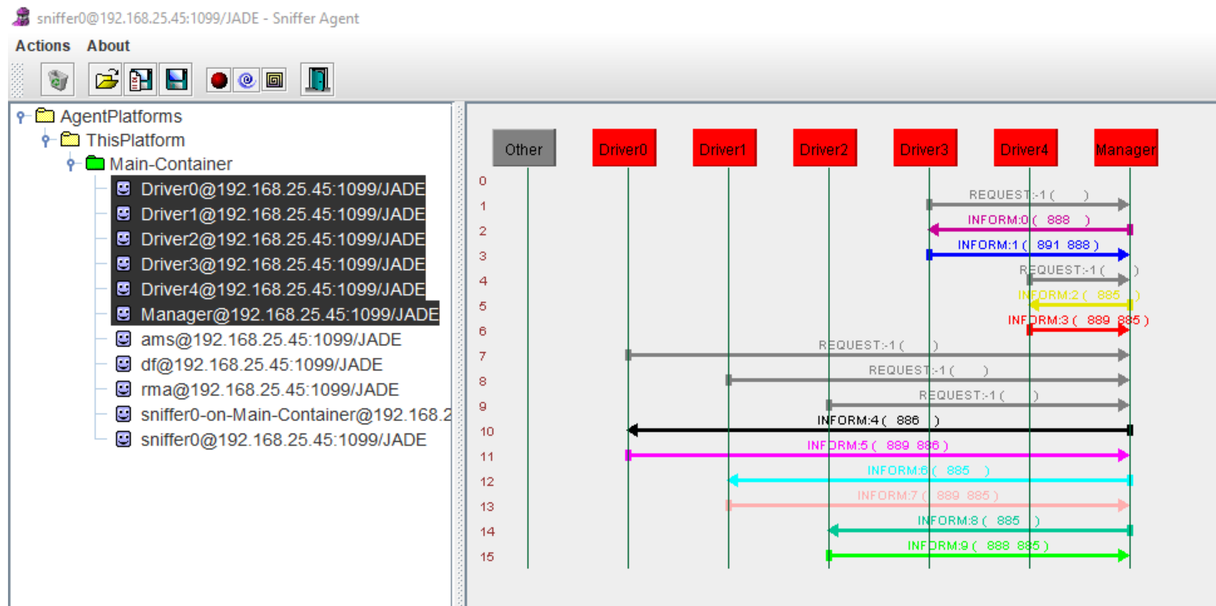


Fonte: Autoria Própria

Para que um agente seja capaz de reagir a percepções do ambiente, existem os comportamentos do mesmo. Os comportamentos primitivos podem ser realizados uma única vez (*One Shot Behaviour*) e os que estão num ciclo, podem ser realizados diversas vezes (*Cyclic Behaviour*). Existem ainda outros mais avançados do JADE.

Utilizando o *sniffer* é possível observar, em tempo real da simulação, como ocorre a comunicação entre cada um dos agentes selecionados. A Figura 4 demonstra a comunicação entre 6 agentes numa simulação.

Figura 4 - Utilização do sniffer na comunicação de 5 agentes



Fonte: Autoria própria

Nesse exemplo foram criados cinco agentes *driver* e um agente *manager*. Todos os agentes *driver* conversam somente com o agente *manager* enquanto o *manager* conversa com todos os agentes *driver*. Na *Figura 4* as setas demonstram quem são os agentes emissor e receptor da comunicação e, acima delas, está descrita qual é a performativa que foi utilizada. Todas as comunicações entre os agentes se iniciam com um *driver* enviando uma mensagem do tipo *request* para o *manager*. Então inicia-se uma conversa entre eles com mensagens de performativa *inform*.

6 VISÃO GERAL DO SUMO

Até agora, foram mostradas as ferramentas que atuam na parte lógica do projeto. Não há uma representação visual dos agentes no ambiente. Dessa forma, existem ferramentas que trabalham de forma a tornar possível a simulação de um projeto de um ambiente de trânsito. Assim, torna-se mais fácil verificar se determinado projeto é viável ou não. Neste capítulo, será analisado o funcionamento de algumas ferramentas de simulação de trânsito tal como a que será utilizada nesse projeto.

Existem três tipos de simuladores de trânsito:

- Macroscópico;
- Microscópico;
- Sub-microscópico.

A visualização microscópica permite uma observação mais detalhada ao objetivo desejado: um ambiente de estacionamento. Dessa forma, esse foi o tipo de simulador escolhido para esse projeto.

Algumas ferramentas de simulação de trânsito foram apresentadas no capítulo sobre trabalhos relacionados, porém nenhuma delas é gratuita. Existe a ferramenta SUMO, que será melhor apresentada na seção 6.1, que se tornou popular e de código fonte aberto. Dessa forma, a ferramenta vem ganhando mais recursos também providos por sua comunidade. Devido a esses fatos, essa será a ferramenta utilizada nesse projeto.

6.1 SUMO SIMULATOR

O SUMO (***S****imulation of* ***U****rban* ***MO****bility*) é uma ferramenta de simulação de trânsito microscópica gratuita e de código fonte aberto. Seu desenvolvimento teve início no ano de 2000. A razão da criação dessa ferramenta foi pode prover uma aplicação de código fonte aberto para apoiar a comunidade de pesquisa de simulação de tráfego para que eles conseguissem implementar seus próprios algoritmos. Ela permite a construção de simulações simples, como apenas um carro andando sob uma faixa de uma estrada, até simulações mais complexas intermodal, que permite carros, ônibus, bicicletas etc, trafegando sob uma mesma simulação de trânsito.

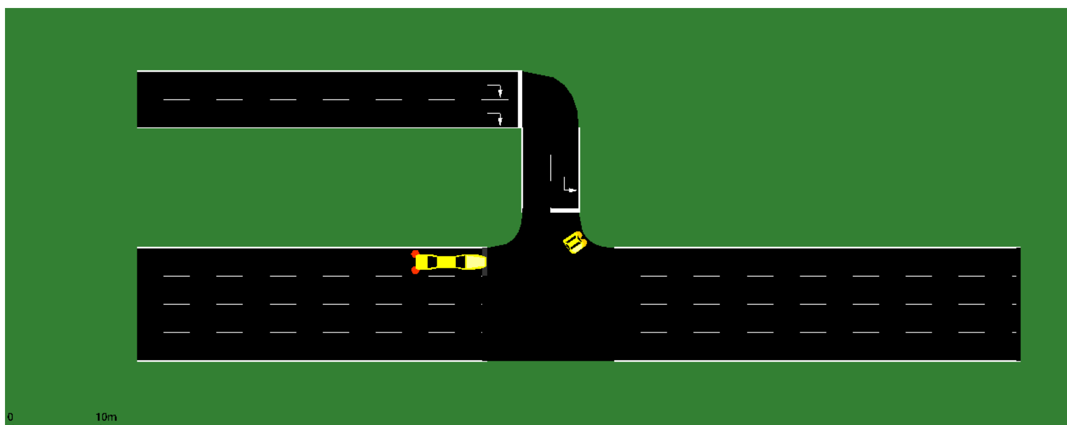
A ferramenta SUMO disponibiliza funcionalidades como a criação de ambientes para a simulação microscópica de veículos, pedestres, transportes públicos, bicicletas etc, que interagem entre si numa rede de trânsito. É possível a criação de modelos de trânsito e importados para um arquivo reconhecível pelo SUMO pela ferramenta NETCONVERT. Além disso, é possível o *download* de diversos arquivos com redes de trânsito prontas para a simulação, alguma delas representando com fidelidade partes do sistema de trânsito de cidades reais.

Para a criação de cenários de simulação no SUMO são necessárias a inclusão de rotas (*routes*) e uma rede (*network*). As rotas deverão ser compostas pelos seguintes elementos:

- Nós (*nodes*): os nós representam as extremidades das pistas. Eles recebem atributos que mostram onde, na simulação, determinada pista deverá ser gerada;
- Vias (*edges*): as vias são as estradas criadas entre os nós. Elas recebem a informação de que nó partirão até que nó devem chegar para que seja criada a via entre eles;
- Tipos (*types*): os tipos são responsáveis por informar os tipos das vias e dos veículos da simulação;
- Conexões (*connections*): as conexões são responsáveis por criar as conexões entre as faixas de uma via na simulação.

A Figura 5 mostra um exemplo simples do funcionamento de uma rede de trânsito utilizando a ferramenta SUMO.

Figura 5 - Exemplo de simulação de trânsito no SUMO



Fonte: Autoria própria

A criação de todos os nós, vias, tipos, conexões e rotas se dá por meio de arquivos XML. O exemplo da Figura 5 apresenta uma rede de trânsito simples formada por três faixas de pistas e dois veículos: um carro a um ônibus.

O Código 4 apresenta a criação dos nós dessa simulação.

Código 4 - Exemplo da criação de nós da simulação

```
1 <nodes>
2         <node id="n1" x="0" y="20" />
3         <node id="n2" x="0" y="0" />
4         <node id="n3" x="50" y="20" />
5         <node id="n4" x="50" y="0" />
6         <node id="n5" x="100" y="0" />
7 </nodes>
```

Fonte: Autoria própria

A criação dos nós da simulação é feito pela marcação *node*. Os nós representam as extremidades das pistas criadas. Dessa forma, a marcação deve receber um *id* único e as posições de x e y na qual o nó estará presente na simulação. No exemplo apresentado, existem cinco nós, ou seja, cinco pontos de extremidade nessa rede de vias.

Código 5 -Exemplo da criação de vias da simulação

```
1 <edges>
2         <edge from="n1" to="n3" id="1to3" type="type1" />
3         <edge from="n2" to="n4" id="2to4" type="type2" />
4         <edge from="n3" to="n4" id="3to4" type="type3" />
5         <edge from="n4" to="n5" id="4to5" type="type4" />
6 </edges>
```

Fonte: Autoria própria

O Código 5 apresenta a criação de vias da nossa aplicação. As vias representam as ligações entre os nós que, graficamente, serão as pistas. Para a criação das vias deve-se declarar os elementos *edges*. Deve-se, assim como nos nós, definir um nome único para ser a identidade de cada uma. Devem ser definidos os pontos de origem de final de cada via, para isso deve-se referenciar dois *nodes* criados anteriormente para que seja criada uma via entre eles. Numa *edge* pode-se também definir seu tipo, que será melhor exemplificado a seguir.

Código 6 - Exemplo de códigos dos tipos na simulação

```
1 <types id="type1" priority="1" numLanes="2" speed="8" />
2         <type id="type2" priority="1" numLanes="4" speed="6" />
3         <type id="type3" priority="2" numLanes="2" speed="5" />
4         <type id="type4" priority="1" numLanes="4" speed="8" />
5 </types>
```

Fonte: Autoria própria

No Código 6 é mostrado como é feita a criação dos tipos em nossa simulação. Nessa etapa, foram criadas especificações sobre as vias já anteriormente criadas. No caso desta simulação, para cada elemento *type*, foi criado um identificador único para representar cada tipo. Nele foi definido as *numLanes*, que são a quantidade de faixas que terá determinada vida, a velocidade que os veículos poderão trafegar sob a mesma. Foi definido o atributo *priority* que indica o grau de prioridade determinada via. Como é possível observar na Figura 5, um dos veículos da simulação está parado esperando enquanto o outro segue seu trajeto. Isso ocorre porque a via do veículo em movimento tem maior prioridade ao veículo parado.

Código 7 - Exemplo da criação de conexões da simulação

```
1 <connections>
2         <connection from="3to4" to="4to5" fromLane="1" toLane="3" />
3         <connection from="2to4" to="4to5" fromLane="3" toLane="3" />
4 </connections>
```

Fonte: Autoria própria

O Código 7 mostra como são criadas as conexões entre as estradas. Para criá-las, deve-se usar a palavra de marcação *connection*. Nela deve-se especificar, pelos atributos *from* e *to,* as vias de origem e destino ao qual se deseja criar a conexão e também a faixa a qual o veículo se encontrava e para que faixa ele deverá ir, definidas pelos atributos *fromLane* e *toLane*, respectivamente. Na simulação de exemplo, os dois veículos devem ocupar uma mesma faixa em sua estrada de destino. Como citado no *Código 6*, um dos veículos teve prioridade sobre o outro devido ao seu atributo *priority*.

Até o momento, foram criados arquivos com informações separadas sobre as vias de nossa simulação. Deve-se então entrelaça-los para que se possa criar a rede de estradas da simulação. Para isso, existe uma ferramenta chamada NETCONVERT do SUMO que tem esse objetivo. Ela recebe as informações de nós, vias, tipos e

conexões para criar uma rede pronta para receber um fluxo de veículos gerando, dessa forma, um arquivo *.net* com todas essas informações.

Código 8 - Exemplo de utilização da ferramenta NETCONVERT

```
netconvert --node-files node.node.xml -edge-files edge.edge.xml -t type.type.xml -o net.net.xml
```

Fonte: Autoria própria

No Código 8 foi feito um comando em linha de comando utilizando a ferramenta NETCONVERT. Nele foram colocados todos os arquivos criados nos códigos anteriores. No fim da execução será criado, nesse exemplo, o arquivo *net.net.xml* que representará a nossa rede de estradas.

Agora que já se possui a rede de estradas prontas, é necessário definir os veículos que serão utilizados e quais serão suas rotas para a interação com as vias.

Código 9 - Exemplo de criação de rotas para a simulação

```
1 <routes>
2         <vType accel="1.0" decel="5.0" id="Car"
3                length="2.0" maxSpeed="100.0" sigma="0.1" />
4         <vType accel="1.0" decel="8.0" id="Bus"
5                length="8.0" maxSpeed="60.0" sigma="0.7" />
6
7         <route id="route1" edges="1to3 3to4 4to5" />
8         <vehicle depart="10" id="car1" route="route1" type="Car" />
9
10        <route id="route2" edges="2to4 4to5" />
11        <vehicle depart="10" id="bus1" route="route2" type="Bus" />
12 </routes>
```

Fonte: Autoria própria

Deve-se criar também um arquivo XML que representará as rotas. Nele deve-se incluir informações sobre os veículos utilizados na simulação e quais serão suas rotas no ambiente de estradas da simulação. No Código 9 primeiramente são criados os tipos dos veículos conforme linhas 2 e 4. Para a criação desses deve-se definir atributos como a identidade única do veículo (*id*), aceleração (*accel*) e desaceleração (*decel*) ao percorrer as vias da rede, seu tamanho (*length*), sua velocidade máxima (*maxSpeed*) e a imperfeição do condutor nas estradas (*sigma*), que irá variar entre 0 e 1. Para essa simulação de exemplo foram criados dois criados dois veículos: um carro e um ônibus.

Em seguida, deve-se criar as rotas da simulação como mostrado nasa linhas 7 e 10. Elas são definidas pela palavra de marcação *route* e deve-se definir seu identificador único e por quais vias essa rota percorre. Para referenciar quais serão as

vias escolhidas, é necessário utilizar os mesmos *Ids* definidos no arquivo da criação das vias.

Por fim, é necessário vincular qual será a rota de cada veículo. Para isso, conforme linhas 8 e 11, se usa a palavra de marcação *vehicle* que deverá referenciar o *ID* do veículo, a referência de qual rota pertencerá a esse veículo, em que momento da simulação esse veículo deverá aparecer (*depart*) e o tipo do veículo.

Para que todos os arquivos criados até agora sejam reconhecidos pelo SUMO, deve-se criar um arquivo de configuração (*.sumo.cfg* ou *sumocfg*).

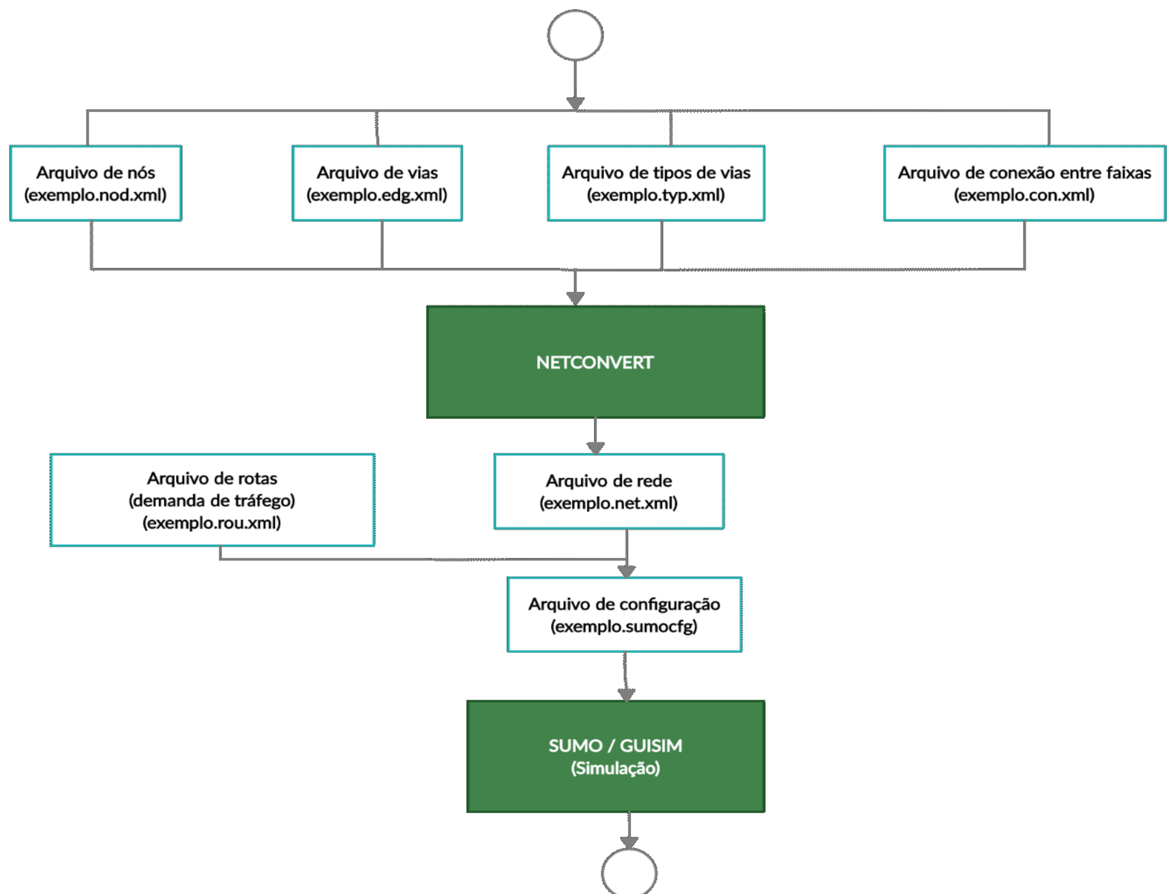
Código 10 - Exemplo de arquivo de configuração do SUMO

```
1 <configuration>
2     <input>
3         <net-file value="park.net.xml"/>
4         <route-files value="park.rou.xml"/>
5         <no-step-log value="True"/>
6         <time-to-teleport value="0"/>
7     </input>
8 </configuration>
```

Fonte: Autoria própria

No Código 10 é apresentado o arquivo de configuração utilizado para a criação da simulação. Nele deve-se referenciar onde se encontram os arquivos de rotas e o arquivo de rede, que já possui vinculados os arquivos de nós, vias, tipos e conexões. Também foram definidos atributos de tempo onde se especificam os momentos de início (*begin*) e fim (*end*) da simulação.

Figura 6 - Estrutura de arquivos de entrada necessários para o SUMO



Fonte: Adaptado de SUMO wiki

A Figura 6 mostra toda a estrutura de arquivos necessárias para a criação da nossa simulação. Agora basta importar o arquivo de configuração pela ferramenta SUMO para iniciar a simulação.

6.1.1 *TraCI*

Anteriormente, foi mostrado como se pode criar uma simples simulação na ferramenta SUMO. Agora é necessária uma forma para que se possa extrair informações sobre o que ocorre nessa simulação. Para isso tem-se o Traffic Control Interface (*TraCI*) que é um protocolo de comunicação que nos permite obter informações sobre toda a simulação e também manipular os comportamentos dos objetos que estão interagindo com nossa rede em tempo real. O *TraCI* utiliza a

arquitetura TCP/IP onde a ferramenta SUMO atuará no lado do servidor. Deve-se definir a porta para a conexão e então a ferramenta irá esperar a conexão de aplicações externas para a comunicação.

Após estabelecida a comunicação entre o SUMO e a aplicação cliente, o cliente pode solicitar o início da execução da simulação no SUMO, pode solicitar informações dos veículos que estarão compondo nosso ambiente ou também consegue obter informações do ambiente simulado.

O cliente também é responsável por encerrar a conexão com o SUMO. Quando o cliente emitir de fechamento, todos os recursos serão liberados e a simulação será encerrada.

6.2 CONSIDERAÇÕES FINAIS

Após o estudo de diversas ferramentas disponíveis para simulações de trânsito, chegou-se à conclusão de que melhor delas para este projeto seria o simulador microscópico SUMO por se tratar de uma ferramenta gratuita, de fácil aprendizagem e que se adapta bem ao sistema de estacionamento pretendido para este trabalho. Apesar disso, o SUMO ainda se demonstra uma ferramenta um pouco trabalhosa já que, por exemplo, todas as rotas dos veículos devem ter sido escritas antes do início da simulação. Dessa forma, deve-se escrever cada rota que poderia ser tomada para cada uma das vagas disponíveis no estacionamento.

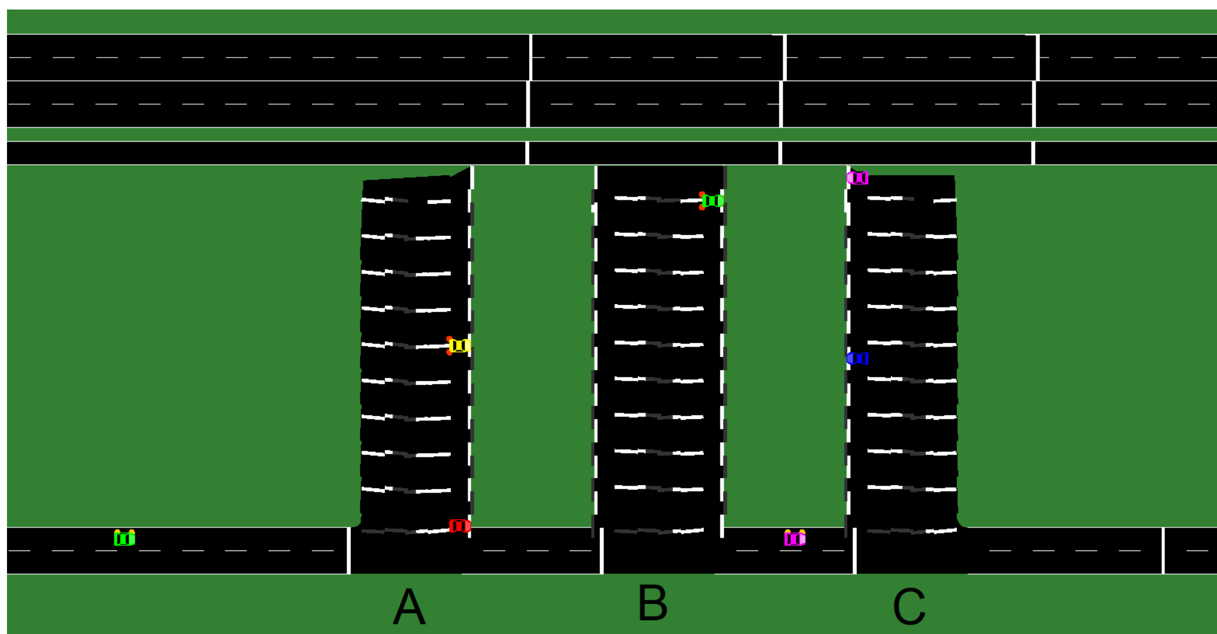
7 INTERLIGAÇÃO ENTRE O SUMO E O FRAMEWORK JADE

Neste capítulo serão apresentadas todas as etapas para o desenvolvimento prático desse projeto. Inclui desde a modelagem da rede de estradas do SUMO e a interligação do protocolo de comunicação *TraCI* e o framework JADE.

7.1 MODELO DE ESTACIONAMENTO

Para o desenvolvimento desse projeto, foi utilizado o protótipo de estacionamento *City Mobil* assim como no trabalho proposto por Heijimeijer (2016). Esse protótipo está disponível ao realizar *download* dos arquivos fonte do SUMO. Ele simula um ambiente de estacionamento composto por diversas vagas e setores. Em seu funcionamento original, este protótipo apresenta a interação de carros, ônibus, pedestres, pontos de ônibus etc. O foco desse projeto está na interação de carros num ambiente de estacionamento utilizando a ferramenta SUMO. Portanto o protótipo foi modificado para atuar de tal forma.

Figura 7 - Simulação de estacionamento utilizando CityMobil



Fonte: Autoria própria

O exemplo *CityMobil* possui, originalmente, uma estrada para carros, possuindo mão única, uma estrada para *Cyber Cars*, que possui mão dupla, uma

passagem para pedestres, possuindo pontos de ônibus e vários setores para o estacionamento de carros. Nesse projeto, faz-se necessário somente a estrada para os carros e foram utilizados apenas 3 setores de estacionamento.

Os arquivos de nós, vias e conexões foram adaptados do projeto *CityMobil* para agir em conformidade com o pretendido neste projeto. Já o arquivo de rotas precisou ser totalmente reescrito.

A simulação dispõe de 3 setores denominados como A, B e C. Os setores A e C possuem 10 vagas de estacionamento cada enquanto o setor B possui 20 vagas totalizando, dessa forma, 40 vagas de estacionamento nessa simulação.

Para cada vaga nessa simulação existe uma rota de chegada e saída para a mesma. Os nomes das vagas seguem um padrão identificando o setor, o número de posição da vaga e se a vaga se encontra do lado esquerdo ou direito do setor. Dessa forma, a rota “vagaC1E”, por exemplo, representa a rota para a segunda vaga do lado esquerdo do setor C. Para as vagas de saída são utilizados os mesmos nomes, porém com o prefixo “_out”. Então a vaga de saída da rota “vagaC1E” é representada pela rota “vagaC1E_out”.

Código 11 - Rotas de chegada e saída de uma vaga da simulação

```
1 <route id="vagaC1E" edges="mainin main0 main1
2      road1-2-0 road1-2-1 slot1-1r" >
3     <stop parking="false" until="1000000" lane="slot1-1r_0"/>
4 </route>
5 <route id="vagaC1E_out" edges="-slot1-1r
6       -road1-2-1 -road1-2-0 main2 main3” />
```

Fonte: Autoria própria

Todas as rotas de chegada possuem a *tag* de *stop*. Nela são definidas as propriedades para a alocação do carro na vaga. Dessa maneira as rotas de chegada são definidas com o percurso total definidos pelas vias até a chegada na vaga. O carro fica parado no fim do caminho da última via que representa a vaga de estacionamento. Após acabado seu tempo de alocação, é assumida a rota de saída que leva o veículo até a estrada final de saída da simulação.

Como nesse projeto foram utilizados apenas veículos para a demonstração, foram definidos apenas os *types* para os veículos. Todos eles possuem as mesmas

propriedades de tamanho, velocidade máxima etc, com exceção da cor. Foram definidos veículos das cores verde, azul, amarelo, rosa e vermelho.

Código 12 - Tipos de veículos utilizados na simulação

```
 1 <vType id="carroVerde" length="3.0" minGap=".5"
 2        guiShape="passenger" maxSpeed="50" color="0.1,1.0,0.0" />
 3 <vType id="carroAzul" length="3.0" minGap=".5"
 4        guiShape="passenger" maxSpeed="50" color="0.0,0.0,1.0" />
 5 <vType id="carroAmarelo" length="3.0" minGap=".5"
 6        guiShape="passenger" maxSpeed="50" color="1.0,1.0,0.0" />
 7 <vType id="carroRosa" length="3.0" minGap=".5"
 8        guiShape="passenger" maxSpeed="50" color="1.0,0.0,1.0" />
 9 <vType id="carroVermelho" length="3.0" minGap=".5"
10        guiShape="passenger" maxSpeed="50" color="1.0,0.0,0.0" />
```

Fonte: Autoria própria

7.2 INTERLIGAÇÃO COM O PROJETO

A interligação entre a ferramenta SUMO com o código-fonte desenvolvido é possível com a biblioteca *TraCI*. O processo realizado se dá de forma cliente/servidor. O SUMO atua do lado do servidor e o *TraCI* é o *middleware* responsável pela comunicação entre o SUMO e o cliente. *TraCI* foi originalmente escrito na linguagem Python e toda a sua documentação pode ser obtida no próprio *site* da ferramenta SUMO. Apesar de ter origem na linguagem Python, a *TraCI* foi implementada por diversos colaboradores em outras linguagens de programação.

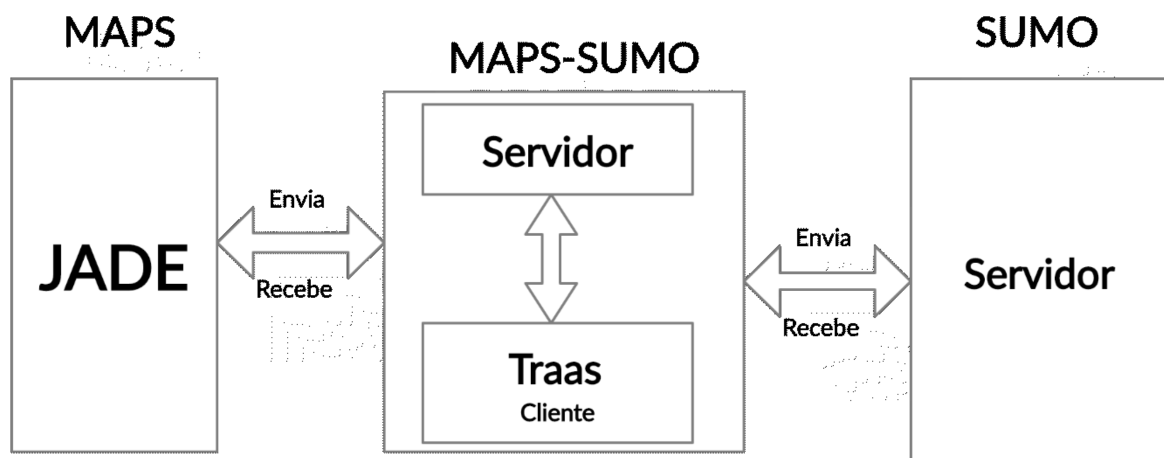
Para a linguagem Java, existe a biblioteca *TraCI4J*, que é uma representação da *TraCI* escrita em Java pelos membros da ApPeAL (*Applied Pervasive Architectures Lab*) do *Politecnico di Torino*. Assim como a *TraCI*, a *TraCI4J* trabalha via TCP. Ela atua como um *middleware* entre o SUMO e a aplicação concedendo informações sobre a simulação em tempo real. Apesar de reescrever as funções da *TraCI* para Java, a *TraCI4J* não é uma cópia exata de todas as funções presentes no *TraCI*, muitas delas não estão presentes na *TraCI4J*.

A *TraCI4J* acabou se tornando um projeto descontinuado, mas deu origem a outra biblioteca: a *TraaS (TraCI as a Service)*. A *TraaS* foi desenvolvida por Mario Krumnow e também tem como objetivo trazer uma biblioteca de desenvolvimento com o *TraCI*. Ela é uma implementação da biblioteca *TraCI4J*. Apesar de ter dado continuidade à biblioteca *TraCI4J*, a *TraaS* traz algumas diferenças entre as duas bibliotecas. *TraaS* reproduz com mais fidelidade que a *TraCI4J* as funções já

desenvolvidas para *TraCI*. Mas, diferentemente da *TraCI*, nem a *TraCI4J* nem a *TraaS* possuem uma documentação bem definida. A *TraaS* possui apenas um *javadoc* para a descrição da API. Por se tratar de uma biblioteca mais recente e mais completa escrita em Java, a fim de fazer a conexão com a ferramenta SUMO, a biblioteca *TraaS* foi a escolhida para ser utilizada nesse projeto.

Para a comunicação com a biblioteca *TraaS*, foi o utilizado o *framework* JADE que controlará toda a aplicação desenvolvida do lado do cliente. Dessa forma, o JADE é responsável por enviar e receber informações necessárias para a realização da simulação. A *TraaS* deve capturar as informações do cliente e enviá-las ao SUMO e também absorver as informações do mesmo para mandá-las ao cliente, ou seja, deverá trabalhar como um *middleware* entre o SUMO e o JADE. Por fim, a ferramenta SUMO se encarrega de realizar a simulação baseada nos arquivos XML já escritos e nas informações que serão recebidas do JADE.

Figura 8 - Interligação entre o JADE e o SUMO



Fonte: Adaptado Heijimeijer (2016)

Nessa simulação foram usados apenas dois tipos de agentes: o agente *manager* e o agente *driver.* Nela coexistiram diversos agentes *drivers* enquanto existirá apenas um agente *manager*. Toda comunicação da simulação é feita entre um agente *driver* e o manager.

Após a interligação de todas as ferramentas necessárias, para a execução desse projeto, podem ser citados três pontos principais na aplicação: a alocação e remoção do veículo na vaga de estacionamento e inserção do mesmo na fila de espera. Essas funções serão todas controladas pelo agente *manager*. Ele é o agente

determinante para a tomada de ações desse projeto enquanto o agente *driver* tem a função de solicitar uma vaga ao agente *manager* para que, após isso, eles mantenham uma "conversa" sobre o *status* do processo avisando que a vaga foi alocada ou que está saindo do estacionamento, por exemplo.

7.2.1 Visão geral

Esta simulação consiste na utilização de um único agente *manager* a *n* agentes *drivers*. O Código 13 mostra a criação de tais agentes por meio do *framework* JADE.

Código 13 – Criação dos agentes da simulação

```
 1 Runtime runtime = Runtime.instance();
 2 ProfileImpl profile = new ProfileImpl();
 3 ContainerController controller =
 4       runtime.createMainContainer(profile);
 5 AgentController rma;
 6 AgentController driver;
 7 AgentController manager;
 8
 9 try {
10    rma = controller.createNewAgent
11          ("rma", "jade.tools.rma.rma", null);
12    rma.start();
13
14    manager = controller.acceptNewAgent
15          ("Manager", new Manager());
16    manager.start();
17
18    System.out.print("Digite a quantidade de veículos:");
19    Scanner scanner = new Scanner(System.in);
20    int qtd = scanner.nextInt();
21
22    for(int i = 0; i < qtd; i++) {
23          driver = controller.acceptNewAgent
24                ("Driver" + i, new Driver());
25          driver.start();
26    }
27 } catch(Exception e) {
28    e.printStackTrace();
29 }
```

Fonte: Autoria própria

No Código 13 é possível observar que na linha 14 é criada uma única instância da classe *Manager* que representa o agente de mesmo nome enquanto a quantidade de agentes *drivers* dependerá do valor digitado pelo usuário. O agente *driver* é criado instanciando a classe de mesmo nome na linha 23.

Antes de mostrar cada uma das principais funções presentes na simulação, será descrita uma visão geral da aplicação deste trabalho.

Durante toda a simulação se fez necessário ter controle sobre algumas informações sobre a aplicação. Estas informações estão descritas no Código 14.

Código 14 - Informações importantes à simulação

```
1    private ArrayList<Veiculo> veiculos;
2    private ArrayList<Veiculo> filaEspera;
3    private SumoStringList rotas;
```

Fonte: Autoria própria

Neste trabalho, todos os veículos presentes na simulação e na fila de espera são armazenados em *ArrayLists* para obter uma melhor eficiência no gerenciamento desses elementos. Além disso, todas as rotas possíveis num cenário dessa simulação são obtidas e guardadas na variável *rotas*.

Os códigos 15, 16 e 17 descrevem as funcionalidades contidas na *GerenciarEstacionamento*, que é a classe de comportamento do agente *manager* estendendo a classe *CyclicBehaviour*.

O Código 15 descreve a inserção de veículos no estacionamento ou na fila de espera.

Código 15 – Alocação de vagas

```
1 switch(etapa) {
2    case 0:
3       ACLMessage mensagem = null;
4       if(mensagemEspera == null) {
5           mensagem = receive();
6       } else {
7           mensagem = mensagemEspera;
8           mensagemEspera = null;
9       }
10       if(mensagem != null) {
11           boolean isNumeric = mensagem.getContent().matches("\\d{2,}");
12           if(isNumeric) {
13               boolean resposta =
14                    adicionarVeiculo(Integer.parseInt(mensagem.getContent()),
15                       mensagem.getSender().getLocalName());
16               try {
17                   ACLMessage resp = mensagem.createReply();
18                   if(resposta) {
19                        resp.setContent(mensagem.getSender().getLocalName() +
20                                   " conseguiu uma vaga para estacionar");
21                       resp.setPerformative(ACLMessage.INFORM);
22                   } else {
23                        resp.setContent(mensagem.getSender().getLocalName() +
24                                   " foi adicionado a fila de espera");
25                        resp.setPerformative(ACLMessage.FAILURE);
26                   }
27                   etapa = 1;
28                   send(resp);
29               } catch(Exception e) {
30                   System.out.println(e);
31               }
32           } else {
33                   respostaEspera = mensagem;
34                   etapa = 1;
35           }
36       }
37       break;
```

Fonte: Autoria própria

Nesse passo, a aplicação espera uma solicitação de vaga de um veículo que chegou ao estacionamento. Após a chegada da mensagem, o agente *manager* tenta adicionar o veículo à uma vaga de estacionamento, caso obtenha sucesso, o veículo é adicionado e após isso é enviada uma mensagem ao agente solicitante da vaga dizendo que a vaga foi alocada. A verificação de lotação no estacionamento é verificada pela função *adicionarVeículo*, conforme linha 14, devolvendo um *boolean*. Caso o estacionamento esteja lotado, o veículo é adicionado à fila de espera e é enviada uma mensagem a ele avisando que ele foi alocado na fila de espera.

Código 16 - Troca de mensagens

```
 1 case 1:
 2   ACLMessage resposta = null;
 3   if(respostaEspera == null) {
 4       resposta = receive();
 5   } else {
 6       resposta = respostaEspera;
 7       respostaEspera = null;
 8   }
 9   if(resposta != null) {
10        boolean isNumeric = resposta.getContent().matches("\\d{2,}");
11        if(!isNumeric) {
12            System.out.println(resposta.getContent());
13        } else {
14            mensagemEspera = resposta;
15        }
16        etapa = 0;
17    } else {
18        block();
19    }
20    break;
```

Fonte: Autoria própria

O Código 16 descreve a etapa onde ocorre a maior parte da troca de mensagens entre o agente *manager* e um agente *driver*. Todas as ações tomadas por ambos os agentes numa conversação são repassadas um ao outro. Esse trecho de código é responsável por aguardar todas as mensagens de um *driver*. Todas as mensagens recebidas serão da performativa *inform*.

Código 17 - Gerenciadores de tempo e fila de espera

```
1  if(filaEspera.size() > 0) {
2     gerenciarFilaEspera();
3  }
4
5  gerenciarRemoverVeiculo();
6  avancarEtapa();
```

Fonte: Autoria própria

O Código 17, diferentemente do Código 15 e Código 16Código *16*, é executado ciclicamente sem nenhuma condição. Esse trecho realiza a chamada de três funções:

- gerenciarFilaEspera: gerencia os veículos contidos na lista de espera e, caso haja vaga disponível, aloca um veículo para a mesma;
- gerenciarRemoverVeiculo: faz verificação constante do tempo restante de todos os veículos estacionados. Quando o tempo de alguma *driver* se esgota, é enviada uma mensagem a esse agente e ele é removido da simulação;

• avancarEtapa: realiza o comando para avançar uma etapa na simulação.

7.2.2 Alocação de veículos

Para a manipulação de informações necessárias dos veículos à aplicação foi desenvolvida uma classe para armazenamento de tais dados. Para cada veículo é armazenado seu ID, o tempo que permanecerá estacionado, sua avalição, sua cor e sua rota. A rota do veículo é a única informação que não é definida desde sua criação, ela é atribuída dinamicamente durante a aplicação.

O valor da avaliação corresponde ao nível de confiança que determinado *driver* possui na simulação, assim como proposto por Heijimeijer (2016) com a utilização do *trust*. Ele é utilizado como critério de desempate quando houver mais de um agente *driver* na fila de espera. Dessa forma, o *driver*, na fila de espera, com maior avaliação será o agente que ocupará a próxima vaga liberada no estacionamento.

Para a alocação de um veículo, o agente *driver* solicita, ao agente *manager*, uma vaga de estacionamento assim que entra na simulação. O agente *driver* deve informar quanto tempo ele permanecerá no estacionamento. Avaliação do veículo é definida aleatoriamente pela aplicação podendo variar com valores de 0 a 500.

A cor do veículo é definida com base em sua avaliação. Esta forma de atribuição é semelhante à definida por Heijimeijer (2016) como mostrado no Quadro 1.

Quadro 1 - Relação entre a cor e o valor de avaliação

Cor	Valor da avaliação
Vermelho	0 a 99
Rosa	100 a 199
Amarelo	200 a 299
Azul	300 a 399
Verde	Acima de 399

Fonte: Autoria própria

A comunicação entre os agentes *driver* e *manager* é feita por meio de mensagens ACL. O agente *driver* deve, então, enviar o tempo que ficará no estacionamento por meio de mensagem para o agente *manager*.

O Código 18 descreve como é feita a solicitação de um vaga de estacionamento por um agente *driver*.

Código 18 - Solicitação de vaga pelo driver

```
1  Random rand = new Random();
2  int index;
3
4  ACLMessage mensagem = new ACLMessage(ACLMessage.REQUEST);
5  mensagem.addReceiver(new AID("Manager", AID.ISLOCALNAME));
6  mensagem.setLanguage("portgues");
7  index = rand.nextInt(6);
8  mensagem.setContent(String.valueOf(tempos[index]));
9  send(mensagem);
```

Fonte: Autoria própria

O valor do tempo está sendo determinado randomicamente dentro de um conjunto de valores predefinidos. Esse valor então é inserido no conteúdo da mensagem, conforme linha 8, e enviado para o agente *manager*.

O agente *manager* se encarrega de fazer a análise do veículo em relação à situação da simulação. Esse agente mantém controle de todas as rotas disponíveis e também de todos os veículos que estão presentes na simulação. Para que seja possível a alocação de um novo veículo, o agente *manager* verifica se existe alguma rota que ainda não está sendo utilizada por nenhum outro veículo na simulação. Caso ainda haja rotas disponíveis, o agente se prepara para fazer a inserção de um novo veículo na simulação.

Para inserir um veículo ou fazer qualquer outra alteração na simulação em tempo real, é necessária utilizar a função *do_job_set* da biblioteca *TraaS*. Como pode ser visto no Código 19, para adicionar um veículo na simulação deve-se usar a função *add* da classe *Vehicle*. Deve-se passar como parâmetro, respectivamente, o ID do veículo, seu tipo, sua rota, *departure*, representando o tempo em que o veículo iniciará na simulação, posição, velocidade e faixa da estrada. O tipo do veículo somente irá se diferenciar pela cor do mesmo, ao qual é definida pela avaliação. A rota selecionada para a inserção será a rota livre escolhida pelo agente *manager*.

Código 19 - Inserção de veículo pela biblioteca *TraaS*

```
conn.do_job_set(Vehicle.add("Veiculo" + contador, type, rotas.get(index), 0, 0.0, 3.0, (byte)0));
```

Fonte: Autoria própria

Após isso deve-se também adicionar o veículo na lista interna para que o agente *manager* também consiga manter controle sobre as avaliações e tempo dos veículos presentes na simulação.

Depois de executada a função da biblioteca *TraaS* para inserir o veículo, o mesmo é imediatamente inserido na simulação e seguirá o fluxo normal de sua rota definida.

7.2.3 Remoção de veículos

A função de remoção de veículos é chamada sempre que o tempo de estacionamento de determinado agente driver é encerrado.

O controle de tempo é feito por uma unidade de medida de tempo do próprio SUMO, neste trabalho definida como Unidade de Tempo do SUMO (uts).

A biblioteca TraaS não permite a alteração da propriedade until da tag stop. Dessa forma, não é possível alterar dinamicamente pela aplicação o tempo que um veículo permanecerá estacionado.

Para contornar esse problema, foi definido um mesmo valor máximo padrão de tempo de estacionamento para todos os veículos. A aplicação armazena o valor do tempo inserido pelo usuário para a criação de um agente driver na simulação e a própria se encarrega de fazer a contagem do tempo de restante de todos os veículos já inseridos. Quando o tempo de determinado veículo se encerra, é enviado o ID do veículo para uma função da biblioteca TraaS que remove o veículo da simulação imediatamente. Após, é inserido outro veículo com as mesmas informações do veículo removido com exceção da sua rota. A rota utilizada é trocada pela rota de mesmo nome com adição do sufixo "_out". Dessa forma, o "novo" veículo partirá do mesmo local que o veículo anterior e seguirá sua rota de saída do estacionamento.

Apesar da ocorrência de remoção de um veículo para a inserção de outro, visualmente não é possível perceber essa troca na simulação o que continua dando a sensação de que é o mesmo veículo que fez as rotas de chegada e saída.

Os veículos definidos com rotas de saída possuem também sufixo "_out" em seu ID. Os veículos marcados com esse sufixo não são contabilizados pelo

armazenamento interno de veículos da aplicação, pois não irão mais interferir mais na aplicação, apenas seguirão a rota de saída até que desapareçam da simulação.

Após a remoção do veículo é enviada, para o agente driver removido, uma mensagem informando que seu tempo para utilização da vaga foi esgotado como indicado entre as linhas 10 e 13 do Código 20.

O Código 20 descreve como é feita a remoção de veículos para essa simulação.

Código 20 - Remoção de veículos

```
1 String id = veiculo.getID();
2 String tipo =
3        String.valueOf(conn.do_job_get(Vehicle.getTypeID(id)));
4 String rota =
5        String.valueOf(conn.do_job_get(Vehicle.getRouteID(id)));
6 conn.do_job_set(Vehicle.remove(veiculo.getID(), (byte)0));
7 conn.do_job_set(Vehicle.add(id + "_out", tipo,
8      rota + "_out", 0, 0, 5, (byte)0));
9
10 ACLMessage mensagem = new ACLMessage(ACLMessage.INFORM);
11 mensagem.setContent("O tempo de " + id + " se esgotou");
12 mensagem.addReceiver(new AID(id, AID.ISLOCALNAME));
13 send(mensagem);
```

Fonte: Autoria própria

Nas linhas de 1 a 5 são obtidas as informações necessárias referentes ao veículo para que se possa criar sua “réplica”. A linha 6 realiza a remoção do veículo da simulação e logo em seguida, na linha 7, é adicionar um veículo na mesma posição, porém com uma rota de saída e *id* de sufixo “_out”. Logo a seguir é enviada uma mensagem comunicando que o tempo do veículo no estacionamento está esgotado.

7.2.4 Inserção de veículos na fila de espera

Quando um agente *driver* solicita uma vaga ao agente *manager* e este verifica que ainda existem vagas disponíveis no estacionamento, a rota referente a essa vaga é destinada ao agente *driver*. Caso todas as rotas paras as vagas já estejam em uso por veículos presentes na simulação, o agente *driver* que a solicitou deve ser encaminhado para a fila de veículos.

O agente *manager* também mantém controle de todos os veículos que se encontram na fila por meio de uma lista de veículos. Quando um agente *driver* é

encaminhado para a fila, todas as informações do veículo são salvas na lista para a espera de uma vaga disponível. A fila de espera de espera não é representada de forma visual neste trabalho, apenas existe o controle desta no código.

A decisão de qual veículo da fila conseguirá a próxima vaga dependerá de sua avaliação. O agente da fila que possuir a maior avaliação será o que receberá rota da próxima vaga livre. Para que não aconteça de algum agente nunca ser chamado por possuir uma avaliação muito baixa, a aplicação aumenta a avalição de todos os agentes na fila a cada etapa da simulação. Dessa maneira, um agente que esteja esperando na fila há muito tempo, receberá cada vez maior avalição para que, assim, possa conseguir uma vaga no estacionamento.

No Código 21Código *21*, a ação para adicionar um veículo na fila de espera está na própria função para alocar um veículo. Quando um veículo consegue uma vaga no estacionamento com sucesso, ele é alocado a essa vaga pelo tempo predefinido, mas caso isso não seja possível pela lotação do estacionamento, é chamada a função *adicionarFilaEspera* para alocar esse veículo na fila de espera.

Código 21 - Inserção do veículo na fila de espera

```
 1 if(estacionado != rotas.size() && !temFila) {
 2    conn.do_job_set(Vehicle.add(localName, type,
 4            rotas.get(index), 0, 0.0, 3.0, (byte)0));
 5    veiculos.add(new Veiculo(localName,
 6            rotas.get(index), avaliacao, time));
 7    comecar = true;
 8    return true;
 9 } else {
10    adicionarFilaEspera(localName, avaliacao, time);
11    return false;
12 }
```

Fonte: Autoria própria

No Código 21 a estrutura verifica se ainda existem rotas que não estão sendo utilizadas pelos veículos da simulação. Para isso, a linha 1 faz uma validação de se o número de veículos estacionados é diferente do número total de rotas da simulação caso falso, significa que o estacionamento está lotado. Nesse caso, é chamada a função adicionarFilaEspera, conforme linha 10, passando como parâmetro seu nome, avaliação e tempo que deseja ficar estacionado.

Código 22 - Função de inserção na fila de espera

```
1 public void adicionarFilaEspera(String localName, int avaliacao, int time) {
2     Veiculo veiculo = new Veiculo(localName, avaliacao, time);
3     filaEspera.add(veiculo);
4 }
```

Fonte: Autoria própria

A função de inserção da fila de espera apenas adiciona o veículo que solicitou a vaga na variável de controle dos veículos da fila de espera conforme linha 3.

Código 23 - Gerenciador da fila de espera

```
 1 if(estacionado != rotas.size()) {
 2     try {
 3        String type = definirCor(veiculoFila.getAvaliacao());
 4        conn.do_job_set(Vehicle.add(veiculoFila.getID(),
 5                type, rotas.get(index), 0, 0.0, 3.0, (byte)0));
 6        veiculos.add(new Veiculo(veiculoFila.getID(),
 7                rotas.get(index), veiculoFila.getAvaliacao(),
 8                veiculoFila.getTime()));
 9        filaEspera.remove(indiceVeiculo);
10
11        ACLMessage mensagem = new ACLMessage(ACLMessage.INFORM);
12        mensagem.addReceiver(new AID(veiculoFila.getID(),
13                AID.ISLOCALNAME));
14        mensagem.setLanguage("portgues");
15        mensagem.setContent("O " + veiculoFila.getID() +
16                " foi removido da fila de espera"
17                        + " e alocado numa vaga de estacionamento");
18        send(mensagem);
19     } catch(Exception e) {
20        System.out.println("GerenciarFailEspera:  " + e);
21     }
22 }
```

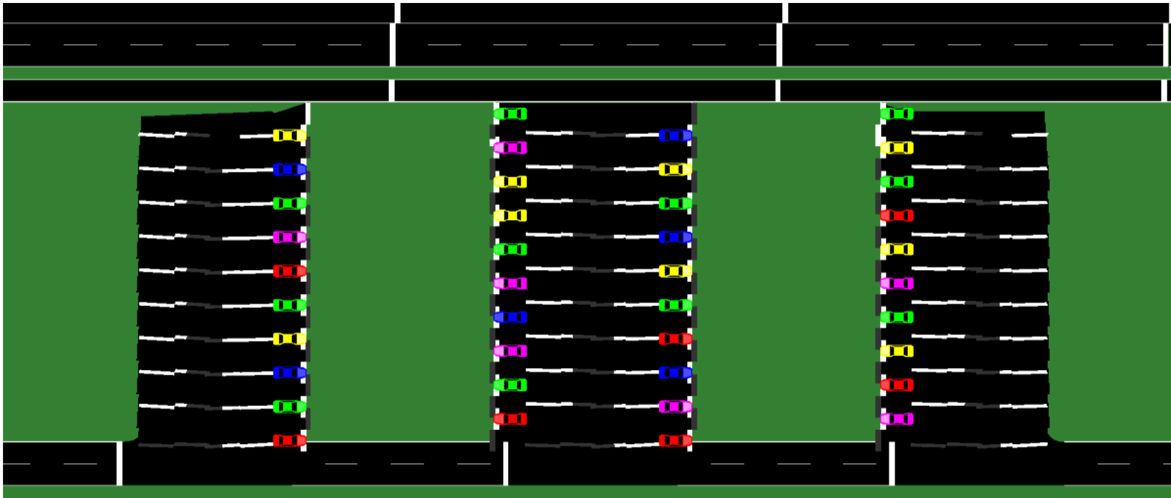
Fonte: Autoria própria

Para adicionar um veículo da fila de espera para uma vaga de estacionamento também é necessário verificar se existe alguma rota que não esteja alocada a outro veículo. Caso verdadeiro, o veículo é adicionado à simulação e removido dos veículos da fila de espera. O veículo a ser adicionado está descrito pela variável *veiculoFila* no Código 23 . Essa variável contém o veículo com a maior avaliação presente na fila de espera. Na linha 3 é chamada a função *definirCor* onde é o veículo obtém sua cor baseado em sua avaliação atual. Conforme a linha 4, o veículo é adicionado à simulação com suas propriedades e na linha 5, o veículo é adicionado à lista interna para o controle de veículos e, logo em seguida, o veículo é removido da fila de espera. A partir da linha 11 é mostrada a criação da mensagem que será enviada ao agente comunicando-o que ele foi removido da fila de espera e alocado à uma vaga de estacionamento.

Por meio dessas três funções principais foi possível o desenvolvimento da simulação de um estacionamento gerenciado e utilizado por agentes. A ferramenta SUMO consegue obter toda informação enviada pela biblioteca *TraaS* e transformá-la na simulação.

Um exemplo de estacionamento lotado utilizando a aplicação desenvolvida pode ser visto na Figura 9.

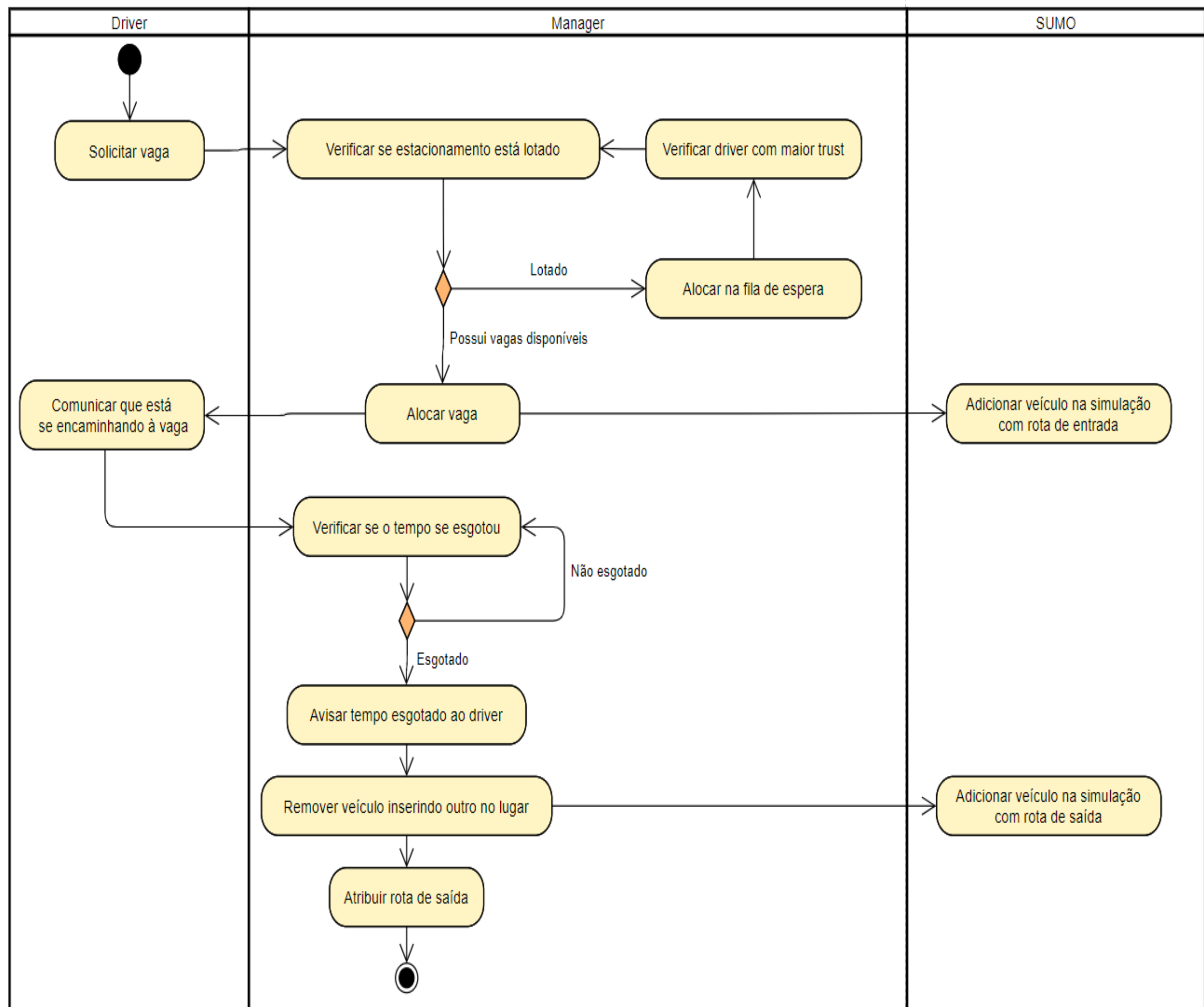
Figura 9 - Simulação de estacionamento utilizando todas as vagas



Fonte: Autoria própria

A Figura 10 apresenta um diagrama de atividades para os principais eventos ocorridos na durante a simulação.

Figura 10 - Diagrama de atividades do projeto



Fonte: Autoria própria

Todos os agentes *driver* entram na simulação solicitando uma vaga ao *manager*. Caso o estacionamento esteja lotado, o agente será encaminhado para a fila de espera, enquanto aguarda até que uma vaga seja liberada e o *manager* possa liberar uma vaga para ele. Após conseguir, o *driver* estaciona e espera em *loop* até que seu tempo no estacionamento seja esgotado. Depois que isso ocorrer, o *manager* avisa o tempo esgotado e o *driver* inicia a rota para sair do estacionamento. Toda a comunicação entre o agente *manager* (JADE) é feita por intermédio da biblioteca TraaS que é responsável por enviar solicitações para adicionar ou remover um veículo ao SUMO com determinada rota.

7.3 CONSIDERAÇÕES FINAIS

Neste capítulo foi demonstrada a interligação do *framework* JADE e a ferramenta SUMO para este projeto. Essa interligação é possível pela biblioteca TraaS utilizada. Foram apresentadas as principais funções do projeto para o gerenciamento do estacionamento além da comunicação entre os agentes.

8 EXPERIMENTOS

Neste capítulo serão descritos os experimentos práticos realizados neste projeto. Serão detalhadas as variações feitas utilizando o ambiente de simulação desenvolvido pela ferramenta SUMO integrado com o projeto MAPS.

8.1 CONFIGURAÇÕES DE AMBIENTE E FERRAMENTAS

Serão apresentados alguns dados estatísticos obtidos pela realização do projeto, para isso torna-se necessária a descrição de todo equipamento físico e *softwares* utilizados durante a realização dos testes desses experimentos.

8.1.1 Configurações do Sistema

Para a realização deste trabalho foram utilizadas as seguintes configurações de *hardware*:

- Fabricante: Acer
- Modelo: Aspire A515-51G
- Processador: Intel(R) Core (TM)i5-7200U CPU @ 2.50GHz 2.71GHz
- Memória instalada (RAM): 8GB DDR4 2133
- Sistema operacional: Windows 10
- Tipo de sistema: Sistema operacional de 64 bits

Para o desenvolvimento da simulação, foi necessária a instalação das ferramentas descritas a seguir:

- SUMO 1.3.1 (https://sumo.dlr.de/daily/userdoc/Downloads.html)
- Eclipse IDE 2019-09 (https://www.eclipse.org/downloads/)
- *Framework* JADE 4.5.0 (https://jade.tilab.com/download/jade/)

Além das ferramentas já citadas, foi também utilizada a biblioteca *TraaS* para o desenvolvimento em JAVA. Essa biblioteca já vem na pasta *bin* após fazer download do SUMO 1.3.1. O modelo para a simulação do estacionamento é um exemplo encontrado também na instalação do SUMO, porém não é encontrado na versão 1.3.1, mas, sim, na 0.27.1 da ferramenta.

Para o ambiente de simulação da ferramenta SUMO foi necessário o desenvolvimento de arquivos de nós, vias e conexões. Para a geração do arquivo *.net* foi utilizada a ferramenta NETCONVERT. O arquivo de configuração do SUMO (*sumocfg*) é o arquivo final reconhecido pela ferramenta para a realização da simulação gerado pela união dos arquivos de rede e o arquivo de rotas. A criação de todos os arquivos já foi descrita anteriormente e todos os experimentos seguiram o mesmo modelo de simulação descrita no projeto *CityMobil* presente nos arquivos de exemplo quando é feito o *download* da ferramenta SUMO.

8.1.2 Informações Sobre os Agentes

Os agentes do tipo *driver* reunirão em si diversas características como tempo que permanecerá no estacionamento, avaliação, cor e sua rota. Ao longo da aplicação, poderão ser adicionados *n* agentes *driver* na simulação a fim de obter os mais diferenciados tipos de resultados.

Já o agente *manager* é responsável pelo gerenciamento de toda a simulação. Qualquer decisão solicitada tomada ou solicitada durante a simulação será decidida pelo agente *manager*. Ele é o responsável por permitir a entrada de um novo veículo ao estacionamento. Cabe a ele atribuir uma rota, fazer verificações até que o tempo de determinado veículo no estacionamento seja esgotado. Desta maneira, atribuindo sua rota de saída para que o veículo deixe sua vaga no estacionamento e vá em direção a estrada final da simulação. Quando o estacionamento estiver lotado, ou seja, quando todas as rotas para a chegada nas vagas já estiverem sendo utilizadas, caberá ao agente *manager* captar as informações do novo veículo e adicioná-lo a fila de espera para uma vaga. Ele determinará baseado na avalição a qual veículo será atribuída uma rota para vaga de estacionamento da próxima vez que alguma estiver livre. Cada veículo aumentará automaticamente sua avaliação a cada etapa que se passar na simulação, dessa forma, evitando que um veículo nunca consiga vaga por possuir uma avaliação muito baixa.

Baseado nesses conceitos, podem ser definidos alguns conjuntos de dados para que se possa obter resultados usando o nosso mesmo ambiente de simulação. Para isso, nesta aplicação, serão definidos um número *n* de agentes que estarão entrarão na simulação no decorrer de seu funcionamento, um valor base de tempo

para a entrada entre um veículo e outro, um conjunto de valores com os tempos possíveis para um mesmo veículo permanecer estacionado e a avaliação definida aleatoriamente ao decorrer da simulação. Levando em conta que o estacionamento desta simulação possui um total de 40 vagas de estacionamento, pode-se trabalhar com os seguintes conjuntos

- *nAgents*: representará o número de agentes que solicitaram uma vaga de estacionamento ao decorrer da simulação. Esse valor pode ser ilimitado, mas, a seguir, serão definidos conjuntos de agentes que ocuparão espaço para a criação de diferentes cenários;
- Tempo estacionado (TE): esse valor representa o tempo que determinado veículo permanecerá estacionado numa vaga da simulação. Seu valor será escolhido aleatoriamente entre os valores do conjunto {60, 90, 120, 180, 270, 360}. Tais valores em Unidade de Tempo do SUMO (uts) e definidos para obter uma gama considerável de diferentes resultados;
- Avaliação: esse valor representa a avaliação que um veículo possui na simulação. Em caso de fila de espera, esse será o valor determinante para que o agente *manager* decida qual veículo deverá ocupar uma vaga de estacionamento da próxima vez que houver uma rota disponível. Seu valor será definido aleatoriamente entre 0 e 500. Apesar disso, quando um veículo entrar na fila de espera por uma vaga, sua avaliação aumentará automaticamente a cada etapa que se passar durante a simulação, ou seja, quando maior o tempo esperando uma vaga, maior será a avaliação de determinado veículo.

Esses são os conjuntos de dados que possibilitaram a diversificação dos cenários a serem criados. Tais valores foram escolhidos baseados no trabalho de Heijimeijer (2016), porém não idênticos.

8.1.3 Criação dos cenários

Com base nos conjuntos de dados anteriormente descritos, foi possível a criação de diferentes cenários para a simulação para que fosse possível a extração

de diversas informações sobre a mesma. Os cenários foram construídos levando em consideração os cenários desenvolvidos nos experimentos de Heijimeijer (2016), apesar de não serem idênticos.

Com base nas características descritas da estrutura do ambiente na criação da simulação na ferramenta SUMO, foram descritos cenários com a utilização de 4 conjuntos de vagas do estacionamento: 10, 20, 30 e 40 vagas.

- O cenário (1) será o que fará utilização de 10 vagas de estacionamento pertencentes ao setor A da simulação;
- O cenário (2) fará utilização de 20 vagas do estacionamento utilizando apenas as vagas do setor B;
- O cenário (3) fará utilização de 30 vagas de estacionamento representando os setores A e B;
- O cenário (4) fará utilização de 40 vagas de estacionamento utilizando todas as vagas pertencentes ao mesmo, que estão localizadas nos setores A, B e C.

Agora que já foram definidos o conjunto de vagas que estará presente em cada cenário, é necessária a definição do conjunto de agentes *driver* presentes na realização de cada cenário.

- Para o cenário (1), serão utilizados 50 veículos (v1) ao decorrer da simulação;
- Para o cenário (2), serão utilizados 80 veículos (v2) para a simulação;
- Para o cenário (3), serão utilizados 250 veículos (v3) para a simulação;
- Por fim, no cenário (4), serão utilizados 60 veículos (v4) para a realização da simulação.

O restante das informações sobre a simulação já foi definido anteriormente. Cada veículo possui uma avaliação atribuída aleatoriamente entre 0 e 500. Quando tais veículos estiverem esperando numa fila de espera, suas avaliaç serão incrementadas periodicamente. Por fim, a decisão do tempo que cada veículo permanecerá na simulação também foi definido num conjunto de dados também anteriormente descritos. Os tempos serão agregados também aleatoriamente para cada veículo.

A Tabela 1 descreve a relação entre em que setores da simulação cada conjunto de agentes estará presente ao decorrer da mesma.

Tabela 1 - Relação entre o conjunto de veículos e os setores da simulação

		Setores		
		A	B	C
Conjunto de Veículos	V1	**X**		
	V2		**X**	
	V3	**X**	**X**	
	V4	**X**	**X**	**X**

Fonte: Autoria própria

Para a geração de todos os gráficos apresentados na seção de resultados a seguir, foi levado em consideração o tempo de cada etapa de aplicação. Essa é uma medida própria da ferramenta SUMO.

Na presente simulação, essa função da biblioteca *TraaS* é definida dentro de uma classe de comportamento possível pelo *framework* JADE. Esse método será executado ciclicamente sem pausa ao decorrer de toda a simulação. Para cada movimento de todos os veículos presentes na simulação (ou mesmo enquanto eles estiverem estacionados) a função para avanço de etapa será acionada.

8.2 RESULTADOS OBTIDOS

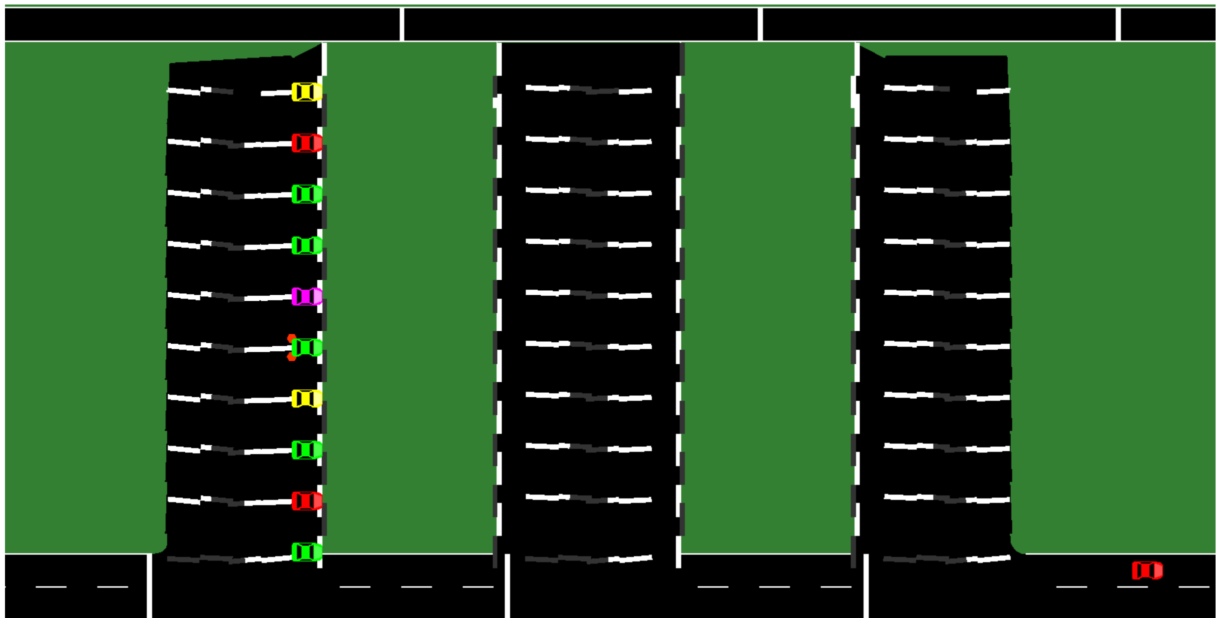
Para a análise e construção dos gráficos para a obtenção de resultados sobre a simulação em seus diferentes cenários, algumas informações deverão ser apuradas ao decorrer da simulação. Os seguinte dados foram apurados:

I. Porcentagem do uso do estacionamento: será analisada a lotação do estacionamento com base no tempo decorrido durante a simulação;
II. Tamanho da fila de espera: assim que o estacionamento se encontrar lotado, todos os próximos veículos serão alocados numa fila de espera até que haja liberação de uma vaga. Será feita análise do quão aumentará a fila de espera dependendo da quantidade de vagas e veículos que solicitarem as vagas;

8.2.1 Cenário 1

O cenário 1 é o que apresenta o menor número de vagas para a simulação, são apenas 10 vagas de estacionamento (ver Figura 11). Para a sua análise foi utilizado o conjunto *v1* de veículos que compreende 50 veículos participantes da simulação. Para o Gráfico 1 será mostrada a porcentagem do estacionamento que está sendo utilizada a cada contagem de tempo determinada pela ferramenta SUMO (*uts*). A contagem do tempo se dá a partir do momento em que o veículo entra no estacionamento, o que significa que ele já tem uma vaga garantida para ele.

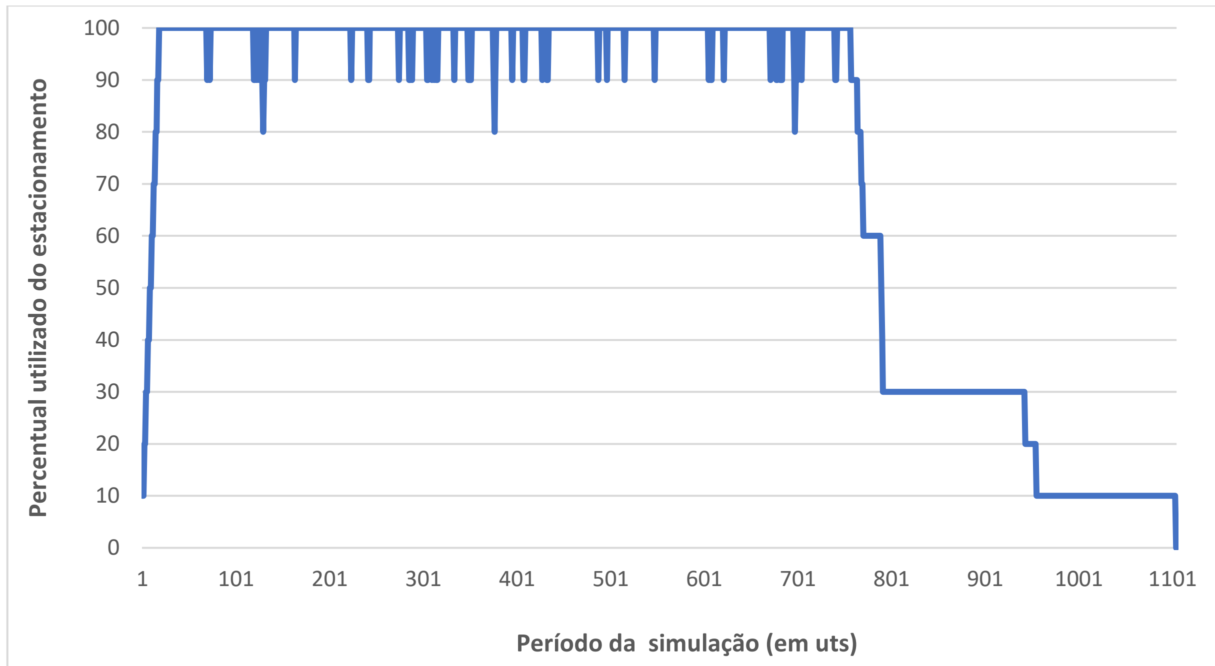
Figura 11 - Cenário 1 da simulação



Fonte: Autoria própria

A Figura 11 mostra o cenário utilizado para essa simulação. O número de veículos neste cenário (50) é consideravelmente maior do que o número de vagas disponíveis. Acredita-se que o estacionamento chegue em sua lotação máxima num espaço de tempo pequeno. Levando em conta que todos os veículos que ocuparam a fila de espera permaneceram lá até que alguma vaga seja liberada, a simulação deverá perdurar por algum tempo até que todos os veículos consigam estacionar pelo tempo desejado.

Gráfico 1 - Percentual utilizado do estacionamento do cenário 1

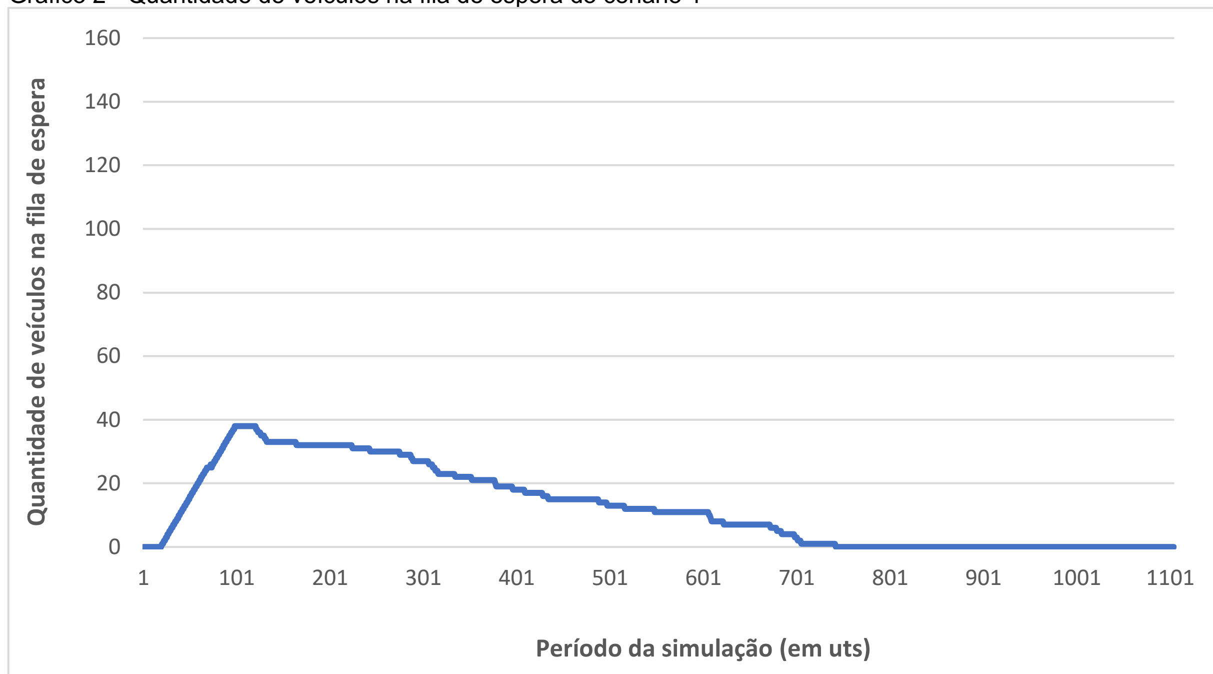


Fonte: Autoria própria

O Gráfico 1 mostra o percentual do estacionamento utilizado em cada período da simulação. É possível que na maior parte do tempo o estacionamento permaneceu lotado devido ao grande número de veículos esperando uma vaga. O gráfico até o tempo de 701uts permaneceu praticamente estável. Houve algumas quedas da porcentagem de vagas em momentos que muitos veículos saiam ao mesmo tempo devido ao tempo estacionado ter sido escolhido aleatoriamente no conjunto proposto.

Até o tempo de 801uts, a grande maioria dos veículos já conseguiram sua vaga de estacionamento e terminaram seu tempo de estadia. Desta forma, todos os próximos veículos conseguiram sua vaga rapidamente. O restante dos veículos permanece no estacionamento até seu tempo final. No gráfico, esse momento é descrito a partir do tempo de 801uts em que a porcentagem do uso de estacionamento decaia até que ela seja finalmente zerada, completando sua missão de conseguir uma vaga de estacionamento para todo o nosso conjunto de veículos.

Gráfico 2 - Quantidade de veículos na fila de espera do cenário 1



Fonte: Autoria própria

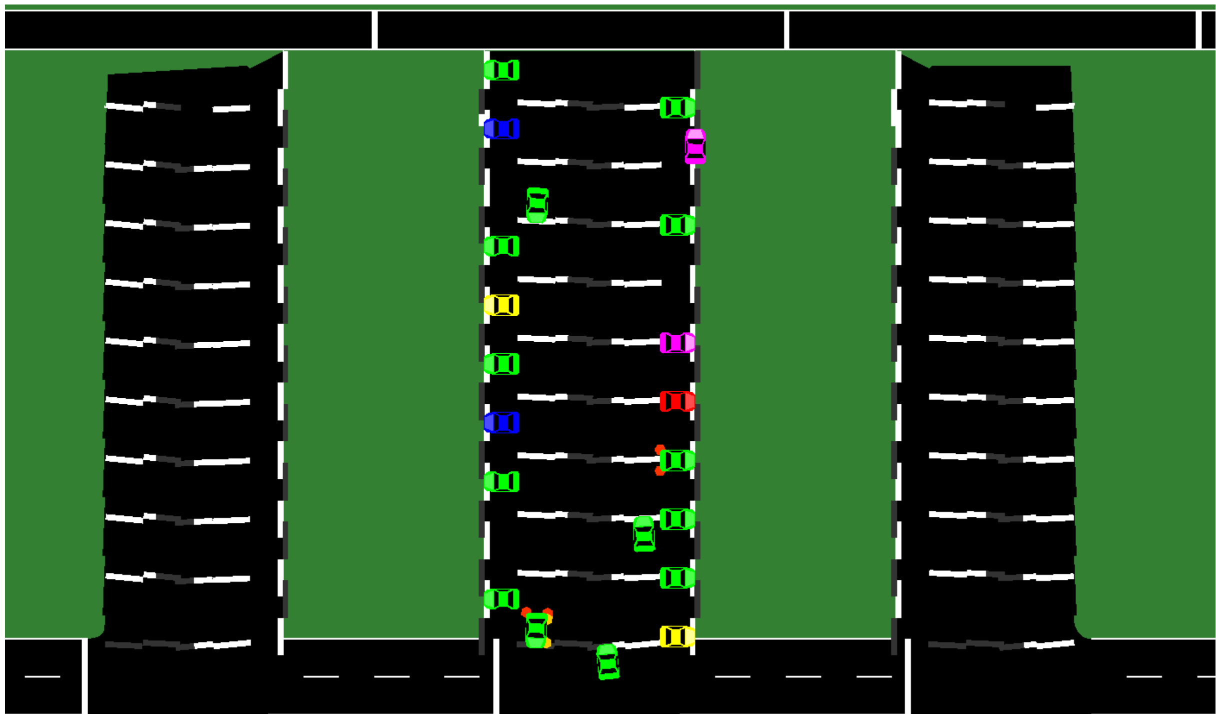
O Gráfico 2 apresenta a relação entre a quantidade de veículos que estavam na fila de espera em relação ao tempo decorrido na simulação. Nota-se que o ponto máximo do gráfico já ocorre logo depois do início da aplicação já que no início o estacionamento se encontra vazio e o intervalo em um veículo e outro é curto, todos os veículos entraram rapidamente no estacionamento fazendo com o que o mesmo atinja sua lotação máximo em pouco tempo. No decorrer do tempo, vagas do estacionamento vão sendo liberadas para fazendo com que a fila de espera vá diminuindo até que se esgote e todos os veículos da simulação tenham conseguido uma vaga de estacionamento para utilização.

Para esse primeiro cenário de teste foi possível perceber que o número de veículos em relação ao número de vagas é acima do esperado para que a alocação de vagas ocorresse com mais fluidez. O rápido intervalo entre a entrada de veículos faz com que em poucos segundos, todos os veículos do grupo solicitem uma vaga de estacionamento ao agente *manager*. Dessa forma, o estacionamento atinge a sua lotação máxima muito rápido fazendo com que a fila de espera de veículos também cresça gradativamente.

8.2.2 Cenário 2

Para o cenário 2 foram utilizadas as vagas de estacionamento presentes no setor B da simulação, que comporta 20 vagas. Nesse cenário foi usado o conjunto *v2* de veículos, que possui 80 veículos. Foram utilizados os mesmos métodos do cenário 1 para a geração dos gráficos. Foram analisadas a porcentagem de utilização do estacionamento e o número de veículos contidos na fila de espera por uma vaga. As duas informações serão levadas em consideração comparadas ao tempo decorrido pela ferramenta SUMO.

Figura 12 - Simulação do cenário 2



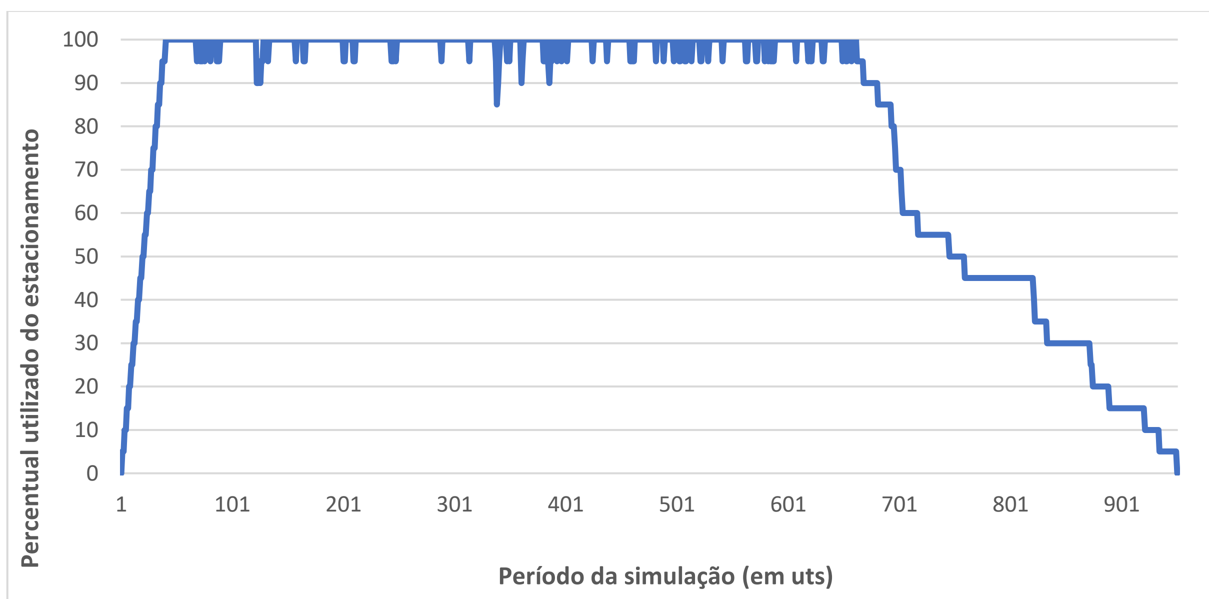
Fonte: autoria própria

A simulação do cenário 2 possui mais vagas comparado ao cenário 1, porém os 2 cenários contêm a totalidade de suas vagas de estacionamento num único setor. A diferença entre os setores é que no setor A, os veículos podem estacionar apenas de um lado do setor, nesse caso, no lado direito. Dessa forma, todos os veículos podem fazer sua rota de chegada pela pista direta do setor para que consiga estacionar e, na rota de saída, fazer seu caminho utilizando a pista esquerda para que se evite congestionamentos. As rotas de entrada e saída do setor A foram definidas dessa maneira. Porém, nessa simulação, o setor B possui o dobro de vagas do setor

A e os *drivers* podem estacionar o veículo tanto do lado esquerdo quanto do lado direito do setor. Mas, diferentemente do setor A, o setor B não possui uma pista que seja exclusiva para a entrada ou saída de veículos. Existem as duas vias de estacionamento para os lados direito e esquerdo do setor, porém é utilizada a mesma pista para a entrada e saída de veículos. Ela se tornou uma via de mão dupla. Devido ao intervalo entre os veículos ser muito pequeno e o conjunto de veículos a procura de uma vaga para estacionar ser grande, ocorre um congestionamento entre os veículos, como é possível ver na Figura 12. A própria ferramenta SUMO é capaz, por si só, resolver eventuais problemas de congestionamento ocasionados durante a aplicação.

No Gráfico 3 é mostrada a relação da porcentagem utilizada do estacionamento em relação ao tempo decorrido na simulação. Percebe-se que esse gráfico se assemelha muito ao gráfico de mesma informação presente no cenário 1. Devido a entrada constante e em pouco de veículos no estacionamento, o mesmo atingirá a sua lotação máxima em pouco tempo decorrido. Ainda em comparação ao cenário 1, o cenário 2 se difere por ter conseguido encerrar sua atividade em menos tempo do que a simulação do cenário 1. Mesmo possuindo um número maior de veículos requisitando vagas de estacionamento, o setor B também possui um número maior de vagas do que o setor B. Proporcionalmente falando, o cenário 2 realmente deveria conseguir encerrar todas as suas atividades em menos tempo decorrido que o cenário 1.

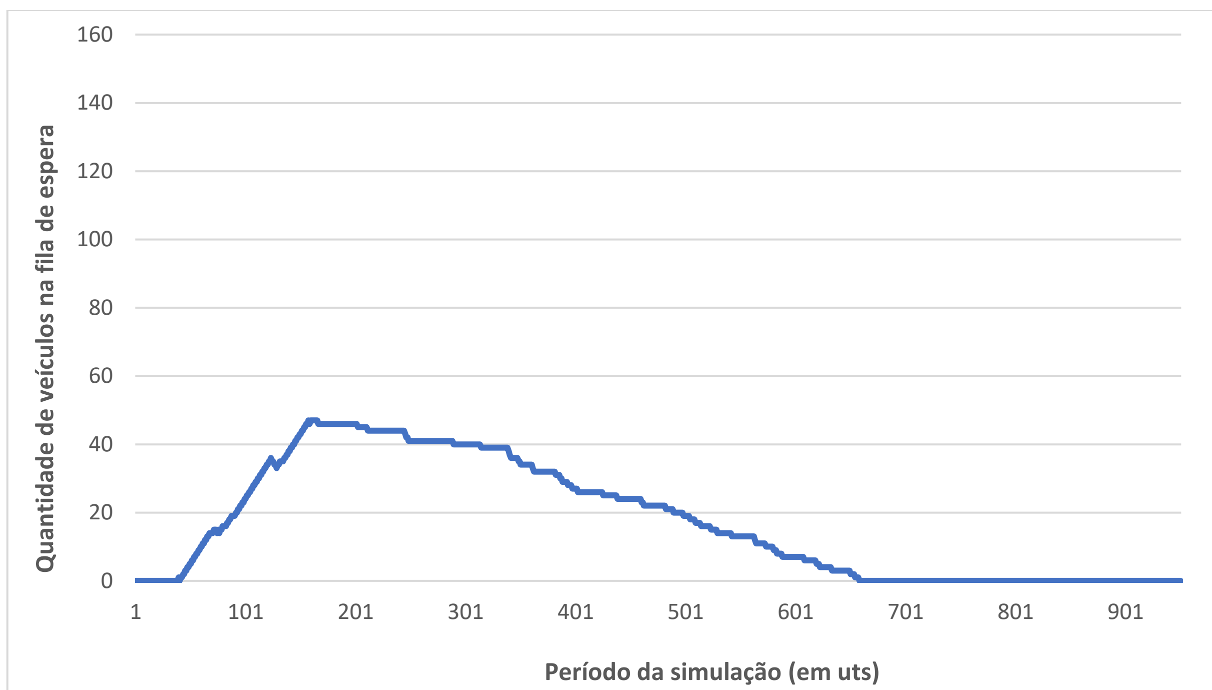
Gráfico 3 - Percentual utilizada do estacionamento no cenário 2



Fonte: Autoria própria

Nota-se que o gráfico permanece quase que constante durante a maior parte do tempo apontando, na maioria das vezes, para a lotação do estacionamento. Ao final, os veículos vão encerrando suas atividades e a porcentagem de utilização do estacionamento seguirá diminuindo até que acabe zerada e a simulação chegue ao seu fim.

Gráfico 4 - Quantidade de veículos na fila de espera do cenário 2



Fonte: Autoria própria

O Gráfico 4 descreve a quantidade de veículos contidos na fila de espera por uma vaga ao decorrer da simulação. Esse gráfico também se assemelha ao gráfico de mesma informação do cenário 1. A fila de espera atinge seu ponto máximo antes da metade da simulação devido ao grande fluxo de veículos. Com o tempo, os veículos irão deixando as vagas de estacionamento e quantidade de veículos na fila de espera diminuirá cada vez mais até que atinja o valor 0.

A simulação do cenário 2 ocorreu muito similar ao cenário 1. A principal diferença entre os dois cenários não esteve nos dados sobre a porcentagem de vagas utilizadas ou quantidade de veículos na fila de espera, mas sim no fluxo do trânsito ao decorrer da aplicação. Na execução da simulação do cenário 1 não houve nenhum problema relacionado a congestionamento apesar do alto fluxo de veículos constantemente.

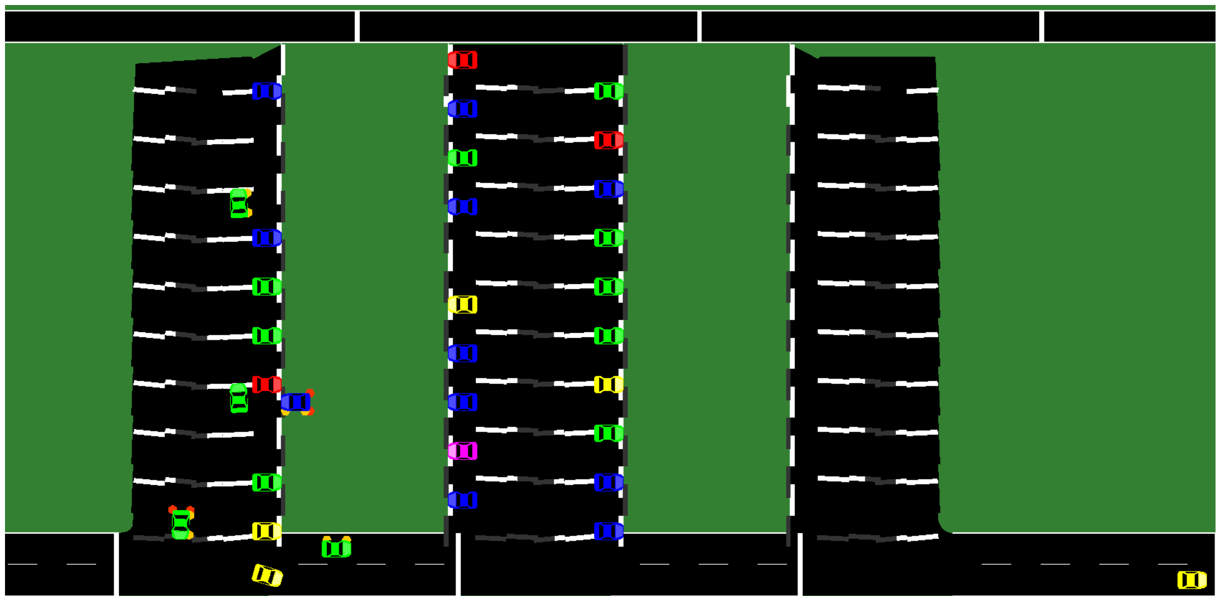
Já no cenário 2, ocorreram várias interferências por causa de congestionamentos principalmente nas fases iniciais da simulação, momento em que há o maior fluxo de carros entrando no estacionamento a procura de uma vaga e, também, em momentos em que ainda existem muitos veículos chegando à vaga, mas outros já começam a sair. Nesse cenário, os congestionamentos são um problema evidente.

8.2.3 Cenário 3

O cenário 3 prossegue com o aumento gradativo no número de vagas totais disponíveis. Neste momento, existem 30 vagas para estacionamento compreendidas entre os setores A e B.

Apesar de o número de vagas de estacionamento vir crescendo em progressão aritmética ao decorrer dos cenários, o número de veículos participantes da simulação mais que triplicou em relação ao cenário anterior utilizando 250 veículos.

Figura 13 - Simulação do cenário 3

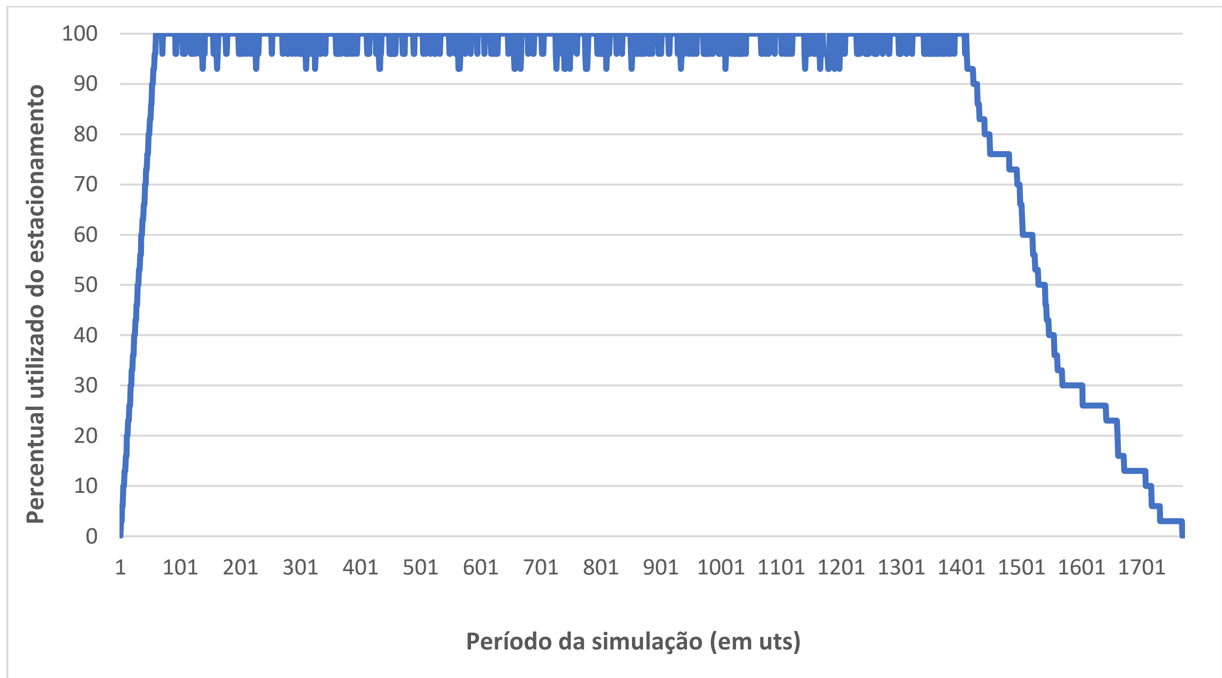


Fonte: Autoria própria

Na simulação do cenário 3, houve um enorme aumento do número de veículos na aplicação. O número de vagas disponível no estacionamento também aumentou, porém, proporcionalmente, esse aumento foi menor comparado ao aumento no número de veículos participantes.

Assim como aconteceu no cenário 2, a simulação do cenário 3 também apresentou um grande problema com congestionamentos, porém agora, esses congestionamentos ocorreram tanto no setor A quanto no setor B. Para o setor A, o congestionamento se forma quando os veículos tentam sair do setor A e entrar na estrada principal para a saída da simulação. Como o fluxo de carros desse setor para o setor B é grande, um congestionamento é criado para seguir essa via. Já no setor B, ocorre da mesma forma que na simulação do cenário 2. Assim como no setor A, também existem grandes dificuldades para seguir pela estrada de saída devido ao grande número de veículos, mas, além disso, os congestionamentos ocorrem dentro do próprio setor por causa dos veículos contido nele.

Gráfico 5 - Percentual utilizada do estacionamento no cenário 3



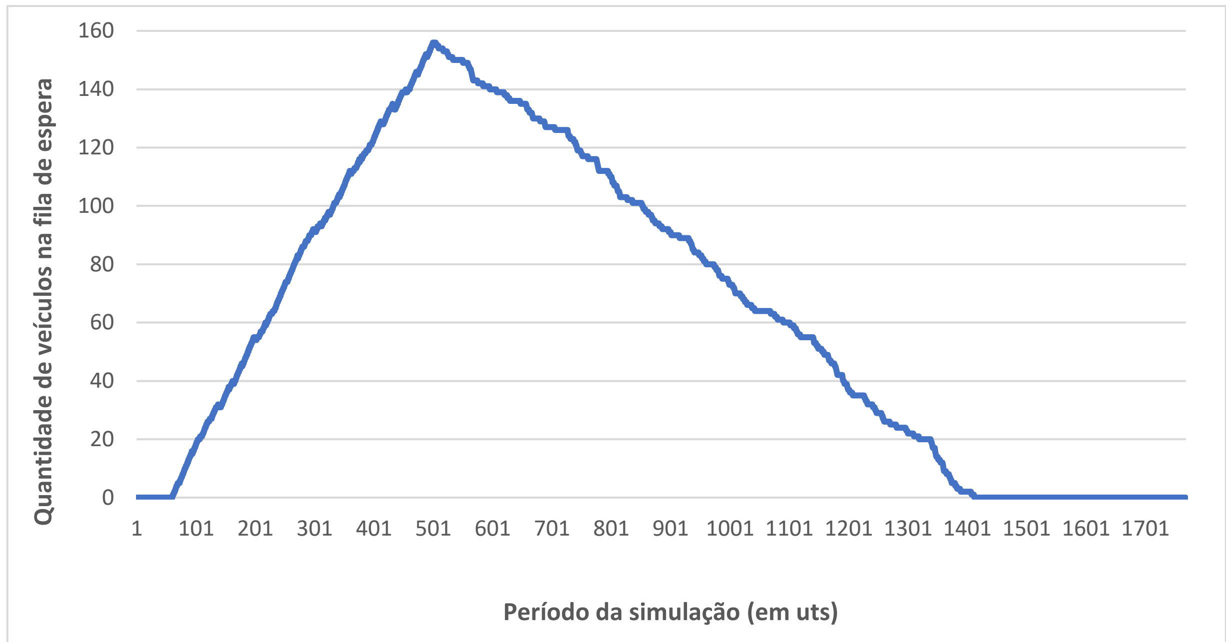
Fonte: autoria própria

O Gráfico 5 mostra a porcentagem utilizada do estacionamento ao decorrer do tempo. Assim como nos outros casos, o pico mais alto do gráfico ocorre nos primeiros momentos da aplicação devido ao alto fluxo de veículos. Como esperado, o tempo de duração até que todos os veículos consigam uma vaga de estacionamento aumentou significativamente devido ao grande aumento de veículos esperando por uma vaga na aplicação.

O gráfico permanece, na maior parte do tempo, com seus valores estáveis, sempre beirando os 100% de utilização. Ao fim da utilização do estacionamento, os

veículos vão sendo liberados até que se encerre o número de veículos na fila de espera e a porcentagem utilizada do estacionamento chegue a zero.

Gráfico 6 - Quantidade de veículos na fila de espera no cenário 3



Fonte: Autoria própria

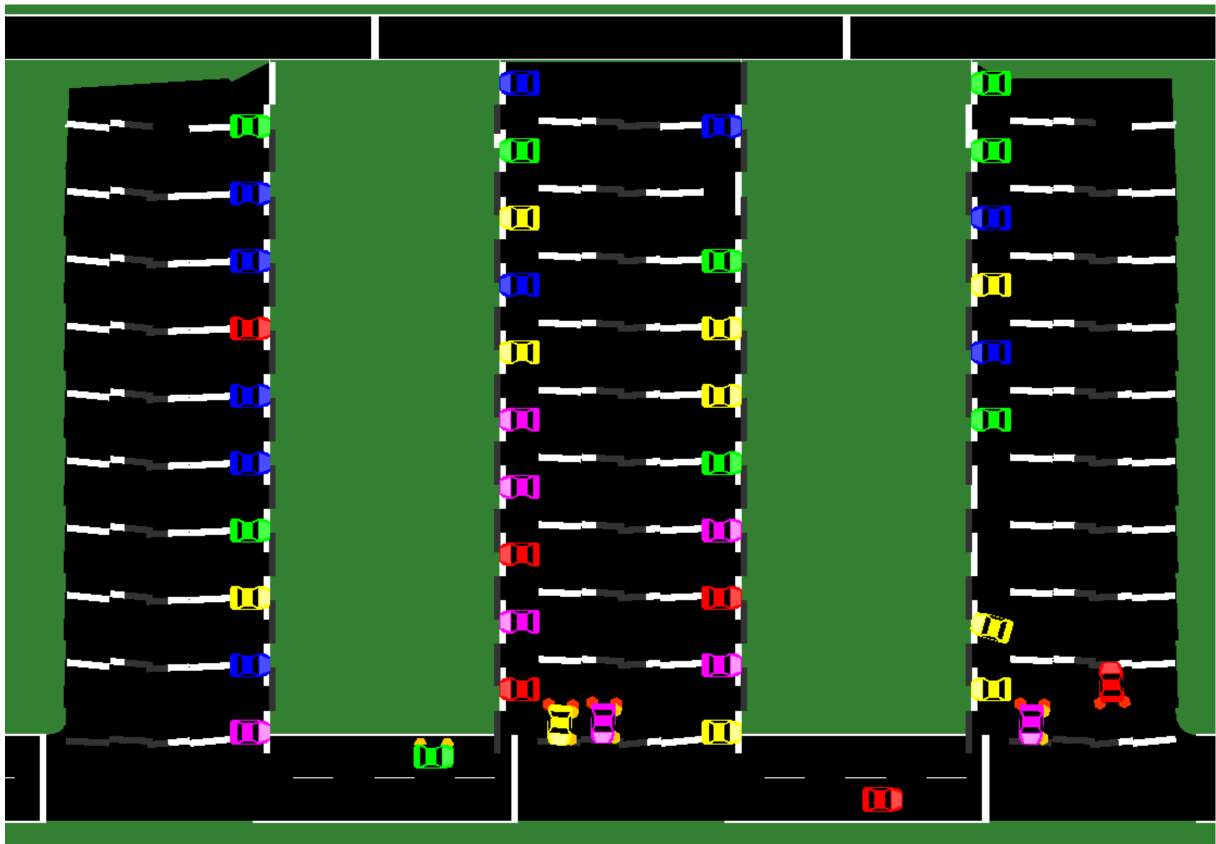
Nesse cenário, pelo motivo de o número de veículos ser muito grande comparado ao número de vagas disponíveis para estacionar, o gráfico de quantidade de veículos na fila de espera demora mais tempo para que ele consiga atingir o seu pico mais alto. Apesar disso, ele segue o mesmo padrão que nos outros cenários. O tamanho da fila de espera vem aumentando gradativamente até que atinja seu pico, que é quando os veículos estacionados começam a liberar as vagas, e, após isso, a fila começa a se esvaziar até que a fila de espera possua zero veículos.

Nesse cenário existe um demasiado número de veículos em busca de uma vaga estacionamento para poucas vagas disponíveis. Isso fez com que houvesse um número muito alto de veículos na fila de espera por uma vaga na simulação. Também foi possível observar o grande congestionamento de veículos formado nos setores A e B, que tinham grande dificuldade para sair da simulação após concluir seu tempo estacionado.

8.2.4 Cenário 4

Nesse último cenário de simulação, foi utilizado todo o conjunto de vagas de estacionamento disponíveis. Esse total compreende entre as vagas dos setores A, B e C. Em contraste ao cenário 3, que possuía um número muito grande de veículos para poucas vagas, no cenário 4 foi definido um conjunto menor de veículos solicitantes para mais vagas de estacionamento.

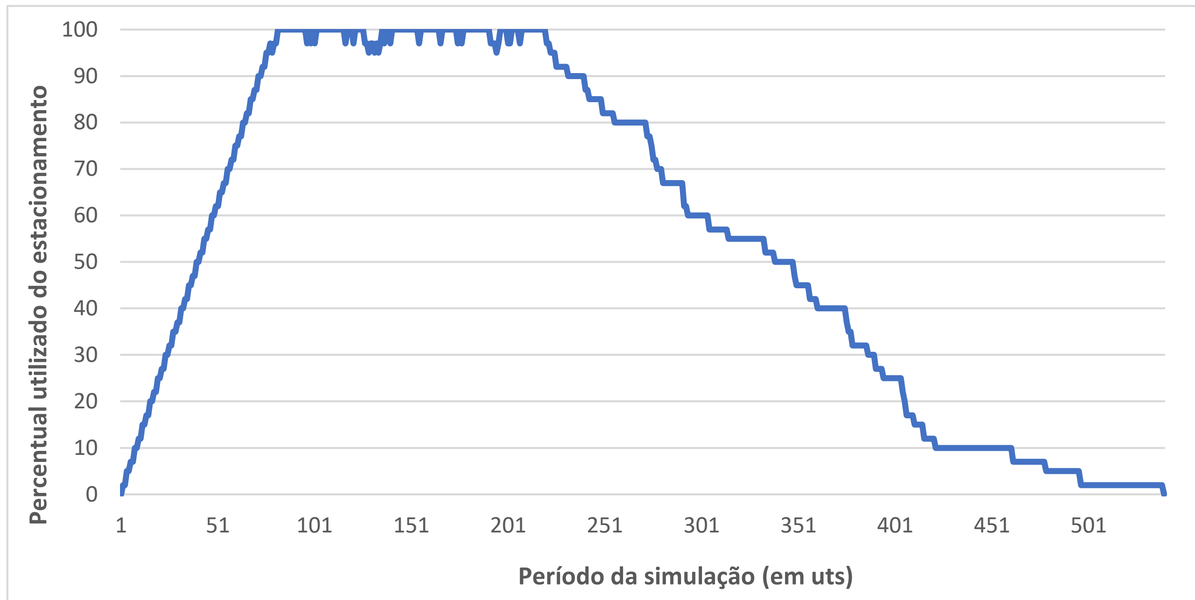
Figura 14 - Simulação do cenário 4



Fonte: Autoria própria

O cenário 4 foi o que obteve o maior número de vagas disponíveis relacionado ao número de veículos. Apenas 20 veículos seriam o número excedente à fila de espera nesse caso. Apesar disso, ainda aconteceram conflitos relacionados a congestionamento. Os pontos de congestionamento foram todos no momento em que os veículos tentam deixar o setor e ingressar na estrada para sair da simulação.

Gráfico 7- Percentual utilizada do estacionamento para o cenário 4



Fonte: Autoria própria

O Gráfico 7 mostra a porcentagem utilizada no estacionamento no cenário 4 em relação ao tempo. Nota-se que esse gráfico se difere dos outros. Devido ao pequeno número de carros solicitantes para um alto número de vagas, o gráfico não permaneceu por muito tempo com 100% do estacionamento sendo utilizado. Ele apenas sobe rapidamente devido ao grande fluxo de veículos, atinge seu ponto máximo e começa a decair até que não existem mais veículos ocupando qualquer vaga na simulação.

Gráfico 8 - Quantidade de veículos na fila de espera do cenário 4

Fonte: Autoria própria

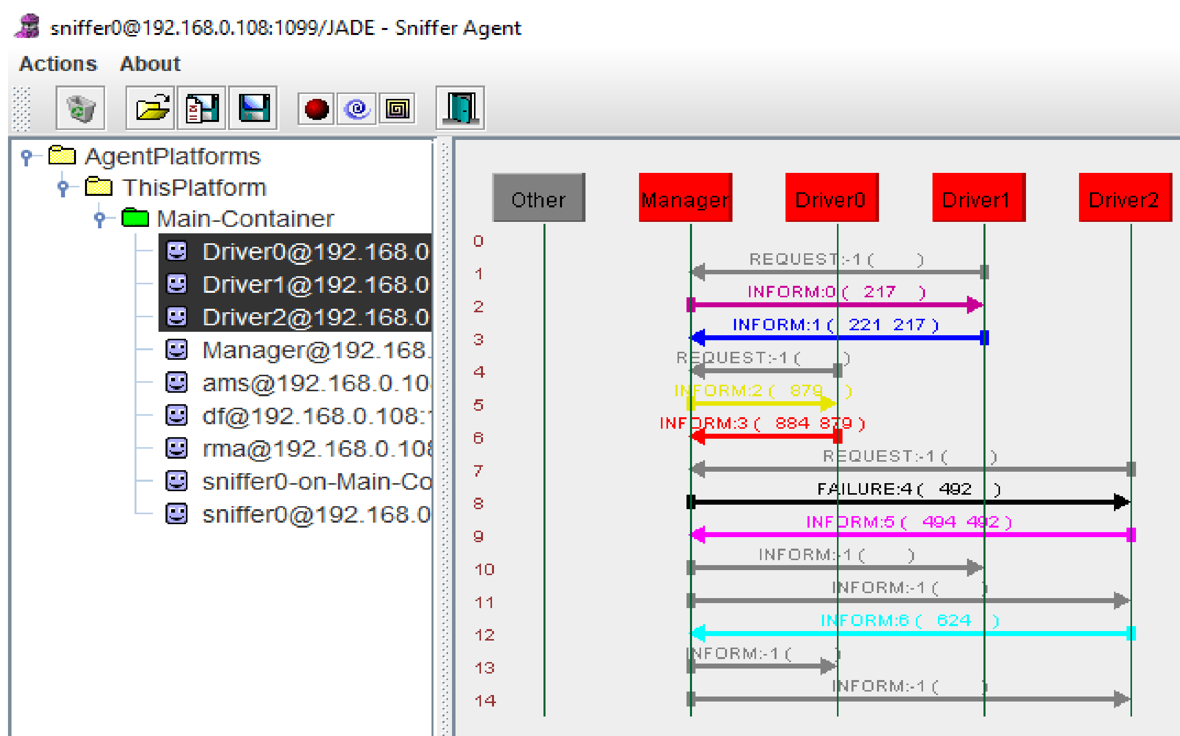
O Gráfico 8 mostra a quantidade de veículos que ocuparam a fila de espera por uma vaga no cenário 4. É possível observar que o gráfico demora certo tempo até que o primeiro veículo entre na fila, isso se deve ao maior número de vagas de estacionamento disponíveis relacionado aos cenários anteriores. O gráfico atinge seu ponto máximo próximo ao final da simulação. Em pouco também, ele é capaz de esvaziar a fila de espera até que ela chegue a zero.

O cenário 4 demonstrou ter sido o que demorou menos tempo para concluir toda a simulação devido a possuir um pequeno número de vagas. Apesar de não haver um grande número de veículos em busca de uma vaga, os congestionamentos não puderam ser evitados.

8.2.5 Comunicação entre os agentes

A Figura 15 mostra como ocorre a comunicação ocorre na simulação utilizando o *sniffer* num ambiente de três agentes. Nessa simulação foram utilizadas apenas duas vagas de estacionamento para que fosse possível demonstrar a comunicação do agente *driver* que entrou no estacionamento sem precisar passar pela fila de espera e também um *driver* que precisou passar pela fila de espera.

Figura 15 - Comunicação de agentes no sniffer



Fonte: Autoria própria

Na simulação, existem três agentes *driver*: *Driver0, Driver1* e *Driver2*. Todos eles iniciam a comunicação enviando uma mensagem de performativa *REQUEST* ao agente *manager* solicitando uma vaga de estacionamento.

Os agentes *Driver1* e *Driver2* receberam uma mensagem de performativa *INFORM* indicando que vagas foram alocadas para eles e que pode se encaminhar a elas. Logo após a mensagem do *manager* alocando a vaga, esses mesmos agentes *driver* responderam, também por mensagens de performativa *INFORM*, que estavam se encaminhando às vagas de estacionamento. Nessa simulação, o agente *Driver2* recebeu uma mensagem de performativa *FAILURE* do *Manager* indicando que o estacionamento está lotado e ele deveria aguardar na fila de espera. O *Driver2* respondeu ao *Manager* que aguardaria a vaga na fila de espera. Quando uma vaga no estacionamento for liberada para o *Driver2*, será seguido o mesmo processo de comunicação que ocorreu com os agentes *Driver0* e *Driver1.* O *manager* avisará que existe uma vaga para o *Driver2* e o mesmo pode estacionar, o *Driver2* responderá que está se encaminhando à vaga e, quando o tempo alocado se esgotar, o *Manager* avisará que o tempo está esgotado.

8.3 RESULTADOS OBTIDOS

A principal dificuldade encontrada para esse projeto se refere às rotas. Para a execução da simulação de um estacionamento em SUMO é necessário que todas as rotas para todas as vagas sejam definidas anteriormente. No caso de um estacionamento de 100 vagas, seria necessário que 200 rotas fossem escritas a mão (chegada e saída) para que fosse possível fazer a atribuição de para um veículo. Dessa maneira, um estacionamento muito grande demandaria muito esforço braçal para a definição de cada rota disponível na simulação.

Outro fator conflitante é o fato de a biblioteca *TraaS* não conseguir mudar o tempo que um veículo permanecer estacionado dinamicamente. Dessa forma, todos os tempos que os veículos permanecerão estacionados deverão ser definidos anteriormente. A alternativa utilizada para esse problema foi excluir o veículo já existente na aplicação e, quando o seu tempo de estadia de esgotasse, esse veículo seria excluído da aplicação e outro veículo idêntico seria posto em seu lugar com a

rota de saída atribuída e já seguindo por essa rota automaticamente assim que fosse inserido.

A criação de um sistema multiagente de Smart Parking integrado com o SUMO usuando o *framework* JADE se demonstrou viável produzindo variados resultados ao longo dos cenários e atigindo o objetivo geral deste trabalho.

A comunicação e ação de todos os agentes da aplicação foi efetiva em todos os cenários não demonstrando perda de informações em tempo de execução em nenhum momento.

8.4 CONSIDERAÇÕES FINAIS

Esse capítulo demonstra a viabilidade da interligação do *framework* JADE com a ferramenta SUMO por intermédio da biblioteca TraaS para a construção de um *Smart Parking* baseado em agentes. Foram propostos cenários com quantidades de vagas e agentes *drivers* diversificados a fim de se obter diferentes resultados e poder fazer uma análise sobre o comportamento dos agentes e a comunicação entre eles. Apesar de fatores ruins ocorridos como trânsito, por exemplo, foi demonstrado em todos os cenários propostos que a simulação foi possível.

9 CONCLUSÃO

Este trabalho tem como objetivo fazer a interligação do *framework* JADE com a ferramenta SUMO em um domínio de *Smart Parking* baseado em agentes. Dessa forma, seria possível fazer uma simulação de estacionamento que fosse gerenciada autonomamente por um *manager*, sem a necessidade de intervenção humana. Por meio deste trabalho foi possível testar a viabilidade dos objetivos desejados.

Com a ferramenta SUMO foi possível observar graficamente os resultados de uma simulação completa de um estacionamento. Por se tratar de uma ferramenta gratuita e de código-fonte aberto, melhorias são sempre feitas por seus colaboradores. A ferramenta foi de fácil aprendizado e utilização já que para a criação dos ambientes de simulação, era apenas necessária a criação de arquivos XML contendo as informações necessárias para o desenvolvimento. A interface gráfica do SUMO dispões de algumas funcionalidades que podem ser alteradas facilitando a análise de diversos aspectos da simulação.

O intuito desse projeto foi desenvolver um protótipo de um SMA para simular um estacionamento autonomo. A aplicação deveria por ela mesma decidir qual seria o melhor comportamento a ser tomado em cada situação da simulação. Dessa forma foi utilizado o *framework* JADE para atingir esse objetivo. Esse *framework* não apresentou nenhum impeditivo que dificultasse a interação entre a simulação do SUMO e o código-fonte em si.

Para a integração o *framework* JADE e a ferramenta SUMO se fez necessária a utilização da biblioteca *TraaS* como *middleware*, uma versão de *TraCI*, mas escrita na linguagem JAVA. Essa biblioteca não possui uma documentação publicada o que, de certa forma, dificultou o seu aprendizado. Porém, com a disponibilização de todas as classes da biblioteca para leitura, foi possível fazer o acompanhamento do que cada função dessa biblioteca pode fazer ou, até mesmo, alterá-las.

Este projeto produziu resultados interessantes e mostrou a viabilidade do desenvolvimento de um estacionamento gerenciado autonomamente utilizando o *framework* JADE e a ferramenta SUMO para análise dos resultados. Com os resultados obtidos nos cenários do Capítulo 8, também foi possível verificar que existem melhorias a serem feitas para trabalhos futuros. Podem ser verificadas formas a se trabalhar com mais de um lote de setores de estacionamentos ao decorrer da simulação trabalhando de forma com que possa haver interação entre eles. Também

devem ser estudadas diferentes formas de tráfego a fim de melhorar a fluidez do trânsito evitando congestionamentos ou até mesmo, ser realizado um refinamento de código para a melhoria da eficiência da simulação em geral. Outros estudos a serem feitos para projetos futuros seriam sobre a interação de mais tipos de componentes da simulação. Este projeto focou apenas na interação de veículos com as mesmas propriedades numa simulação de estacionamento. Como o próprio protótipo da *CityMobil* utiliza, também podem ser utilizados ônibus, pedestres, diferentes tipos de carros com propriedades diferentes, interações com pontos de ônibus, verificações de diferença de circulação entre um motorista que dirige bem com outro que não etc.

Dessa forma foi possível concluir que este projeto é viável ao objetivo proposto, uma simulação de um estacionamento guiado por um gerenciador autônomo. A aplicação não apresentou perda de informações ou instabilidade durante todo o processo já que a troca de mensagens entre os agentes sempre ocorria como o esperado independentemente do número de agentes na simulação. Em todos os cenários foi possível a alocação de vagas de estacionamento para todos os veículos independentemente da quantidade de vagas de estacionamento dispostas e do conjunto de veículos que irão solicitar uma vaga.

Assim como para esse projeto e, principalmente, simulações com mais de um agente *manager* ou, também, mais de um estacionamento disponível, pode ser utilizado, como um trabalho futuro, o registro desses agentes no DF (*Directory Facilitator*) do JADE. Dessa maneira, os agentes *manager* conseguirão manter periodicamente uma lista dos agentes atuantes na simulação e informações sobre os mesmos.

10 REFERÊNCIAS

AIMSUN LIVE. **Aimsun Live for real-time traffic management.** Disponível em: <https://www.aimsun.com/aimsun-live/papers/>. Acesso em: 01/10/2020.

ALVARES, L.; SICHMAN, J. **Introdução aos Sistemas Multiagentes.** Jornadas de Atualização em Informática – XVI JAI. 1997, Brasília.

AZEVEDO, T.; et. al. JADE, TrasMAPI and SUMO: A tool-chain for simulating traffic light control. **Proceedings of the 8th international Workshop on Agents in Traffic and Transportation**. 2014, Paris, França.

BAINES, V; PADGET, J. A situational awareness approach to intelligent vehicle agents. **SUMO2014 – Modeling Mobitlity with Open Data**. Berlim - Alemanha, v. 24, p. 1-17, mai, 2014.

BATISTA JÚNIOR, A. A. COUTINHO, L. R. Incorporating explicit coordination mechanisms by agents to obtain green waves. **Advanced Methods and Technologies for Agent and Multi-Agent Systems.** v. 252, p. 137-145, 2013.

BATTY, M.; et. al. Smart Cities of the Future. **UCL Working Papers Series,** Londres - Inglaterra, v. 118, p. 1-40, out. 2012.

BELLIFEMINE, F; et. al. **Developing multi-agent systems with JADE**. Wiley Series in Agent Technology, 2007.

BUSSMAN, S. DEMAZEAU, Y. **An agent model combining reactive and cognitive capabilities**. In: Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS-94), Munich,Germany, 1994.

CASTRO, L. **Modelagem e Implementação de um Sistema Multiagente Utilizando a Plataforma JaCaMo Para Alocação de Vagas em um Estacionamento Inteligente**. 2015. 79 f. Trabalho de Conclusão de Curso de Ciência da Computação, Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2015.

DENNET, D. **The Intentional Stance.** MIT Press, 1987.

DEUTSCHE TELEKOM. MWC 2014: Deutsche Telekom and Pisa start Smart City Project. Disponível em: <https://www.telekom.com/en/media/mediainformation/archive/mwc-2014--deutsche-telekom-and-pisa-start-smart-city-project-360446>. Acesso em: 01/10/2020.

DI NAPOLI, C.; et. al. **Negotiating Parking Spaces in Smart Cities**. In: ADVANCES IN PRACTICAL APPLICATIONS OF HETEROGENEOUS MULTI-AGENT SYSTEMS. Salamanca. Espanha, 2014.

DUCHEIKO, Felipe. F. Pinz, A. P; Alves. V. Implementação de Modelo de Raciocínio e Protocolo de Negociação para um Estacionamento Inteligente com Mecanismo de Negociação Descentralizado. **REVISTA JUNIOR DE INICIAÇÃO CIENTÍFICA EM CIÊNCIAS EXATAS E ENGENHARIA**, v. 1, p. 25-32, 2018.

FERBER, J. **Les Systèmes mult-agents: vers une verse intelligence collective.** Editora: InterEditions, 1996.

FIPA. Welcome to FIPA. Disponível em: <http://www.fipa.org>. Acesso em: 01/10/2020.

FIPA. ACL Message Structure Specification. Disponível em: <http://www.fipa.org/specs/fipa00061/SC00061G.html>. Acesso em: 01/10/2020.

GENESERETH, M. R.; KETCHPEL, S. P. **Software Agents**. Communications of the ACM, 1994.

HEIJIMEIJER, A. V. H. **Interligação entre a Ferramenta de Simulação SUMO e o Projeto MAPS**. 2016. 85f. Trabalho de Conclusão de Curso de Ciência da Computação, Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2016.

JaCaMo. The JaCaMo approach. Disponível em: <http://jacamo.sourceforge.net/?page_id=40>. Acesso em: 01/10/2020.

JADE. Java Agent DEvelopment. Diponível em: <http://jade.tilab.com/>. Acesso em: 01/10/2020.

JASON. A Java-based interpreter for na extended version of AgentSpeak. Disponível em: <http://jason.sourceforge.net/wp/>. Acesso em: 01/10/2020.

JUNCHEM, M.; BASTOS, R; Arquiteturas de Agentes**. Technical Report Series**. 2011, Porto Alegre. Brasil.

LABROU, Y. et. al. **Agent Communication Languages: The Current Landscape**. *IEEE Intelligent Systems*, 1999.

MAYFIELD, J. et. al. **Evaluating KQML as an Agent Communication Language**. Springer-Verlag, Heidelberg, 1996.

MELLADO, A.; ALVES, G. V.; Pinz, A. P. Uma comparação entre soluções de smart parkings baseados em agentes inteligentes. In: WESAAC 2019 - 13th Workshop-School on Agents, Environments, and Applications, Florianóplois. **Anais 13th Workshop-School on Agents, Environments, and Applications**. Florianopólis: Editora da UFSC, 2019. v. 1. p. 1-6.

PARAMICS. Paramics Microsimulation – 3D traffic simulation. Disponível em: <https://www.paramics.co.uk/en/>. Acesso em: 01/10/2020.

PTV VISSIM. Traffic Simulation Software | PTV Vissim | PTV Group. Disponível em: <https://www.ptvgroup.com/en/solutions/products/ptv-vissim/>. Acesso em: 01/10/2020.

RUSSEL, J; NORVIG, P. **Artificial Intelligence - A Modern Approach***.* 3rd ed. Prentice Hall, 2010.

SAKURADA, L.; et. al. Development of Agent-Based CPS for Smart Parking Systems. In**: IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society,** 2019, Lisboa, Portugal. IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society, 2019. v. 1. p. 2964-2969.

SOARES, G. *et. al.* Agent-based Traffic Simulation using Sumo and JADE: an integrated platform for Artificial Transportation Systems. **Simulation on Urban Mobility User Conference**. Berlim, Alemanha. 2014.

SHOHAM, Y; **Agent-oriented Programming**. Artificial Intelligence. 51-92. 1993.

SUMO Wiki. Simulation of Urban MObility. Disponível em: <https://sumo.dlr.de/docs/index.html>. Acesso em: 01/10/2020.

TRAAS. TraCI/TraaS**.** Disponível em: <https://sumo.dlr.de/docs/TraCI/TraaS.html>. Acesso em: 01/10/2020.

WOOLDRIDGE, M. **Intelligent Agents. Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence.** MIT Press, 1999.

WOOLDRIDGE, M. **An Introduction to MultiAgent Systems**. 2nd ed. [S.1.]: Wiley Publishing, 2009.

APÊNDICE A - INSTALAÇÃO DO SUMO

A versão da ferramenta SUMO utilizada nesse projeto é a 1.3.1. Essa era a última versão lançada até o momento da realização desse projeto. Apesar de a última versão ter sido a escolhida para esse projeto, ela não possui os arquivos do modelo de simulação do *CityMobil*. Este está disponível na versão 0.27.1 da ferramenta SUMO.

Os arquivos para *download* da ferramenta SUMO podem ser encontrados em: https://sumo.dlr.de/docs/Downloads.html.

Existem diversas formas de *downloads* para a ferramenta, escolha a que for de sua preferência.

Após a instalação da ferramenta será possível fazer a verificação de toda sua organização de pastas compostas por sua documentação, dlls, arquivos de exemplo (onde se encontra o protótipo *CityMobil*) etc.

Para a utilização da ferramenta em linha de comando é necessário que se declare a variável SUMO_HOME nas variáveis de ambiente do sistema.

Utilizando o Windows 10, deve-se:

- Ir em “Painel de Controle”;
- Ir em “Sistema e Segurança”;
- Ir em “Sistema”;
- Clique em “Configurações avançadas do sistema”;
- Em avançado, ir em “Variáveis de ambiente…”;
- Ir em novo “Novo...” e criar uma variável de nome “SUMO_HOME” com valor apontando para a pasta de instalação da ferramenta SUMO;
- Selecione “Ok” e na variável “Path”, deve-se clicar em “Editar...”;
- Adicionar uma nova variável referenciando a mesma variável SUMO_HOME definida anteriormente;
- Por fim, aperte “Ok” para todas as janelas abertas no processo.

A partir desse momento, é possível executar comandos usando a própria linha de comando para chamados de funções da ferramenta SUMO, como a própria NETCONVERT, muito utilizada nesse projeto.

Quando é feito o *download* da versão 1.3.1 da ferramenta SUMO, a biblioteca *TraaS* também vem baixada dentro da pasta *bin* que está presente na pasta raiz da

instalação do SUMO. Diferentemente de seu antecessor, *TraCI4J*, a biblioteca *TraaS* vem no formato *.jar* pronto para a importação.