UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

CAVSIM – CONNECTED AND AUTONOMOUS VEHICLE SIMULATOR

MATIAS GUIOMAR HENSCHEL

BLUMENAU
2017

MATIAS GUIOMAR HENSCHEL

CAVSIM – CONNECTED AND AUTONOMOUS VEHICLE SIMULATOR

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Daniel Theisges dos Santos, Mestre - Orientador

BLUMENAU
2017

CAVSIM – CONNECTED AND AUTONOMOUS VEHICLE SIMULATOR

Por

MATIAS GUIOMAR HENSCHEL

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Daniel Theisges dos Santos, Mestre – Orientador, FURB
Membro:	Prof. Aurélio Faustino Hoppe, Mestre – FURB
Membro:	Prof. Marcel Hugo, Mestre – FURB

Blumenau, 5 de julho de 2017

Dedico este trabalho a todos que estiveram presentes na minha vida, e fizeram de mim quem eu hoje sou.

AGRADECIMENTOS

A Deus, que me trouxe até aqui, e me deu a oportunidade de ser quem eu sou.

À minha família, que me foi suporte em toda necessidade e esteve comigo nos momentos mais alegres e tristes de minha vida.

Ao meu orientador, que soube me incentivar para o sucesso me guiando nas horas de dúvida.

Aos meus amigos, que me motivaram a persistir no trabalho e foram pacientes nos dias frustrantes.

Aos diversos usuários e participantes de fóruns online (em especial StackOverflow), que me economizaram muitas horas de pesquisa e frustração.

“You could either watch it happen or be part of it.”

Elon Musk

RESUMO

Nos últimos anos, estudos sobre troca de informações entre veículos que transitam em vias públicas têm sido realizados com o propósito de reduzir acidentes e melhorar a eficiência no trânsito. Este trabalho apresenta o desenvolvimento de um sistema multiagente para simulação de veículos conectados em cruzamentos, com objetivo de remover a necessidade de semáforos. O sistema foi desenvolvido na linguagem de programação C# e utilizou a ferramenta Boris.NET para fazer a comunicação entre os agentes. O usuário pode especificar os ambientes desejados em arquivos JSON que são lidos na inicialização da simulação. Também foi criada uma interface gráfica 2D para visualizar a simulação, de forma que possa ser acompanhado um agente ou um ponto do mapa. A partir de casos de teste foram feitas comparações entre veículos conectados e veículos manualmente dirigidos. A análise dos dados provou a maior eficiência de veículos que se comunicam entre si, mostrando sua maior eficiência em cruzamentos. Pode-se concluir que estudos na área de veículos conectados são de grande valor para o desenvolvimento da área de transporte, e sua simulação pode ajudar a comprovar e testar teorias previamente afirmadas por estudiosos.

Palavras-chave: Sistema multiagente. Simulação de veículos. Veículos conectados. Veículos autônomos.

ABSTRACT

In recent years, research over information exchange between vehicles has been growing, with the goal to improve safety and efficiency in traffic. This project describes the development of a multi-agent system for simulation of connected vehicles in road crossings, in order to remove the need for traffic lights. The system was developed using the C# programming language and Boris.NET platform for communication between agents. The user can specify the desired environments using JSON files which are read at the simulation startup. A 2D graphical interface was also created to view the simulation, so that it can follow an agent or stay at a map position. Comparisons were made based on test cases between connected and manually driven vehicles. Data analysis proved the efficiency of vehicles that communicate with each other, showing their superior efficiency at crossroads. Finally, this project concludes that studies related to connected vehicles are of high value for transportation development and that simulation may help prove concepts and theories previously affirmed.

Key-words: Multi-agent system. Vehicle simulation. Connected vehicles. Autonomous vehicles.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADAS – Advanced Driver Assistance Systems

ATMS – Advanced Traffic Management System

FAD – Fully Automated Driving

FIPA – Foundation for Intelligent and Physical Agents

HAD – Highly Automated Driving

IDE – Ambiente de Desenvolvimento Integrado

KQML – Knowledge Query and Manipulation Language

MER – Modelo Entidade-Relacionamento

RF – Requisito Funcional

RNF – Requisito Não-Funcional

SDK – Software Development Kit

SMA – Sistemas Multiagente

SOAP – Simple Object Access Protocol

V2I – Veículo para Internet

V2R – Veículo para Infraestrutura de Ruas

V2S – Veículo para Sensor

V2V – Veículo para Veículo

WCF – Windows Communication Foundation

WSDL – Web Service Description Language

XML – Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Segundo Cheng (2014), os veículos são parte indispensável na vida moderna. Porém, Azevedo (2015) mostra que acidentes de trânsito ainda são uma das principais causas de morte no Brasil, recebendo o oitavo lugar na lista com quase 42 mil mortes no ano de 2013.

Diversos estudos como o de Komada et al. (2013) apresentam o risco que o fator humano traz ao transporte. Azevedo (2015) também cita excesso de velocidade, imprudência e consumo de álcool como alguns dos principais fatores que causam mortes no trânsito. Como possível solução para este problema, a tecnologia vem se integrando cada vez mais aos veículos, com promessas de melhor segurança e eficiência. A automatização da direção é um exemplo de tecnologia trazida para resolver estes problemas.

Segundo Cheng et al. (2014), automóveis equipados com a capacidade de comunicação, também chamados de veículos conectados, tendem a ser a próxima fronteira para a revolução automotiva, com estimativas de que "o mercado global [de veículos conectados] deve chegar a 131,9 bilhões de dólares em 2019" (TRANSPARENCY MARKET SHARE, 2013 apud CHENG et al., 2014, p. 289, tradução nossa).

Veículos conectados não somente podem coletar dados detalhados sobre si, como movimentos precisos e rota, mas também compartilhá-los com os veículos ao seu redor e com o ambiente para um controle de tráfego mais ágil e seguro (LEE; PARK, 2012, p. 81). Em um cruzamento, por exemplo, os veículos podem trocar informações sobre o momento de chegada e adaptar suas rotas, removendo a necessidade de semáforos.

Para que essas tecnologias emergentes sejam melhor compreendidas, fazem-se úteis simulações. Simuladores trazem informações quanto a eficiência e correto funcionamento ao imitarem o real comportamento de uma situação, fazendo possível que alterações sejam feitas ainda em fase de projeto, o que pode diminuir custos (CRAIG, 1996). No caso de simulação de veículos conectados, a utilização de sistemas multiagentes é comum, pelo potencial de resolver problemas complexos do mundo real (JIN et al., 2012), e características como a comunicação são essenciais para que diferentes agentes possam cooperar em busca de um objetivo comum (HÜBNER, 1995, p. 17).

Tendo conhecimento desta conjuntura, este trabalho visa desenvolver um simulador para melhor compreender o cenário que as tecnologias de veículos conectados podem trazer. O simulador utilizará um ambiente multiagente para simular um cruzamento com veículos conectados comparando resultados com veículos dirigidos, cujo foco está primeiramente na segurança e em seguida na eficiência do tráfego.

1.1 OBJETIVOS

O objetivo deste trabalho é desenvolver um sistema que simula um ambiente multiagentes de veículos inteligentes, que se comunicam entre si, removendo a necessidade de semáforos.

Os objetivos específicos são:

a) desenvolver um ambiente de simulação multiagente para a plataforma Windows otimizado para a simulação de veículos;
b) desenvolver um sistema que simula o tráfego de veículos conectados em cruzamentos, comparando-os com veículos dirigidos;
c) criar uma interface gráfica para acompanhar um agente selecionado pelo usuário.

1.2 ESTRUTURA

O trabalho divide-se em 4 capítulos. O primeiro capítulo introduz o trabalho apresentando também seus objetivos. O segundo capítulo traz a fundamentação teórica para o trabalho, trazendo maior conhecimento para as áreas de veículos, sua automatização, simulação de veículos e sistemas multiagente, apresentando em seguida trabalhos correlatos a este. O terceiro capítulo detalha o desenvolvimento do sistema, descrevendo seus requisitos, especificações, resultados e funcionamento da aplicação. O quarto e último capítulo relata as conclusões do trabalho, juntamente com suas possíveis extensões.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo traz conhecimentos sobre as áreas teóricas do trabalho. A seção 2.1 apresenta informações sobre veículos e direção automática, comentando o avanço tecnológico na área de veículos autônomos. Na seção 2.2 são destacados pontos a respeito de veículos conectados, citando suas vantagens assim como características sobre a simulação dos mesmos. Na seção 2.3 são apresentados conceitos a respeito de sistemas multiagente, além de comentários sobre a comunicação entre agentes e simulações em tempo real. A seção 2.4 apresenta os trabalhos correlatos que foram utilizados como referência para desenvolvimento deste.

2.1 VEÍCULOS E DIREÇÃO AUTOMÁTICA

Com o rápido desenvolvimento tecnológico e econômico, “veículos são [hoje] parte indispensável da vida moderna” (CHENG et al., 2014, p. 289, tradução nossa). O número de licenciamentos de veículos no Brasil vem crescendo nos últimos 20 anos (ANFAVEA, 2016), consequentemente o número de congestionamentos aumentou. Segundo Albuquerque (2008), somente engarrafamentos na cidade de São Paulo geram custos bilionários, causados por horas perdidas de trabalho, aumento do custo de transporte e problemas de saúde relacionados com a poluição.

Para resolver estes problemas, a tecnologia está se integrando cada vez mais aos veículos, crescendo o estudo sobre veículos autônomos. Segundo Barbosa, Oliveira e Souza (2010 apud KRAUSS NETO, 2013), há uma convergência global no sentido da utilização de carros elétricos e automóveis equipados com a capacidade de comunicação tendem a ser a próxima fronteira para a revolução automotiva.

Nesse sentido, existem diferentes classificações quanto ao nível de integração da tecnologia aos veículos. Hesse (2014, tradução nossa) classifica em cinco categorias, ordenadas com nível de automação crescente:

a) somente direção: o motorista exerce total controle do veículo;
b) direção assistida: função de direção é exercida pelo motorista e demais tarefas podem ser assistidas com tecnologia;
c) parcialmente automático: o sistema pode assumir controle da direção, mas o motorista deve estar pronto para tomar controle a qualquer momento. A tecnologia nesse nível também é conhecida como Direção Altamente Automática (Highly Automated Driving - HAD);
d) alta automação: o motorista não tem mais a necessidade de monitorar o sistema

durante todo tempo, e, em caso de necessidade, o veículo solicita o motorista com antecedência;

e) totalmente automático: o sistema tem controle total do veículo e, se em algum momento for requisitado o motorista e este não estiver presente ou não atender, o veículo retorna a uma condição de mínimo risco. Essa tecnologia também pode ser chamada de Direção Completamente Automática (Fully Automated Driving - FAD).

O crescimento no nível da automação traz maior segurança e transforma o motorista cada vez mais em um passageiro do veículo. Com a utilização de Sistemas de Assistência Avançada de Direção (Advanced Driver Assistance Systems - ADAS), os veículos são capacitados de diversos sensores, câmeras e processadores para auxiliar na sua automação (INTEL, 2017). Este projeto visa simular veículos com FAD, sem a necessidade de um motorista ou passageiros.

2.1.1 Veículos Autônomos

"As primeiras iniciativas de robotizar veículos iniciaram nos anos 80, na Alemanha, [...] na Universidade de Bundeswehr Munich (UniBW)" (FIGUEIREDO, 2009, p. 5, tradução nossa). Mas principalmente na última década o assunto vem cada vez mais se difundindo na população, com matérias em populares portais de notícias, como The New York Times (BOUDETTE, 2017), Forbes (NEWMAN, 2016) e The Washington Post (WADHWA, 2017).

As promessas que essa inovação no mercado automotivo traz são muitas, desde maior segurança nas vias até fatores econômicos favoráveis. Pesquisas como a de Mersky e Samaras (2016) mostram como veículos autônomos tem um consumo de combustível reduzido comparado com condutores humanos. Friedrich (2015 apud BECKER et al., 2016) sugere melhorias de capacidade nas vias devido à maior eficiência no transporte.

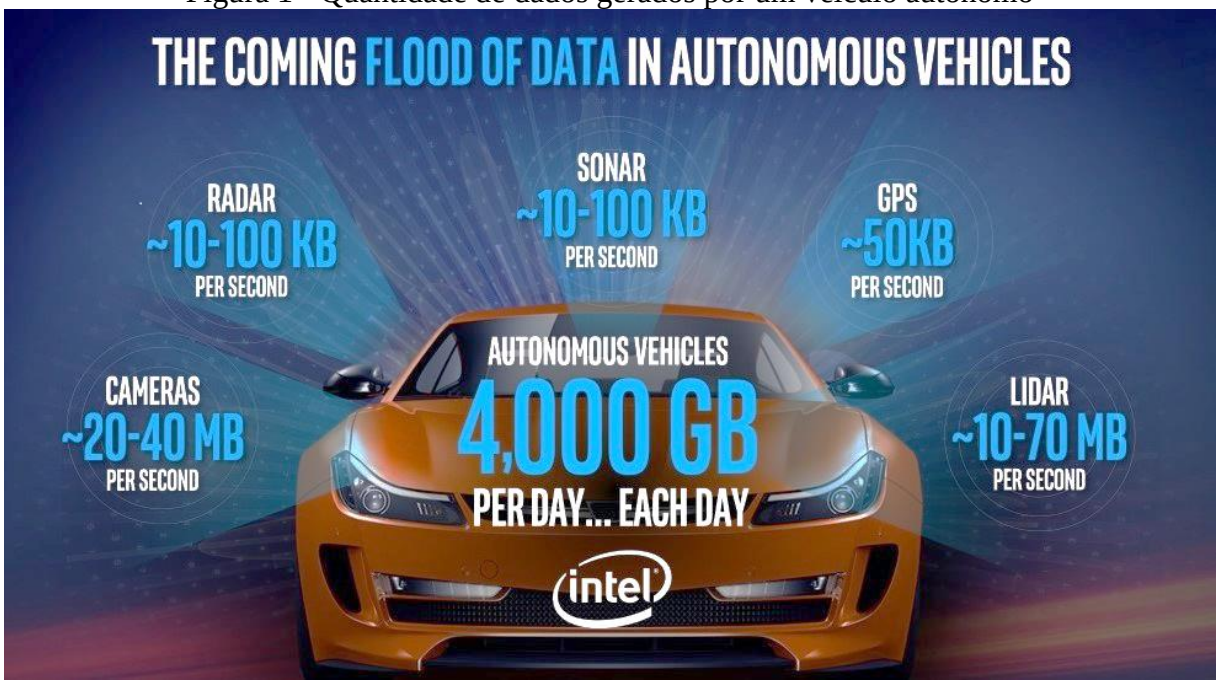
No fator segurança, os veículos autônomos removem o fator humano, responsável pela grande maioria dos acidentes, que podem ser causados por fatores como embriaguez, estresse e sono. Azevedo (2015) mostra os acidentes de trânsito como 8ª (oitava) principal causa de mortes no Brasil, citando imprudência como um dos fatores mais relevantes. Komada et al. (2013) apresentam um estudo que relaciona a falta de sono com o aumento no número de acidentes.

Segundo Liden (2016), em um cenário ideal, o passageiro seria responsável apenas por indicar o destino do veículo e este veículo o levaria em segurança, sem a presença de um

condutor humano. Isso possibilitaria que veículos autônomos oferecessem a oportunidade de transporte até mesmo a crianças, idosos e deficientes físicos (BECKER et al. 2017).

Porém, para que esta tecnologia cresça existem diversas barreiras a serem superadas. Segundo Intel (2017), "A construção de um veículo autônomo e seguro requer [...] habilidade para processar enormes quantidades de dados do ambiente". Intel Corporation (2017) cita ainda que, além do alto poder de processamento, a tecnologia para veículos autônomos "requer extensiva estrutura de serviço de dados e conectividade. Cada veículo autônomo vai gerar grande quantidade de dados" e, para poderem trocar essa grande quantidade de informações a cada segundo, "precisam de banda larga nos dois sentidos que seja confiável, assim como centrais de dados com capacidade de receber essas informações". A Figura 1 demonstra a quantidade de dados prevista que um veículo autônomo gerará e processará diariamente.

Figura 1 - Quantidade de dados gerados por um veículo autônomo



Fonte: Landau (2017).

Mesmo com a grande quantidade de desafios, são tantos pontos positivos em relação a veículos autônomos que Wadhwa (2017, tradução nossa) sugere que no futuro "a pergunta vai ser se humanos sequer deveriam tomar controle do veículo".

2.2 VEÍCULOS CONECTADOS

São definidos como conectados os "veículos que tem a capacidade de se comunicar sem fio com seus ambientes internos ou externos" (CHENG et al., 2014, p. 289, tradução nossa). Tais veículos apresentam vantagens de não somente coletar dados precisos sobre si,

como movimentos precisos e rota, mas também compartilhá-los com os veículos ao seu redor e ambiente para um controle de tráfego mais ágil e seguro (LEE; PARK, 2012).

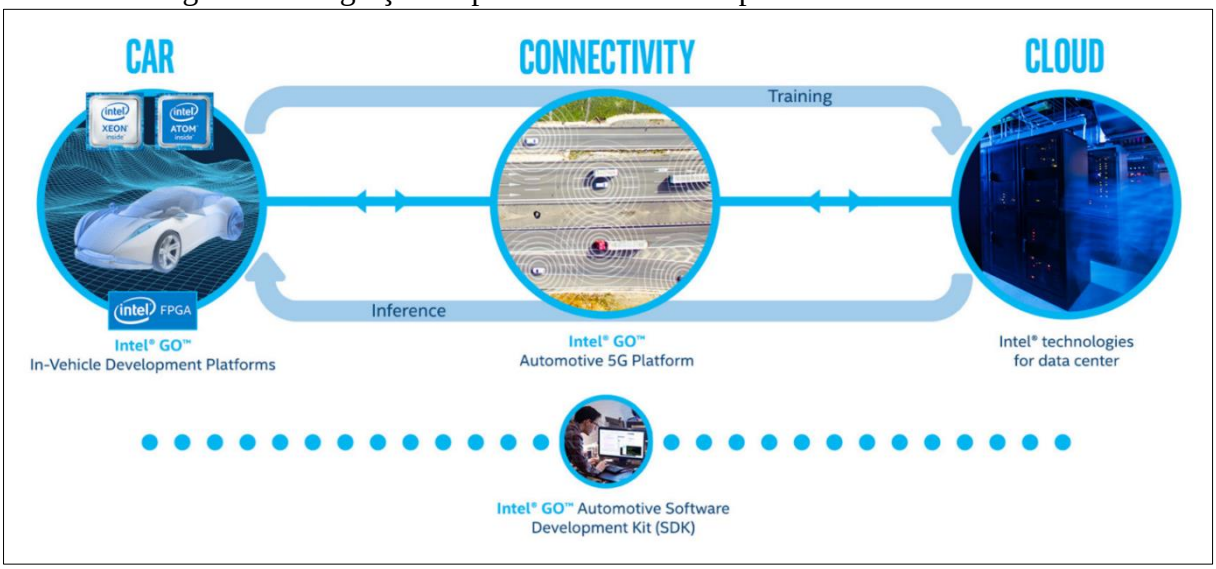
Estes veículos, ao trabalharem de forma cooperativa, podem utilizar da troca de informações para buscarem uma solução conjunta no trânsito. Informações como tráfego, velocidade do veículo e localização podem influenciar outro veículo a otimizar sua rota. Além disso, dados precisos sobre tempo de chegada em um cruzamento e um mediador podem eliminar totalmente a necessidade de semáforos (JIN et al., 2012).

Cheng et al. (2014) explica que os veículos conectados apresentam diferentes interações, podendo dividir-se em:

a) veículo-sensores (Vehicle to Sensor - V2S): é a comunicação feita internamente no veículo, com sensores no motor, freios e outros. Exige grande precisão e baixo tempo de resposta para garantir a maior segurança possível;
b) veículo-veículo (Vehicle to Vehicle - V2V): é utilizada para se comunicar diretamente com outros veículos. Algumas de suas funções são identificar trajetória dos veículos próximos, prever frenagens e garantir segurança na troca de faixas e vias;
c) veículo-infraestrutura (Vehicle to Road Infrastructure - V2R): se refere a comunicação dos veículos com as estruturas da via, como semáforos, sensores nas rodovias e sinalização;
d) veículo-internet (Vehicle to Internet - V2I): o veículo poderia utilizar V2I para conectar-se em servidores que provêm serviços, como cálculo de rotas, consultas de tráfego e previsão do tempo. Além disso, “o acesso a partir dos veículos permite que usuários acessem serviços comuns na internet”, como redes sociais ou sites de notícias.

É importante ressaltar que existem pesquisas que não citam a comunicação veículo-internet, usando a sigla V2I como veículo-infraestrutura. Isso se dá por que ainda existem muitas barreiras para se alcançar uma conexão estável, como mostram os estudos de Calderon et al. (2014) e Cheng et al. (2014). Entretanto, Landau (2017) lembra que “Para liberar o verdadeiro potencial da direção automatizada é necessária uma rede sem fio confiável, robusta e disseminada”, e por isso empresas estão apostando na próxima geração de comunicação de dados sem fio para alcançar esses patamares: o 5G. A Figura 2 apresenta uma tecnologia proposta que utiliza o 5G para fazer a comunicação dos veículos com a nuvem.

Figura 2 - Integração da plataforma Intel GO por meio de conexão 5G

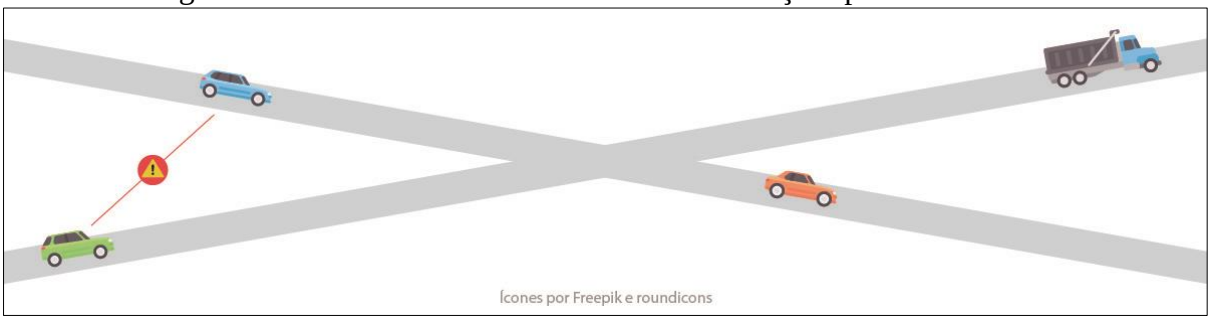


Fonte: Intel Corporation (2017).

Além das diversas vantagens dos veículos autônomos citadas na seção 2.1.1, os veículos conectados possuem características próprias que podem oferecer vantagens em um grande leque de situações. Por possuírem a capacidade de comunicação, esses veículos têm uma capacidade muito maior de conhecer o ambiente ao seu redor, prevendo algumas situações e evitando outras. Abaixo segue uma breve lista de situações citadas por US DoT (2017) e Dowling et al. (2016) nas quais um veículo conectado pode ter vantagens em relação a um veículo autônomo:

a) em cruzamentos de vias, veículos conectados podem trocar informações sobre a chegada no cruzamento e otimizar as rotas para que nenhum veículo tenho que parar, assim como demonstra a Figura 3;

Figura 3 - Veículos conectados trocando informações para cruzamento



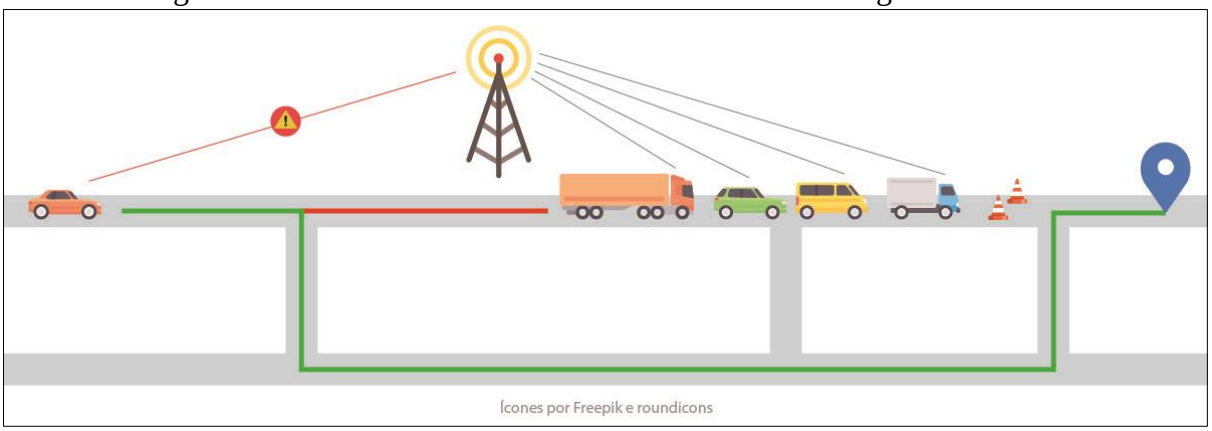
Fonte: elaborado pelo autor.

b) em caso de um veículo de emergência precisar da via disponível para acelerar a sua chegada, ele pode comunicar os demais veículos para desviar da rota ou liberar espaço antes mesmo da aproximação do veículo de emergência;

c) veículos conectados podem verificar rotas congestionadas ou interditadas por meio de uma comunicação com a internet e evitá-las. Essa situação é apresentada na

Figura 4;

Figura 4 - Veículo conectado alterando rota devido a congestionamento



Fonte: elaborado pelo autor.

d) comunicando previamente o momento da frenagem a outro veículo, é possível que este então calcule a frenagem para minimizar a desaceleração e oferecer mais conforto aos passageiros.

Cada situação necessita de diferentes ferramentas para que possa ser executada. Enquanto casos como o recálculo de rota (item c) podem hoje ser executados por aplicativos em smartphones com acesso à internet, situações que são críticas em relação à segurança exigem também funcionalidades integradas ao veículo.

2.2.1 Simulação de ambiente com veículos conectados

A necessidade de simular veículos já existe faz algum tempo, há registros de projetos como o de Barceló et al. (1999) já no século passado. Essa necessidade surgiu com o grande aumento na quantidade de veículos e a limitação das vias, exigindo que um modelo pudesse identificar pontos críticos.

Fatores imprevisíveis como o tempo, infraestrutura presente na região e incidentes afetando as vias devem ser levados em consideração nas simulações, pois mudam o comportamento dos motoristas e alteram a incidência de diferentes resultados. Porém, esses fatores não seguem uma sequência lógica, o que reforça a afirmação de Hertkorn et al. (2002, p. 1, tradução nossa) de que “é impossível descrever tráfego com uso de fórmulas matemáticas”.

Além da complexidade, é importante destacar algumas características que podem variar dependendo do modelo adotado para a simulação. De acordo com Hertkorn et al. (2002, tradução nossa), algumas destas características podem ser destacadas de acordo com alguns critérios:

a) espaço: o ambiente pode ter um espaço discreto ou contínuo, que influencia a precisão da posição de seus agentes e objetos;
b) número de modalidades: a quantidade de modalidades a serem simuladas pode variar. Somente automóveis seria um exemplo de apenas uma modalidade, enquanto multimodalidade pode ser exemplificada com a simulação simultânea de trens, automóveis, metrô e outros;
c) escala: pode ser microscópica, simulando cada veículo individualmente, ou macroscópica, abstraindo grupos de agentes para uma simulação mais enxuta.

Para executar essas simulações, são utilizados programas que tem como entrada dados sobre o ambiente e sobre os agentes a serem simulados, e trazem como saída informações ao usuário na forma de registros, tabelas ou interfaces gráficas. Para simular cenários como este, é comum o uso de sistemas multiagentes, que têm sido mais usados na simulação de tráfego pelo potencial de resolver problemas do mundo real (JIN et al., 2012). Estes programas podem ter um foco específico, como simular um semáforo ou cruzamento, ou um âmbito mais abrangente, chegando a simulação de cidades inteiras.

2.3 SISTEMAS MULTIAGENTE

Segundo Doniec et al. (2008, p. 1443, tradução nossa), "simulação consiste na reprodução do comportamento dinâmico de fenômenos reais utilizando modelos". A simulação em um sistema multiagente busca aumentar as possibilidades transpassando limitações de outros paradigmas de programação. "Sistemas multiagente podem ajudar a simular fenômenos complexos e dinâmicos [...] que são difíceis de expressar utilizando formalismos matemáticos" (DONIEC et al., 2008, p. 1443, tradução nossa).

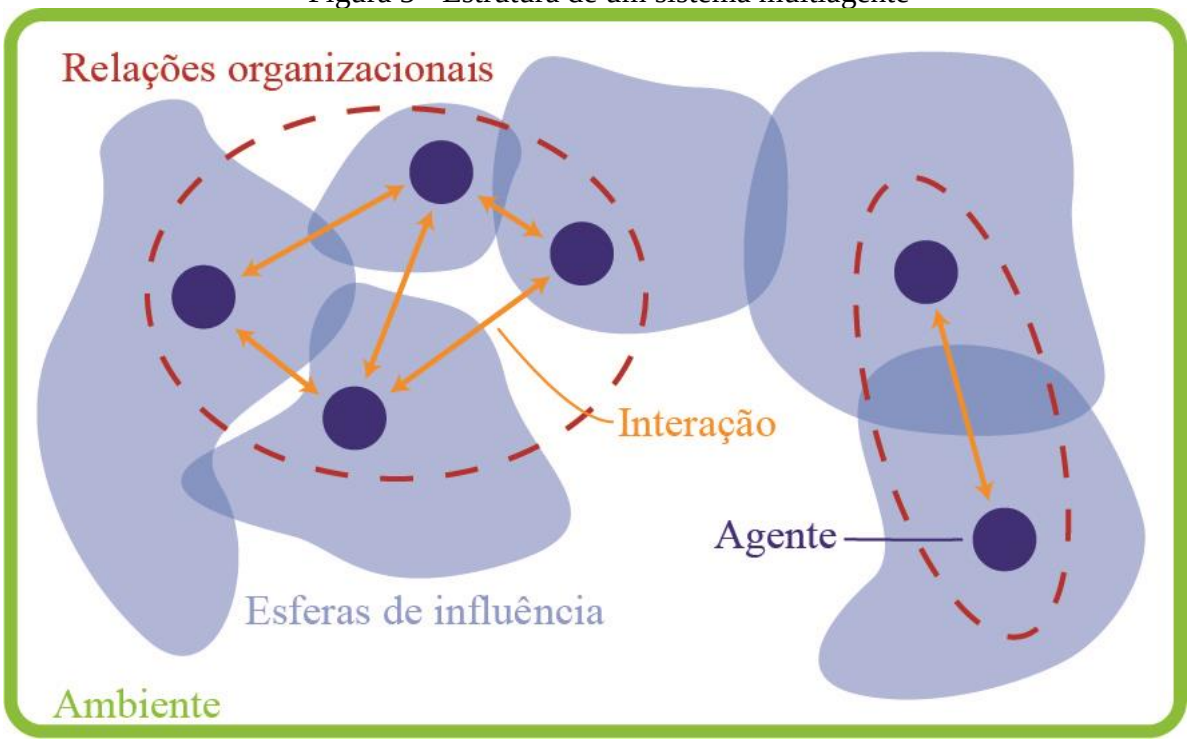
Segundo Silva (2005), as principais características de um agente são comunicação, autonomia, monitoramento, atuação e inteligência. Destaca-se em um Sistema Multiagente (SMA) a importância da comunicação, que é necessária para tornar o ambiente o mais conhecido possível e influencia diretamente nas ações dos agentes, que negociam ou cooperam com objetivo de solucionar um problema comum aos agentes.

Assim como descrito por Bordini, Hübner e Wooldridge (2007), diferente de outros paradigmas, os sistemas reativos se diferenciam dos demais por manterem uma interação de longo prazo com o ambiente em que estão. Por causa dessas características, sistemas com características reativas “são muito mais difíceis de serem corretamente e eficientemente projetados” (BORDINI; HÜBNER; WOOLDRIDGE, 2007, p. 2, tradução nossa). Mais do

que apenas características reativas, os agentes ainda se destacam por exibirem certa autonomia no ambiente, com objetivos e instruções de como alcançá-los.

Segundo Hübner (1995), o estudo de SMA se dá no comportamento inteligente de uma sociedade de agentes autônomos, sendo que os agentes podem solucionar mais de um problema, podem entrar e sair da sociedade, e sua organização interna pode sofrer alterações. Além disso, Bordini, Hübner e Wooldridge, (2007) descrevem algumas das principais propriedades de um agente como sendo autonomia, proatividade, reatividade e habilidade social. A Figura 5 demonstra a estrutura básica de um sistema multiagente.

Figura 5 - Estrutura de um sistema multiagente



Fonte: baseado em Bordini, Hübner e Wooldridge, (2007, p. 6).

Com a intenção de criar um padrão de desenvolvimento para agentes e sistemas multiagente, foi criada em 1996 a Foundation for Intelligent Physical Agents (FIPA), órgão que em 2005 foi absorvido pela IEEE e que contribuiu para a criação de normas e especificações no desenvolvimento dessa área (IEEE, 2017). Dos seus feitos, um dos de maior destaque foi a criação do FIPA-ACL, um padrão de comunicação bastante aprovado pela comunidade no seu lançamento.

2.3.1 Comunicação entre agentes

A habilidade social citada no parágrafo anterior indica como os agentes vão se comunicar, trocar informações e, consequentemente, agir. Afinal, como diz Horling e Lesser (2005, p. 281, tradução nossa), “todo sistema multiagente [...] tem alguma forma de

organização, mesmo que implícita ou informal". Esta comunicação pode se dar por uma característica social dos agentes no ambiente, que pode ser de natureza competitiva, em que os agentes disputam para que seus objetivos (ou de seu grupo) sejam atingidos independente dos demais. Ou de natureza cooperativa, quando os agentes necessitam da troca de informações para maximizar o conhecimento entre eles e, consequentemente, melhorar a sua performance conjunta.

Para exercer essa comunicação é importante que os agentes tenham uma comum linguagem para a troca de informações, garantindo que ambas as partes tenham suas mensagens corretamente interpretadas. Para resolver este problema, nos anos 90 foi criado o Knowledge Query and Manipulation Language (KQML), que criou padrões para a troca de mensagens entre agentes. Esse padrão foi logo substituído pelo FIPA-ACL, linguagem criada pela FIPA baseada em outras especificações existentes na data.

Assim como um padrão de mensagens, é necessário um padrão para a troca destas. Baker e Chauhan (1998) citam alguns dos principais em SMA:

a) comunicação direta: refere-se à forma de comunicação em que os agentes tem conhecimento do outro agente, enviando mensagens diretamente até o mesmo em um canal ponto-a-ponto;
b) comunicação assistida: é definida pela comunicação em SMA em que os agentes utilizam um agente facilitador para a transmissão das mensagens, como num sistema centralizado. Usada frequentemente quando o número de agentes no ambiente é alto;
c) difusão: acontece quando um agente envia uma mensagem a todos os agentes do ambiente. Bastante útil quando os agentes não conhecem os demais;
d) quadro-negro: refere-se ao modelo em que os agentes utilizam uma mesma base de informações compartilhada, a qual todos acessam e nela escrevem.

2.3.2 Ambiente multiagente

Assim como os agentes, o ambiente em que ele se encontra também pode ser caracterizado e "afeta diretamente no design apropriado para o programa do agente" (RUSSEL; NORVIG, 2003, p. 38, tradução nossa). Russel e Norvig (2003) descrevem algumas das propriedades que permitem classificar os ambientes simulados:

a) visibilidade: descreve a capacidade dos sensores de observar o ambiente. Podem ser parcialmente observáveis ou totalmente observáveis, sendo o primeiro a situação deste projeto;

b) determinismo: identifica se o estado do ambiente é determinado somente pela atuação do seu atual estado e as ações do agente ou se existe algum grau de entropia envolvido. Pode ser classificado em determinístico e estocástico. Em um caso de real de trânsito o ambiente é considerado estocástico, mas como o objetivo deste projeto é de simular os agentes somente, será desenvolvido em um ambiente determinístico;
c) dinamismo: é considerado dinâmico um ambiente que sofre alterações enquanto o agente determina suas ações, necessitando que este observe o ambiente em tempo real. Este projeto visa simular um ambiente dinâmico;
d) tempo e espaço: podem ser considerados discretos ou contínuos, dependendo da sua precisão e intervalos determinados. Para maior precisão na simulação, serão utilizados intervalos de tempo e espaço contínuos;
e) número de agentes: pode ter somente um agente, ou vários, sendo então considerado multiagente. Para a simulação de vários veículos será desenvolvido um ambiente multiagente.

Conforme Panait e Luke (2005), SMA se mostram muito importantes em ambientes que possuem algum tipo de restrição, de forma que não funcionem como um único agente, nem como agentes totalmente independentes. Isso se mostra válido para o caso dos veículos conectados, em que, apesar da troca de informações para realização de tarefas em conjunto, como travessia de cruzamentos, os agentes têm suas tarefas independentes de um mestre.

Ao desenvolver um agente, Russel e Norvig (2003) citam quatro principais características que devem ser levadas em consideração para uma correta modelagem: indicadores de performance, ambiente, sensores e atuadores. Essas características são usadas diretamente nos programas que os agentes executam. O Quadro 1 descreve o ciclo de execução de um agente baseado no livro de Russel e Norvig (2003), destacando as características previamente citadas.

Quadro 1 - Ciclo de execução de um agente

função de execução de um agente
    verificar **ambiente** com os **sensores**
    adicionar ao conhecimento do agente dados recebidos dos **sensores**
    procurar entre as possíveis ações a que tem **melhor performance**, baseado no conhecimento do agente
    executar ação

Fonte: adaptado de Russel e Norvig (2003, p. 45).

O ciclo descrito no Quadro 1 é um dos tipos de agentes mais simples, mas destaca os fatores principais do programa. Existem agentes que agem diretamente por reflexo de

sensores, outros orientados a objetivo ou a utilidade e até agentes que verificam a consequência das suas ações para melhorá-las. Com as diversas opções, não existe uma que é errada, pois, segundo Russel e Norvig (2003, p. 54, tradução nossa), “o design apropriado de um programa para um agente depende da natureza do ambiente”.

Outra característica importante ao ambiente se dá em como os seus agentes são organizados. Foi citada na seção 2.3 a característica autônoma dos agentes, porém, quando estes são organizados em grupos, precisam de uma coordenação conjunta para uma melhor execução de seus objetivos. Esta coordenação pode vir de um agente, que, para poder coordenar os demais de forma a maximizar a performance, precisa ter o máximo de conhecimento possível. Além disso, existem opções de coordenação distribuída, onde os agentes negociam a melhor opção e a executam, como apresentado por Olfati-Saber, Fax e Murray (2007).

2.3.3 Simulação em Tempo Real

Bergero e Kofman (2010, p. 125, tradução nossa) descrevem sistemas em tempo real como “sistemas em que há uma limitação sujeita ao tempo real [...] e respostas a um estímulo têm um prazo a ser cumprido, independente da carga do sistema”. Em outras palavras, Bélanger, Venne e Paquin (2010) descrevem simulação em tempo real como aquela em que o tempo de execução é aceitavelmente similar ao do mundo real. Sendo assim, este tipo de simulação se difere de outros em que o tempo de cada ciclo de execução é independente, limitado somente à capacidade de processamento da máquina em que executa.

Para que aconteça a sincronia dos eventos, simulações em tempo real tem como requisito que um ciclo de suas ações execute em até um tempo máximo determinado, quando um ciclo não é concluído dentro deste limite temos um *overrun*. Segundo Bergero e Kofman (2010, p. 125, tradução nossa), a forma de lidar com esta irregularidade pode classificar o sistema em:

a) *hard real-time system* (sistema em tempo real rígido): no qual um *overrun* é considerado ineficaz, podendo causar erros críticos no sistema;
b) *soft real-time system* (sistema em tempo real suave): no qual a irregularidade é aceitável, podendo ser ignorada ou corrigida.

Por muito tempo este tipo de simulação foi pouco usado pelo seu alto custo computacional necessário para garantir a ausência de *overruns*. Mas devido à recente “queda no custo e crescimento em performance, [...] a capacidade de resolver problemas mais

complicados em menos tempo tem melhorado" (BERGERO; KOFMAN, 2010, p. 125, tradução nossa), fazendo com que essa tecnologia possa ser usada de forma mais abrangente.

A utilização de sistemas em tempo real apresenta diversas vantagens principalmente para simulações, pois quando trata o intervalo de tempo de forma semelhante ao mundo real facilita a visualização e interpretação do ambiente. Entender o comportamento de veículos na simulação, visualizando-os de maneira familiar, pode muito ajudar na compreensão e análise do contexto que se estuda com a simulação.

2.4 TRABALHOS CORRELATOS

Nesta seção serão abordados os trabalhos correlatos. A seção 2.4.1 apresenta um artigo escrito por Jin et al. (2012) e contempla um projeto de simulação de um sistema multiagente de veículos conectados para controlar cruzamentos de vias. Na seção 2.4.2 é descrito um simulador desenvolvido por Hertkorn et al. (2002) com foco na mobilidade urbana, que tem como unidade mínima uma única pessoa e é capaz de considerar uma cidade por completo. O trabalho de Freire (2004), descrito na seção 2.4.3, é um simulador de veículos para identificar ruas com maior risco de congestionamento, e teve sua extensão com foco na melhoria gráfica por Ranghetti (2007). O trabalho de Cartolano et al. (2008) é brevemente descrito na seção 2.4.4, que busca precisão com a simulação de um ambiente de veículos autônomos juntamente com a comunicação sem fio presente neste ambiente.

2.4.1 Advanced Intersection Management for Connected Vehicles Using a Multi-Agent Systems Approach

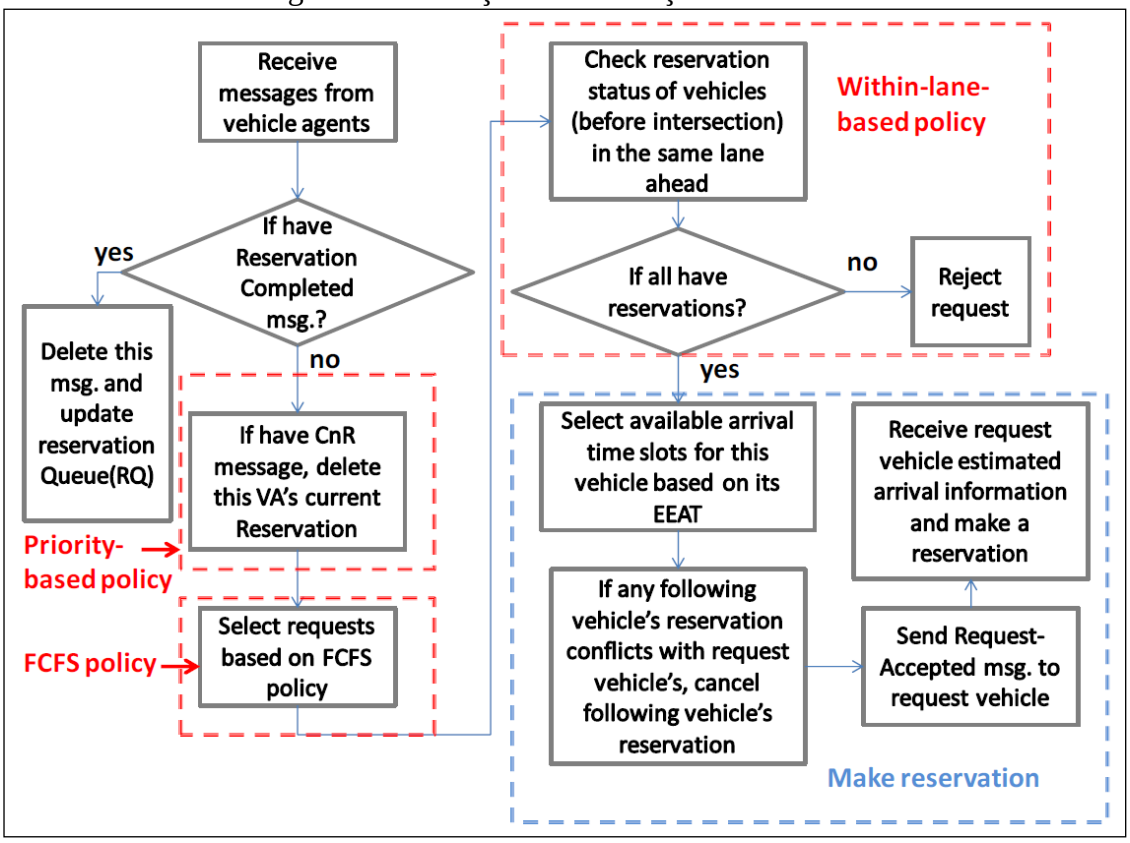
O trabalho desenvolvido por Jin et al. (2012) teve como objetivo resolver o problema de interseção de vias automotivas considerando um ambiente multiagente simulado composto por veículos conectados. O principal ponto de destaque deste trabalho é a remoção dos semáforos, utilizando somente comunicação entre os agentes (veículos e um agente de controle de cruzamento). Para tal, foi desenvolvido um Sistema de Gerenciamento de Tráfego Avançado (Advanced Traffic Management System - ATMS).

Este sistema recebe dos veículos informações sobre a previsão de chegada no cruzamento, reserva um período de tempo para a travessia e provê respostas sobre a situação da reserva, que pode ser aceita para o momento solicitado, após o momento solicitado ou rejeitada. A Figura 6 apresenta um infográfico do processo do ATMS para receber uma solicitação, identificar a viabilidade e responder o solicitante com o tempo autorizado ou a negação da solicitação. É importante ressaltar que as simulações foram feitas considerando

duas vias de mão única que se cruzam, onde apenas um veículo atravessasse o cruzamento de cada vez.

Comparado com um cruzamento que apresenta um semáforo comum, o ATMS chegou a ter um ganho de 87% em simulações de congestionamento, reduzindo não somente o tempo de viagem assim como a emissão de gases poluentes. Portanto, a simulação provou que em um ambiente real um ATMS pode trazer grandes benefícios ao usuário e ao meio-ambiente.

Figura 6 - Aceitação de solicitação de travessia



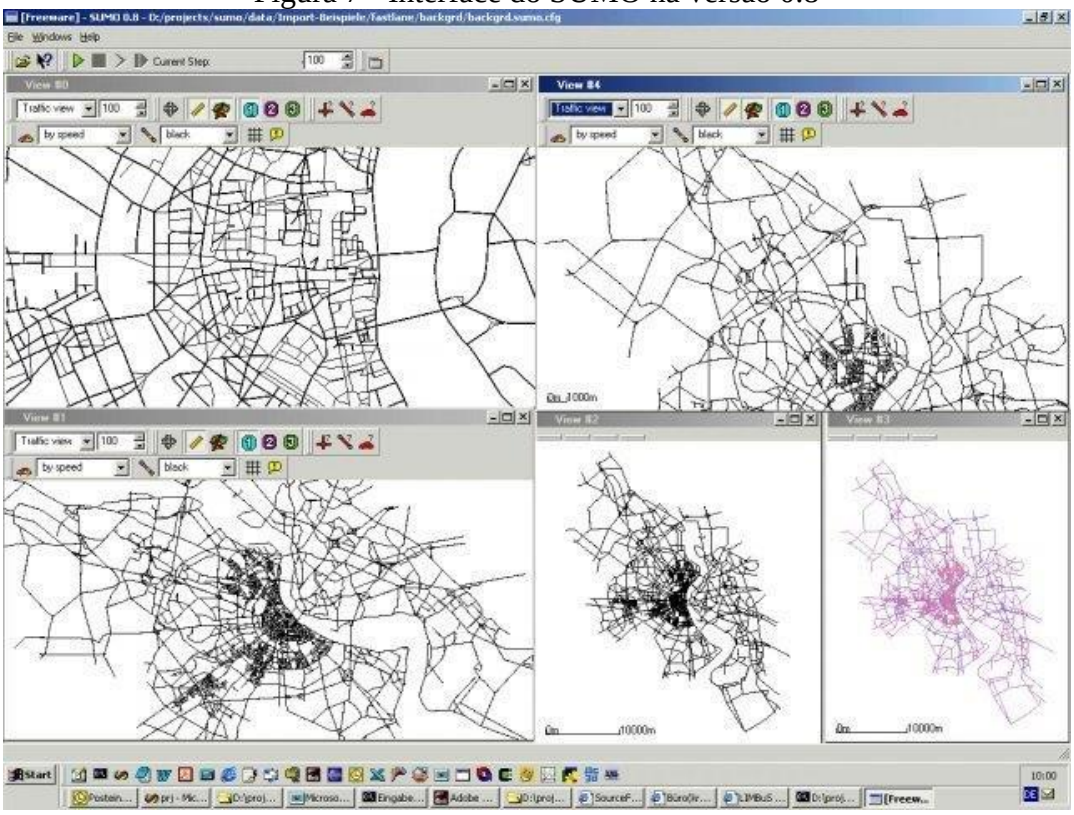
Fonte: Jin et al. (2012).

2.4.2 SUMO – Simulation of Urban Mobility: an Open Source Traffic Simulation

Este simulador de código aberto foi desenvolvido inicialmente por Hertkorn et al. (2002) e tem como principal objetivo simular a mobilidade de uma cidade por completo, considerando não somente o trânsito de automóveis, como também transporte coletivo. O projeto tem como menor unidade uma pessoa, e simula o transporte da mesma entre dois pontos da cidade, considerando sub-rotas que podem ser compostas por diferentes formas de transporte. Cada veículo tem seu tipo, local e velocidade independentes, e a simulação ocorre sem colisões, utilizando em seu ambiente espaço contínuo, porém tempo discreto, com intervalos de 1 segundo. Outro ponto de destaque deste projeto é a simulação de mais pistas em uma via, área citada como extensão em diversos projetos semelhantes.

O projeto é desenvolvido em Python e C++ e ganhou atenção em diversos outros trabalhos na área pela sua complexidade, contendo simulação considerando indivíduos, interface gráfica completa, possibilidade de importar mapas de outras ferramentas conhecidas no mercado, simulação de intervalos de tempo entre semáforos e outras funcionalidades. O relatório Sumo (2016) apresenta diversas pesquisas relacionadas ao projeto, com aplicações, casos de uso e desenvolvimentos. A Figura 7 apresenta uma imagem da interface do sistema na sua versão 0.8, em uma simulação para a cidade de Cologne, Alemanha.

Figura 7 - Interface do SUMO na versão 0.8



Fonte: Sumo (2011).

O projeto obteve sucesso, concluindo os objetivos inicialmente estabelecidos de desenvolver uma plataforma modular e multiagente para a simulação de tráfego de veículos. Além disso, o projeto tem seu desenvolvimento estendido há mais de 10 anos, sendo sua atualização mais recente em maio de 2017 para a versão 0.30.0 (SUMO, 2017).

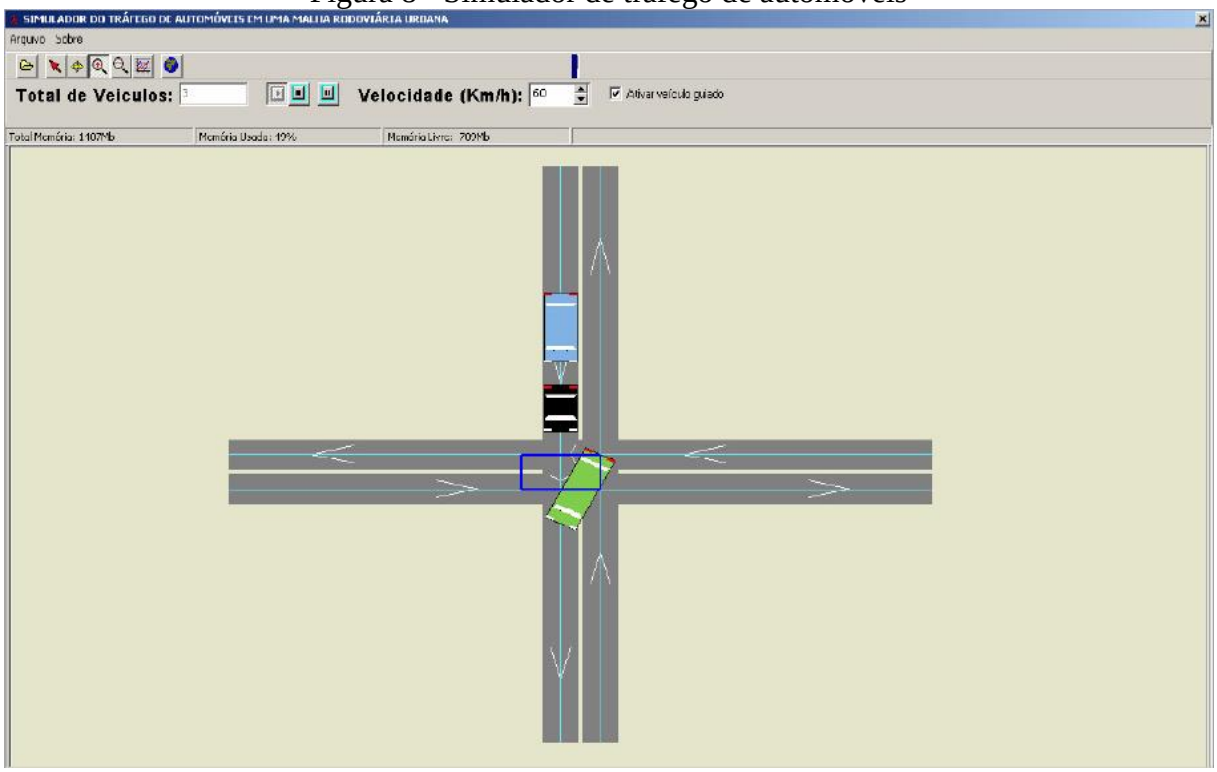
2.4.3 Simulador de Tráfego de Automóveis em uma Malha Rodoviária

O objetivo do trabalho de Freire (2004), estendido por Ranghetti (2007), é simular e verificar o comportamento do tráfego de veículos em uma malha rodoviária. Uma das principais motivações do projeto é “verificar a validade de projeto de vias” (FREIRE, 2004, p.13), podendo-se executar uma simulação antes de haver investimento modificando o trânsito no local.

Para simular diversos veículos independentes, foram utilizados processos concorrentes dentro da linguagem Object-Pascal, do ambiente Delphi. Para a interface gráfica, foi utilizado a biblioteca OpenGL. A interface permite ao usuário selecionar uma malha rodoviária e definir o limite de velocidade e a quantidade de veículos de cada via, criando também semáforos em nodos com mais de uma entrada. Na execução, o tamanho dos veículos e suas rotas são aleatórios.

Segundo Ranghetti (2007), o projeto conseguiu simular casos reais de engarrafamentos, provando que pode ser de grande valor para planejamento de vias, se destacando pela possibilidade de visualização em 3D e opção de guiar um veículo. Entretanto, o sistema deixa bastante espaço para expansão e melhorias de usabilidade para o usuário. A Figura 8 apresenta uma interface do simulador em que o usuário está com foco em um cruzamento.

Figura 8 - Simulador de tráfego de automóveis



Fonte: Ranghetti (2007, p. 55).

2.4.4 iTETRIS: An Integrated Wireless and Traffic Platform for Real-Time Road Traffic Management Solutions

O trabalho de Cartolano et al. (2008), descrito mais recentemente por Bauza et al. (2013), visa preencher uma necessidade na simulação de veículos conectados em grande escala. Entre os principais objetivos do sistema, tem-se a necessidade de que seja uma plataforma global, aberta, precisa e de longa durabilidade.

Partindo de diversos padrões europeus para comunicação entre veículos, o trabalho desenvolve um simulador altamente modular e flexível, utilizando o SUMO (trabalho citado na seção 2.4.2) para simulação dos veículos e o ns-3 (simulador de comunicação sem fio) para simular a comunicação sem fio entre veículos (V2V) e entre veículos e infraestrutura das vias (V2R), ligados por um módulo central, chamado de iTETRIS Control System (iCS). Para iniciar a simulação, iCS envia comandos ao SUMO e ao ns-3, indicando arquivos de configuração para inicialização. Para garantir que os dois programas estejam corretamente sincronizados, o iTETRIS possui um módulo de gerenciamento de sincronização, pois o tempo de ciclo de execução dos programas é diferente. Para garantir eficiência da simulação em grande escala, os módulos devem rodar simultaneamente na mesma máquina.

Segundo Cartolano et al. (2008), um simulador com as características do iTETRIS deve suprir uma necessidade presente no mercado europeu. Para seu progresso, recebeu suporte da Comissão Europeia e tem parceiros como Thales Group, Centro Aerospacial Alemão (DLR) e Eurecom.

3 DESENVOLVIMENTO

A seguir são descritos os requisitos, a especificação e a implementação do sistema. Em cada tópico são apresentadas brevemente características do trabalho, possuindo ao fim um item dedicado aos resultados obtidos.

3.1 REQUISITOS

Os requisitos para o desenvolvimento deste trabalho são:

a) o sistema deve permitir que diferentes ambientes sejam configurados e/ou selecionados (Requisito Funcional – RF);
b) o sistema deve permitir a inclusão de veículos no ambiente (RF);
c) o sistema deve exibir um resumo dos dados do ambiente simulado (RF);
d) o sistema deve possuir uma interface gráfica que permite o acompanhamento de um veículo selecionado pelo usuário, apresentando dados em tempo real sobre o mesmo (RF);
e) o sistema deve permitir a exportação de um arquivo com dados detalhados sobre a simulação dos agentes (RF);
f) o sistema utilizará uma arquitetura multiagentes (Requisito Não-Funcional – RNF);
g) os veículos simulados devem comunicar-se entre si utilizando a biblioteca Boris, trocando informações relevantes para o cumprimento de seus objetivos como agentes (RNF);
h) a simulação deve ser independente de uma interface para visualização (RNF);
i) o sistema será desenvolvido para a plataforma Windows (RNF);
j) o sistema será desenvolvido em C#, na IDE Visual Studio (RNF).

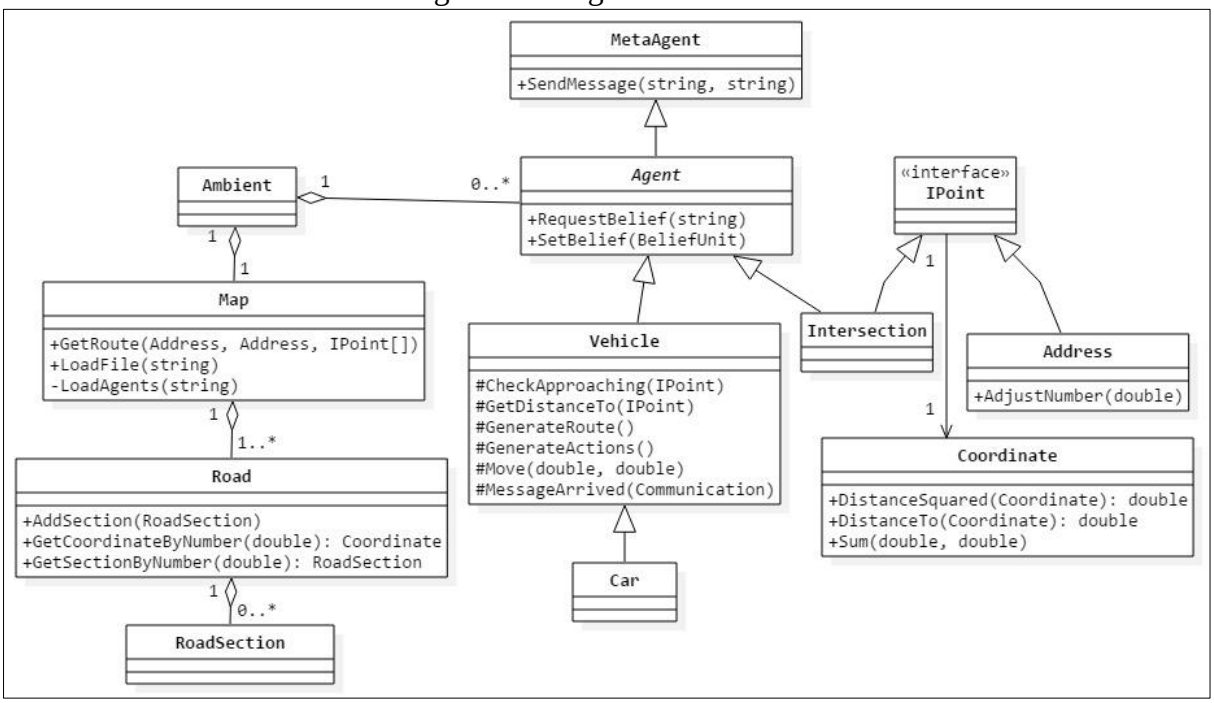
3.2 ESPECIFICAÇÃO

A seguir encontram-se alguns diagramas relacionados ao projeto que facilitam no entendimento do sistema desenvolvido. Os diagramas apresentados foram criados utilizando a ferramenta StarUML v2.8.

3.2.1 Diagrama de classes

A Figura 9 apresenta o diagrama de classes do sistema que simula os agentes. O diagrama apresenta as principais classes do sistema relacionadas à simulação multiagente. A partir do diagrama podemos identificar alguns pontos de destaque, descritos na sequência.

Figura 9 - Diagrama de classes



Fonte: elaborado pelo autor.

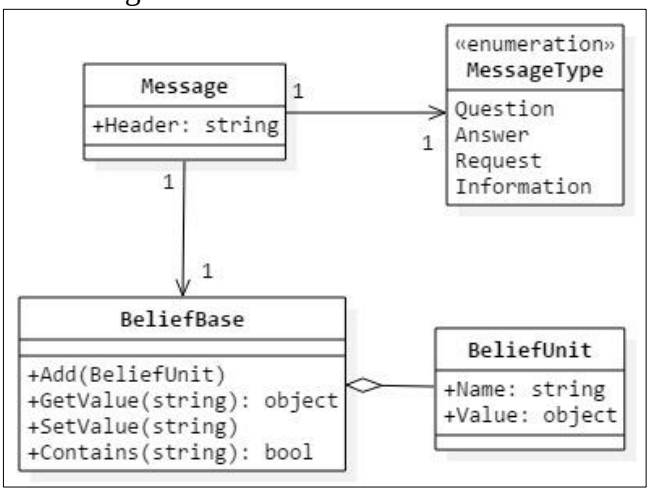
A classe `Ambient` possui atributos e relações que a deixam como centro da simulação. Ela é a classe que tem o controle central do ambiente simulado, tanto dos agentes como das características gerais. O mapa e todos os agentes envolvidos estão diretamente conectados à classe.

Todas as classes que representam algum ponto no mapa implementam a interface `IPoint`, que tem como atributo uma coordenada. Dessa forma, tanto endereço (`Address`) como interseções (`Intersection`) tem sempre um ponto de referência no mapa.

Todas as classes que exercem algum tipo de comunicação no ambiente derivam da classe `Agent`, que por sua vez herda da classe `MetaAgent` da biblioteca Boris.NET. A classe `MetaAgent` registra os agentes em um portal com um ID único e garante que todas as classes possuam um método para enviar mensagens e uma atributo para interpretar mensagens recebidas. A classe abstrata `Agent` adiciona algumas características próprias do sistema ao agente, como uma base de conhecimento (`BeliefBase`). A classe `Vehicle` define métodos e atributos base para que uma classe que representa um veículo, sendo `Car` uma classe com características mais específicas.

Para executar a comunicação entre os agentes, foi utilizado um conjunto de classes descritas na Figura 10. A classe `Message` é a principal, que armazena o título, o tipo e o conteúdo da mensagem. Cada variável a ser passada é adicionada na classe `BeliefBase` como par nome-valor em uma classe `BeliefUnit`. Ao transmitir a mensagem, a classe `Message` é serializada em formato XML pelo sistema.

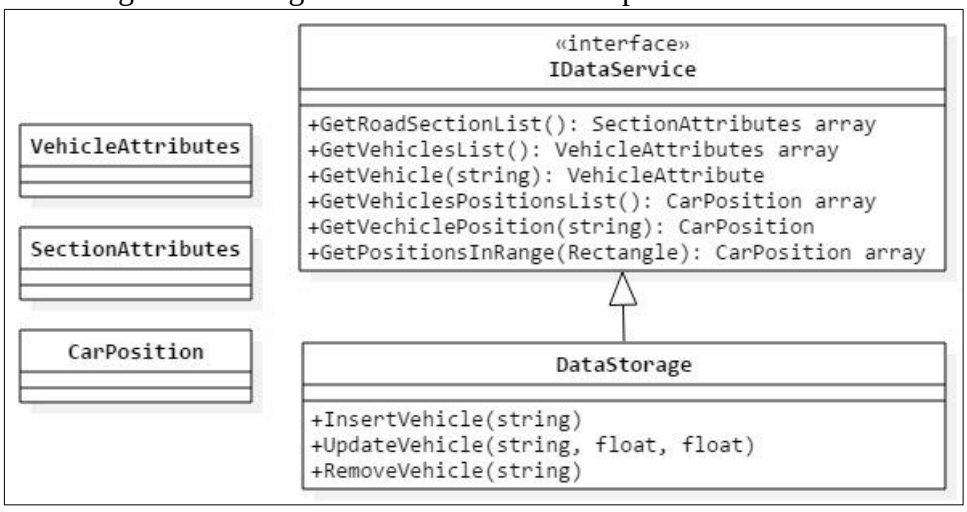
Figura 10 - Diagrama das classes relacionadas à comunicação



Fonte: elaborado pelo autor.

Para fazer a troca de informações entre a simulação e a interface gráfica é utilizado um serviço de Web Service. Para que o serviço funcione por meio do Windows Communication Foundation (WCF), são necessárias uma interface e uma classe que a implemente, no caso deste sistema, `IDataService` e `DataStorage`, respectivamente. Além disso, foram criadas classes que armazenam informações para facilitar a troca de mensagens com demais sistemas, estas classes todas são serializáveis. As classes utilizadas pelo serviço são apresentadas na Figura 11.

Figura 11 - Diagrama das classes criadas para o Web Service

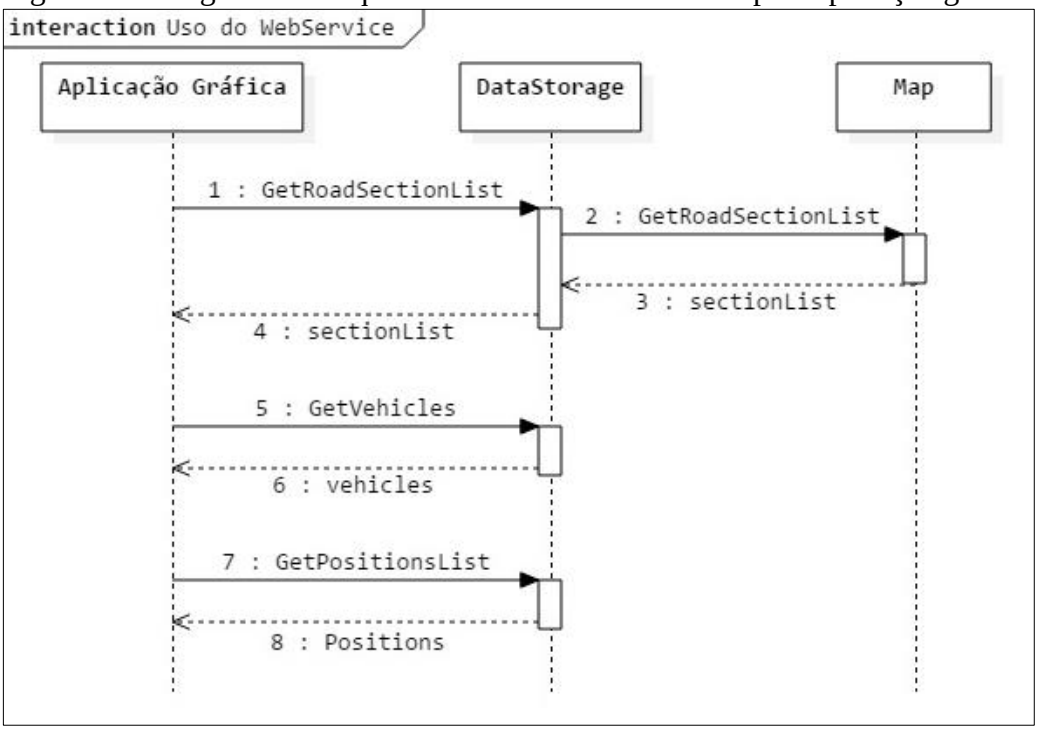


Fonte: elaborado pelo autor.

3.2.2 Diagrama de sequência da aplicação gráfica

Para buscar os dados necessários para visualizar a simulação, a aplicação gráfica utiliza um serviço de WebService mantido pela simulação. O processo de busca de informações é apresentado na Figura 12. O fluxo mostra a solicitação de informações pela aplicação gráfica à simulação, com informações sobre o ambiente, os agentes e suas posições.

Figura 12 - Diagrama de sequência de uso de WebService pela aplicação gráfica



Fonte: elaborado pelo autor.

A partir do diagrama de sequência, é importante destacar que a aplicação gráfica é independente da simulação, rodando em um processo separado, possivelmente em outra máquina, se necessário. Dentre as principais características do diagrama, é importante destacar que as ações descritas nos itens 1, 5 e 7 são chamadas por meio de Web Service utilizando-se do WCF. Além disso, a ordem em que são apresentadas as chamadas acontece na inicialização do programa, tendo a última ação (`GetPositionsList`) sendo repetida para atualizar as posições dos veículos na aplicação gráfica.

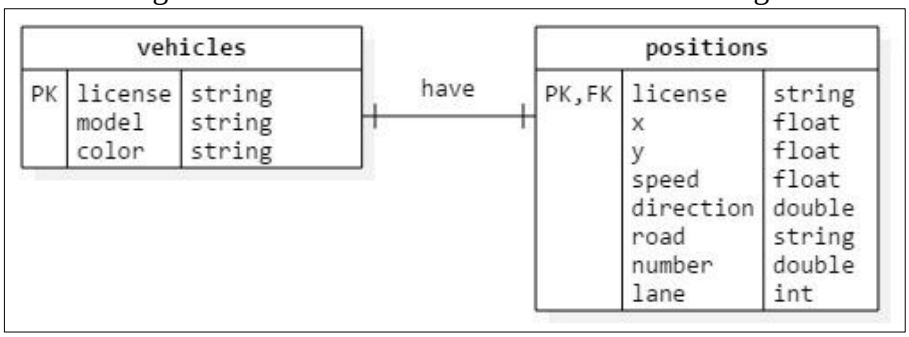
Outro aspecto de destaque no diagrama de sequência é que, diferente da chamada do item 1, que busca informações em outras classes. As chamadas dos itens 5 e 7 retornam valores presentes na própria classe `DataStorage`, por meio do armazenamento de dados que acontece utilizando-se duas tabelas, que são atualizadas pelos veículos ao movimentarem-se.

3.2.3 MER da base de armazenamento disponibilizada no Web Service

Para armazenar as informações dos veículos de forma a ter ágil acesso pelo Web Service, optou-se por criar tabelas na classe que implementa o serviço. Essas tabelas fazem parte de um `DataSet`, classe da plataforma .NET que cria um banco de dados não persistente que pode ser acessado por programação de forma similar a outras classes. O Modelo Entidade-Relacionamento (MER) das tabelas encontra-se na Figura 13. São apenas duas

tabelas: uma para a listagem dos veículos e outra para as suas posições e características da posição atual.

Figura 13 - MER do DataSet na classe DataStorage



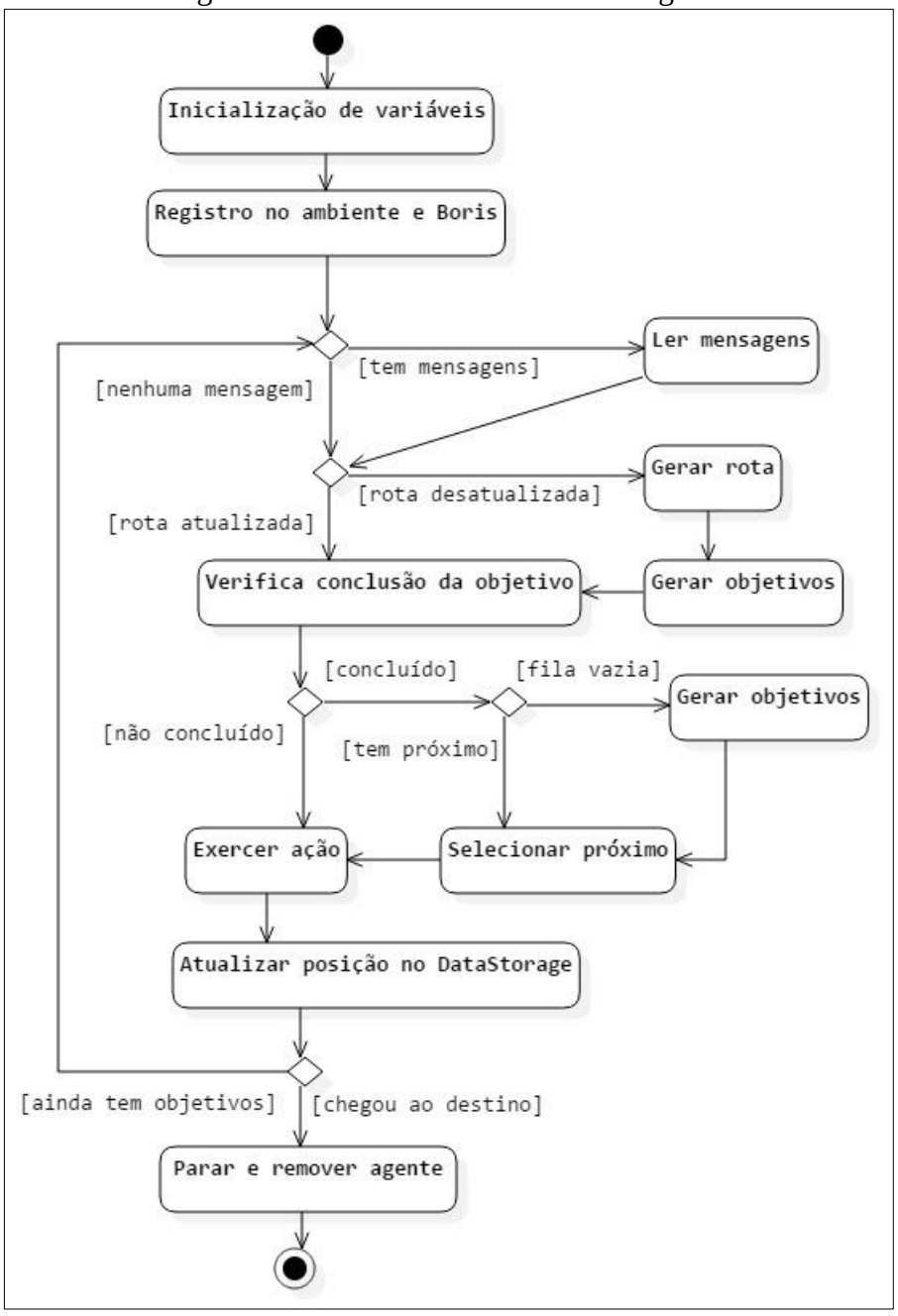
Fonte: elaborado pelo autor.

3.2.4 Diagrama do ciclo de atividade de um agente

Cada ciclo exercido por um agente em seu processo executa diversos comandos para garantir que este execute as ações necessárias no tempo certo. Os principais itens deste ciclo estão apresentados em um diagrama de atividade na Figura 14. O agente é primeiramente iniciado e registrado no ambiente. Em cada ciclo, ele verifica por mensagens, atualiza a rota caso o destino tenha sido alterado e verifica a conclusão do seu atual objetivo. Finalmente, ele exerce sua ação e atualiza sua posição na classe `DataStorage`. O ciclo se repete até que o agente não tenha mais destinos ou tarefas a executar.

É importante ressaltar que o veículo possui atributos para armazenar os próximos pontos da rota e os objetivos, que envolvem acelerar, frear e manter a velocidade. As listas são preenchidas quando estão vazias e ainda existem informações para preenchê-las novamente.

Figura 14 - Ciclo de atividade de um agente



Fonte: elaborado pelo autor.

3.3 IMPLEMENTAÇÃO

A seguir são descritas as etapas do desenvolvimento do trabalho, juntamente com uma descrição do seu uso.

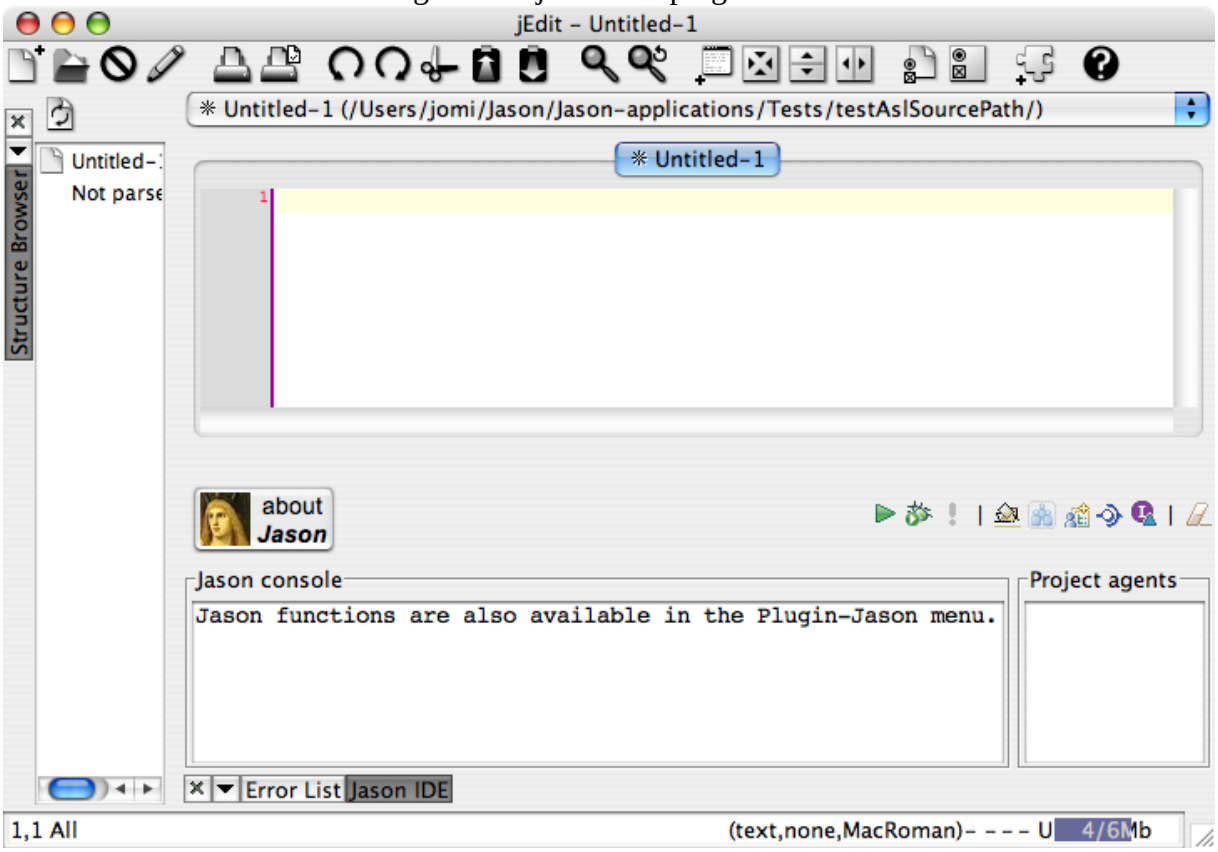
3.3.1 Pesquisa inicial e avaliação de softwares

Como primeira etapa do trabalho, foi necessário fazer uma pesquisa a respeito de sistemas multiagente, para tomar maior conhecimento do assunto e de como se dá o desenvolvimento do mesmo. Nesta etapa inicial do trabalho foi estudado o funcionamento de

sistemas de simulação multiagente e opções de softwares e Kits de Desenvolvimento de Software (SDK) relacionados que poderiam ser utilizados para o desenvolvimento do trabalho.

A primeira ferramenta pesquisada foi o Jason, cuja interface pode ser visualizada na Figura 15 como um *plugin* da ferramenta jEdit. A ferramenta é uma plataforma desenvolvida em Java que contém um interpretador para a linguagem AgentSpeak. As principais vantagens notadas nesta ferramenta são a grande documentação e a possibilidade de integração com uma linguagem bastante popular, o Java. Porém, por utilizar AgentSpeak para a programação principal, que é uma linguagem interpretada, a ferramenta não é otimizada para simulações em grande escala. Além disso, avaliando o funcionamento do programa por meio do livro escrito pelos autores (BORDINI; HÜBNER; WOOLDRIDGE, 2007), foi identificado que a ferramenta tem um ciclo de atualização bastante custoso, com algumas funções imutáveis, fazendo com que a opção fosse descartada.

Figura 15 - jEdit com plugin Jason

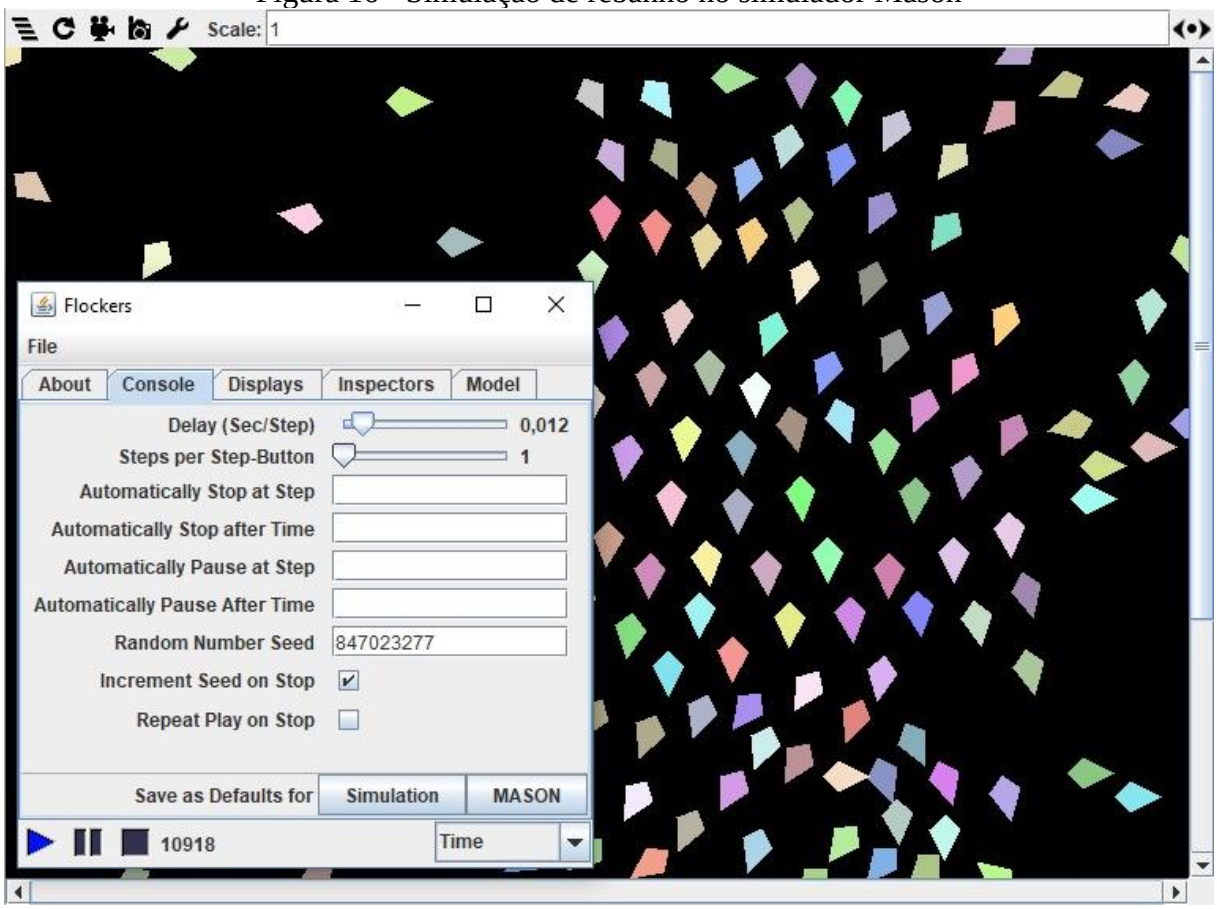


Fonte: Jason (2017).

Buscando avaliar mais opções, foi pesquisado a respeito de outras ferramentas utilizadas no mercado para a simulação multiagente. A ferramenta NetLogo, apesar de possuir exemplos na área de tráfego, notou-se ser limitada em sua programação por utilizar de uma linguagem própria com pouco suporte a orientação a objetos.

Por fim, foi estudada a ferramenta Mason, que apresentou por meio de exemplos ser altamente integrado com Java, tendo assim suporte a diversos paradigmas e bibliotecas nativas da linguagem, além de bibliotecas fornecidas pela ferramenta para visualização da simulação. Entretanto, dos exemplos simulados nesta fase de testes, percebeu-se dificuldade na visualização da simulação por atuar com um ciclo de atualização desconsiderando o tempo real da simulação. Essa característica deixa a ferramenta interessante para simulações em que se busca um resultado final, mas torna difícil analisar a execução com precisão e consistência. Um dos exemplos fornecidos pela ferramenta Mason pode ser encontrado na Figura 16.

Figura 16 - Simulação de rebanho no simulador Mason



Fonte: elaborado pelo autor.

Com o conhecimento obtido avaliando os softwares, foi considerada a opção de desenvolver então uma ferramenta própria contendo as estruturas essenciais observadas nas demais e pudesse rodar em tempo real, facilitando assim sua visualização. Um quadro comparativo citando pontos positivos e negativos das ferramentas, incluindo a opção de uma ferramenta própria, pode ser encontrado no Quadro 2.

Quadro 2 - Quadro comparativo de ferramentas de desenvolvimento

Ferramenta	Vantagens	Desvantagens
Jason	- Várias funções customizáveis - Integração com Java - Extensa documentação	- Utiliza AgentSpeak (interpretado) - Ciclo de atualização custoso - Utiliza turnos em vez de tempo real
Mason	- Bibliotecas para visualização 2D e 3D - Modelos separados da visualização (*Model / View / Controller*) - Grande quantidade de exemplos nativos	- Trabalha com turnos em vez de tempo real - Dificuldade em visualizar a simulação
NetLogo	- Exemplos na área de trânsito - Possibilidade de uso de extensões	- Linguagem de programação própria - Programação estruturada com suporte limitado a POO
Ferramenta própria	- Liberdade no desenvolvimento - Possibilidade de otimização para veículos conectados - Conhecimento prévio da linguagem de programação - Possibilidade de simulação em tempo real	- Necessidade de implementação e testes do sistema multiagente

Fonte: elaborado pelo autor.

3.3.2 Técnicas e ferramentas utilizadas

Uma breve descrição das principais ferramentas utilizadas neste trabalho e seu uso pode ser encontrada a seguir.

3.3.2.1 Visual Studio Community 2017

O Visual Studio 2017 é um Ambiente de Desenvolvimento Integrado (IDE) que na sua versão Community é gratuito para uso. O ambiente suporta diversas linguagens de programação, com destaque para a plataforma .NET da Microsoft, desenvolvedora da ferramenta. Para o sistema, o código foi desenvolvido todo dentro da ferramenta em C# 5.0 para .NET Framework 4.5.2.

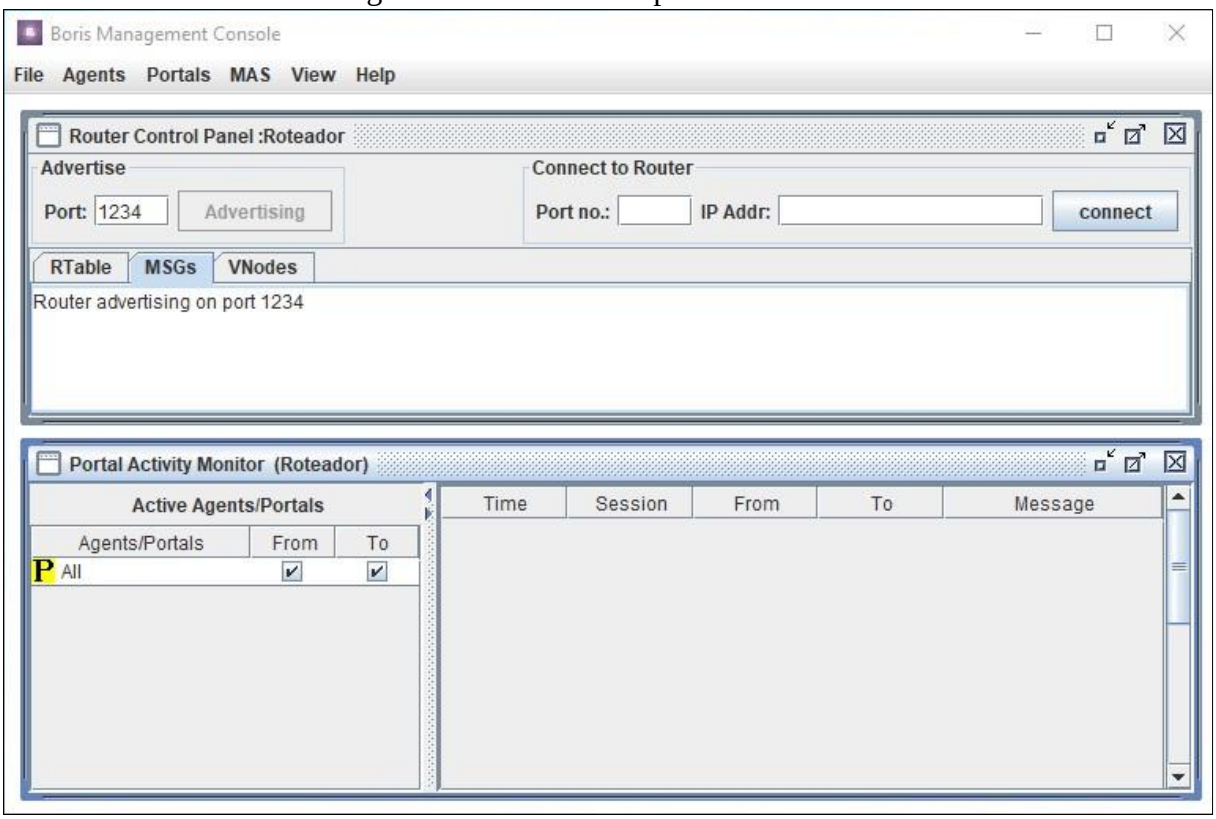
3.3.2.2 Bitbucket

Para repositório online de arquivos e versionamento foi optado pela utilização da ferramenta Bitbucket. Essa opção garantiu uma consistência de arquivos entre diferentes máquinas em que fosse trabalhado e uma cópia dos arquivos no caso de necessidade de restauração, seja por completo ou para uma versão anterior. A integração do Git com o Visual Studio 2017 foi muito útil durante o trabalho, evitando que qualquer outra ferramenta ou código fossem necessários, fazendo a sincronização dos arquivos com facilidade pela interface.

3.3.2.3 Boris.NET

Boris.NET é uma extensão da plataforma Boris, a qual foi criada para facilitar o desenvolvimento de um sistema multiagente com classes e métodos que facilitam a comunicação entre agentes. Essas classes foram integradas aos agentes desenvolvidos no sistema, garantindo a comunicação por meio da plataforma. A interface de um roteador do Boris pode ser visualizada na Figura 17.

Figura 17 - Roteador da plataforma Boris



Fonte: elaborado pelo autor.

3.3.2.4 Json.NET

Json.NET é um *framework* de código aberto para a plataforma .NET. O *framework* tem como objetivo facilitar a utilização de arquivos JSON na plataforma, que nativamente possui suporte apenas para serialização em XML. Neste trabalho sua utilização se deu na leitura dos arquivos de configuração do ambiente na abertura da simulação.

3.3.2.5 Windows Communication Foundation

Windows Communication Foundation é um SDK desenvolvido pela Microsoft com o propósito de permitir a comunicação entre softwares. O SDK é nativo da plataforma .NET desde a sua versão 3.0 e encapsula diversas tecnologias de comunicação, formando um modelo de programação comum. Apesar de ser uma tecnologia proprietária da Microsoft, por

utilizar padrões comuns do mercado (como XML, SOAP e WSDL) o SDK permite que haja uma comunicação heterogênea entre sistemas. Sua contribuição com este trabalho foi na criação de um Web Service para acesso às informações da simulação. Dessa forma a aplicação gráfica poderia ser criada independente da simulação.

3.3.2.6 MonoGame para XNA

Para o desenvolvimento de uma interface gráfica, foi optado pela ferramenta da Microsoft para desenvolvimento de jogos: XNA 4.0. Porém, a plataforma foi descontinuada pela Microsoft em janeiro de 2013, mas é ainda mantida pela comunidade por meio de um projeto chamado MonoGame, que usa a estrutura do XNA para o desenvolvimento para múltiplas plataformas. Sendo assim, foi criada uma janela que exibe a movimentação dos veículos da simulação.

3.3.3 Desenvolvimento do ambiente multiagente

A partir do conhecimento obtido na etapa de pesquisa e levantamento de softwares assim como nos testes executados, foram identificados os principais aspectos de um sistema multiagente. Foram criadas inicialmente provas de conceito para que, em seguida, fosse iniciado o desenvolvimento do sistema. As provas de conceito estão a seguir na seção 3.3.3.1 e o desenvolvimento inicial na sequência, na seção 3.3.3.2.

3.3.3.1 Provas de conceito

Para validar a opção pela alternativa de desenvolvimento próprio, foram desenvolvidos dois testes iniciais. O primeiro foi um teste da ferramenta Boris.NET e a sua capacidade de trocar mensagens entre agentes. Parte do código pode ser encontrado no Quadro 3.

Quadro 3 - Teste com a plataforma Boris.NET

```
1  Portal myPortal = new Portal("London");
2  myPortal.Connect(1234);
3
4  MetaAgent agentA = new MetaAgent("Sam");
5  agentA.MessageReceived += new
          MetaAgent.MessageReceivedHandler(metodo);
6
7  myPortal.AddAgent(agentA);
8  agentA.SendMessage("Dave", "Bom dia, Dave!");
```

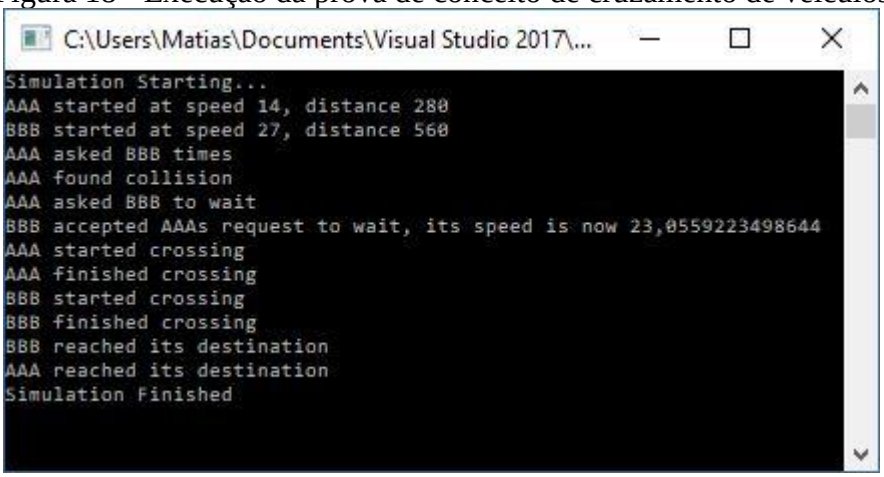
Fonte: elaborado pelo autor.

Para rodar o teste é necessário inicializar o roteador Boris (demonstrado na Figura 17) e identificar a porta que será utilizada. Com essa inicialização é possível instanciar um `Portal` e conectá-lo ao roteador. Na linha 4 do Quadro 3 é criado um agente `MetaAgent` que recebe

como parâmetro obrigatório um identificador único, usado pelo Boris para direcionar as mensagens. O agente então recebe uma função para executar ao receber a mensagem e se conecta ao portal. Após essa configuração, o agente pode enviar mensagens identificando o destinatário e o conteúdo.

O segundo teste foi uma simplificação do sistema a ser criado: dois veículos que se conhecem negociando o momento de travessia em um cruzamento para evitar colisões. Para este teste não foi criada uma interface gráfica e a comunicação foi simplificada à chamada de métodos de um diferente objeto. A execução deste teste pode ser visualizada na Figura 18.

Figura 18 - Execução da prova de conceito de cruzamento de veículos



Fonte: elaborado pelo autor.

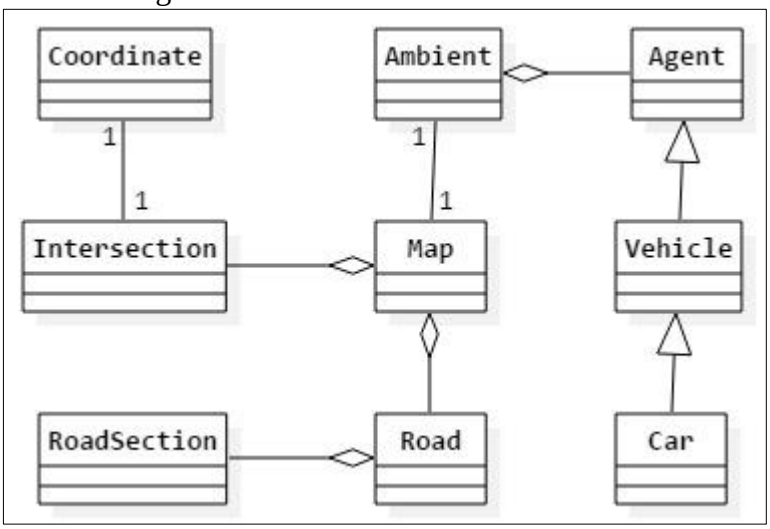
3.3.3.2 Integração de um ambiente multiagente

Com os conceitos provados, foi iniciado o desenvolvimento do sistema multiagente e definidas as primeiras classes a serem desenvolvidas. Essas classes iniciais podem ser identificadas na Figura 19.

Diferente de outras plataformas, que utilizam uma classe para distribuir o tempo da execução entre os agentes, foi optado nesse trabalho por deixar o gerenciamento do tempo com o próprio processador, deixando com que cada agente fosse executado em uma *thread* separada. Foi adotada essa opção para garantir uma maior independência dos agentes entre si, buscando também se beneficiar da concorrência proporcionada por processadores com múltiplos núcleos.

Para garantir a sincronia entre os agentes, foi criada a classe `AgentTimer`, que utiliza o relógio do computador para informar o tempo corrido desde o último ciclo do agente. Essa classe é de grande importância pois garante que a simulação funcione como tempo real. As funções dos agentes relacionadas a tempo se baseiam no tempo informado por esta classe para que façam movimentos proporcionais ao tempo decorrido.

Figura 19 - Primeiras classes do sistema



Fonte: elaborado pelo autor.

Após a implementação das principais funções, notou-se, a partir de outros softwares, a necessidade de armazenar o conhecimento do agente de uma forma simples e flexível. É necessário que o agente possa ter variáveis a seu respeito, do ambiente, seus objetivos e sobre agentes próximos, e possa adicionar e remover essas informações com eficiência. Para isso, foi adicionada a classe `BeliefBase`, que representa a base de conhecimento do agente. Essa classe contém um `Dictionary`, que armazena um par índice-valor e tem seu acesso mais rápido por meio da ordenação dos índices.

Para armazenar os próximos pontos do mapa em que faz alguma interação, o veículo tem como atributo uma fila listando-os, com cruzamentos ou endereços de destino. Mas essa simplificação provou-se insuficiente em testes e foi adicionada uma lista de objetivos, gerada a partir da rota, que indica ações como acelerar, manter velocidade e frear. Cada objetivo está representado em uma instância da classe `AgentAction` e possui uma lista de `ActionTrigger`, que guarda as condições para o objetivo ser considerado cumprido e uma ação opcional para executar na sua conclusão.

Com as implementações anteriormente citadas, tornou-se possível desenvolver a primeira versão funcional do projeto, na qual é criado um veículo que acelera até a velocidade máxima e freia ao se aproximar do seu destino. Nessa versão ainda não há presença de múltiplos agentes ou interseções de vias, cujas implementações estão descritas na seção 3.3.4.

Como última parte desta etapa inicial, foi desenvolvida uma rotina que facilita o cadastro de mapas no sistema. A rotina lê um arquivo do tipo JSON, exemplificado no Quadro 4. No arquivo são descritas as ruas presentes nos mapas, os pontos de união e/ou cruzamentos e os trechos que os unem. Uma documentação completa de como escrever um arquivo de inicialização se encontra na seção 3.3.7.

Quadro 4 - Exemplo de arquivo JSON de ambiente

```
{
  "name": "Nome do Mapa",       // opcional
  "author": "Matias Henschel",  // opcional
  "log": "C:\\logs\\test.txt",  // opcional
  "autonomous": true,           // opcional
  "roads": [ {                  // Ruas
    "name": "Rua Bonita",
    "speedLimit": 11.0    // opcional, em m/s
  } ],
  "nodes": [              // Cruzamentos
    { "x": 0, "y": 400 }, // 0
    { "x": 800, "y": 400 }      // 1
  ],
  "edges": [ {                  // Trechos
    "road": "Rua Bonita",
    "from": 0,
    "to": 1,
    "direction": 2,       // opcional
    "length": 800,        // opcional
    "lanes": [ 1, 1 ]           // opcional
  } ]
}
```

Fonte: elaborado pelo autor.

Para fazer a desserialização do arquivo, lendo-o em forma de objeto, fez-se necessário a utilização de uma biblioteca, uma vez que a linguagem C# não apresenta uma rotina nativa para o mesmo. Em uma busca online foram consideradas algumas alternativas, optando-se pela biblioteca Json.NET. Essa opção é desenvolvida especialmente para o universo .NET e, em comparativos feitos pelo próprio desenvolvedor, apresenta performance superior a outras bibliotecas.

3.3.4 Desenvolvimento da comunicação entre agentes

Para tornar o ambiente multiagente cooperativo fez-se necessária a presença de uma forma de comunicação entre os veículos, e, conforme discutido na seção 3.3.1, optou-se pelo uso da biblioteca Boris para essa função. Para ter disponíveis as rotinas de comunicação exigidas para isso, é necessário apenas estender a classe `MetaAgent` disponibilizada pela biblioteca. Essa classe garante que o agente possa executar o método `SendMessage()` para enviar mensagens e executa métodos adicionados ao atributo `MessageReceived` quando recebe uma mensagem. A classe abstrata `Agent` estende a classe `MetaAgent`.

Para a troca de mensagens, o método aceita somente dois parâmetros: o agente destinatário e o conteúdo da mensagem, ambos do tipo `string`. Sendo assim, uma implementação completa do padrão FIPA-ACL (citado na seção 2.3.1) não é possível, pois alteraria código que está incluso dentro da biblioteca. Com essa situação o conteúdo da

mensagem se torna parte de grande valor, pois expressa todas as informações ao agente destino.

Para aprimorar o conteúdo da mensagem, foi originada a classe `Message`, a qual possui três atributos: um tipo de conteúdo, indicando a intenção da mensagem; um título e uma base de conhecimento, na qual são armazenadas variáveis relevantes para a mensagem. Para poder ser transmitida pelo método `SendMessage()`, a classe é serializada no padrão XML por meio de funções presentes no C#.

Dispondo da habilidade de comunicação, o primeiro desafio é executar a travessia de dois veículos por um cruzamento prevenindo qualquer risco de colisão. Para garantir uma mesma hierarquia entre os agentes (sem preferências), foi optado que a orientação dos veículos e organização dos momentos de travessia acontecesse na interseção. Para que isso fosse possível, a classe `Intersection` tornou-se um agente por meio de herança da classe `Agent`.

Sendo assim, o agente na interseção precisa, ao receber uma mensagem, verificar a possibilidade de o agente atravessar a via com segurança segundo os parâmetros por ele informados. Para fazer a verificação, a interseção armazena todas os momentos de travessia em uma lista ordenada. Ao receber uma nova solicitação de travessia, a interseção busca na lista outros agendamentos que possam coincidir com o solicitado por meio de um comando LINQ. O LINQ é um componente nativo à plataforma .NET que dá acesso aos dados de forma similar a um comando SQL, facilitando o acesso a um conjunto de dados. Nas linhas 2 a 7 do Quadro 5 pode ser observado um comando LINQ que filtra os itens de `schedule` que podem ter colisão com o veículo. Com os resultados, busca uma opção de reorganização das rotinas e avisa os agentes envolvidos.

Quadro 5 - Código de busca de travessias coincidentes

```
1 List<CrossingScheduleItem> subset = new List<CrossingScheduleItem>(
2     from item in schedule
3     where
4         (item.Road != road) &&
5         (item.From <= vTo && item.To >= vFrom)
6     orderby item.To ascending
7     select item
8 );
```

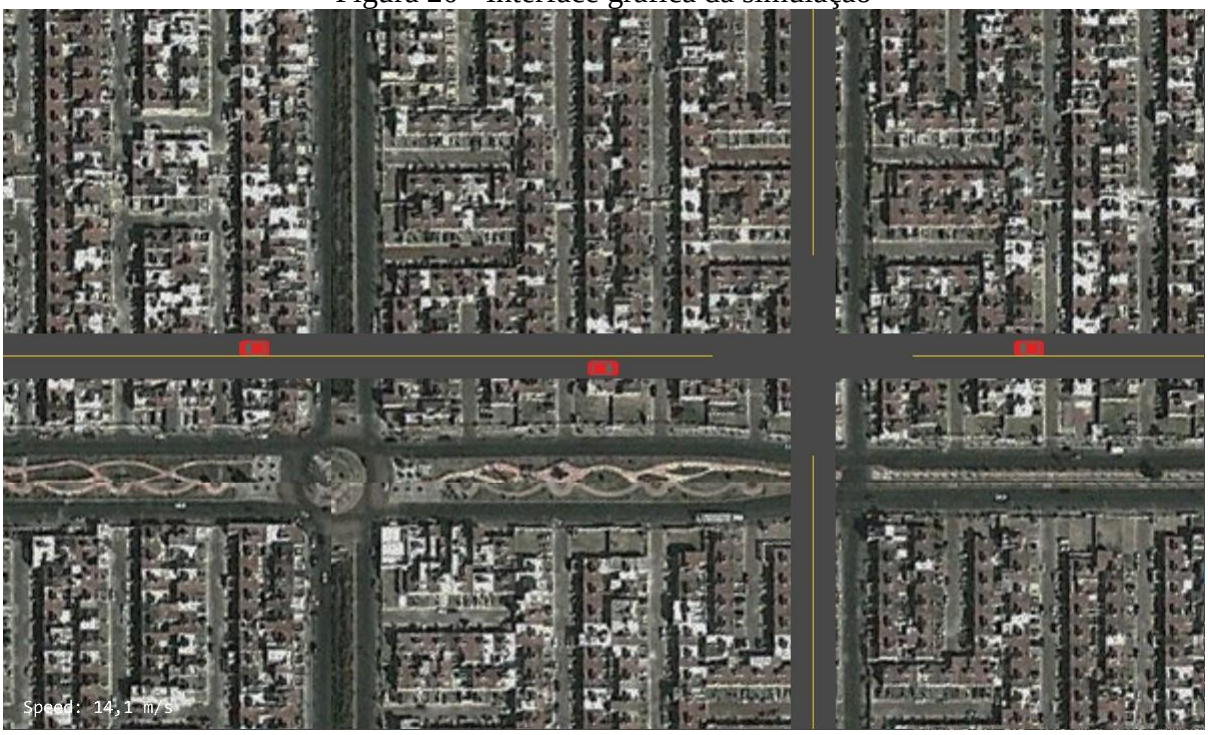
Fonte: elaborado pelo autor.

3.3.5 Visualização gráfica da simulação

Para avaliar a simulação com maior precisão, a próxima etapa no desenvolvimento tornou-se a visualização gráfica desta. Com foco na rápida construção da visualização, optou-se pela plataforma MonoGame.

Essencialmente, a janela tem a necessidade de mostrar as vias e os veículos em suas posições. O ambiente é mostrado em uma interface de duas dimensões, com vista área (também chamada de *top-down*). O usuário pode mover a câmera pelo ambiente ou optar para que a câmera acompanhe um veículo, além de poder aproximar-se ou afastar-se por meio de uma função de zoom. A interface completa pode ser visualizada na Figura 20, que inclui um fundo para melhor identificação das posições e, no caso da simulação corrente, duas vias de mão dupla que se cruzam e três veículos.

Figura 20 - Interface gráfica da simulação



Fonte: elaborado pelo autor.

Com o intuito de separar a simulação da visualização gráfica, foi decidido que estes seriam distintas aplicações. Entretanto, para que estas estivessem separadas, é necessário o uso de uma tecnologia na qual ambas pudessem trocar informações. Pensando na possibilidade de separar-se inclusive o processamento destes em diferentes máquinas, Web Service mostrou-se como uma opção bastante viável, sendo uma tecnologia aberta e suportada por vários sistemas operacionais. Para armazenar os dados foi utilizado a classe `DataTable`, que cria um banco de dados não persistente, ou seja, disponível somente enquanto a simulação está rodando. E para adicionar o Web Service ao sistema, optou-se pelo WCF.

Para inserir o Web Service, é necessário utilizar a biblioteca nativa `System.ServiceModel` e criar uma interface que indica os métodos presentes no serviço (nesse caso `IDataService`), assim como adicionar algumas configurações à interface e ao arquivo `App.config`. No caso deste trabalho, o serviço fica hospedado em um IP disponível para a rede

local na porta 8000. Para acelerar o processo de atualização dentro do próprio sistema, optou-se por utilizar internamente a mesma instância de objeto utilizada dentro do Web Service. Dessa forma os agentes têm acesso diretamente à classe `DataStorage` (classe que implementa a interface). O código que cria a interface é bastante compacto e pode ser observado no Quadro 6.

Quadro 6 - Código para inicialização do Web Service

```
 1  Uri baseAddress = new Uri("http://192.168.0.100:8000/MVP1/");
 2  DataStorage dataStorage = new DataStorage();
 3  ServiceHost selfHost = new ServiceHost(dataStorage, baseAddress);
 4
 5  try {
 6     selfHost.AddServiceEndpoint(typeof(IDataService),
           new WSHttpBinding(), "DataService");
 7     selfHost.Open();
 8  } catch (CommunicationException ce){
 9      selfHost.Abort();
10  }
```

Fonte: elaborado pelo autor.

Para fazer a leitura do serviço via C# é necessário vinculá-lo ao projeto. No Visual Studio 2017 é possível adicionar um serviço pela interface simplesmente informando o seu endereço quando este está ativo. Tendo o projeto adicionado, basta importar o serviço no código e utilizar a classe cliente, acessando os métodos de forma similar à uma instância dentro do próprio projeto. O Quadro 7 mostra um trecho de código que acessa o serviço e coleta informações dos veículos presentes na simulação.

Quadro 7 - Código para acesso ao Web Service

```
1  using NomeDoProjeto.MVP1Storage;
2  ...
3  DataServiceClient dsClient;
4  dsClient = new DataServiceClient("BasicHttpBinding_IDataService",
      "http://192.168.0.100:8000/MVP1/DataStorage");
5
6  VehicleAttributes[] listaVeiculos = dsClient.GetVehiclesList();
```

Fonte: elaborado pelo autor.

Na classe `DataStorage`, devido ao acesso de diversas instâncias e processos, foi necessário adicionar um comando de bloqueio (`lock`) nas chamadas de leitura e escrita à tabela. Dessa forma garante-se que o dado seja consistente e não seja alterado durante uma leitura.

A conclusão desta etapa auxiliou na visualização da simulação, facilitando a compreensão de irregularidades e correção destas. Por estar hospedado em um endereço de rede local, testes puderam realizar-se acertadamente em máquinas distintas presentes na mesma rede.

3.3.6 Crescimento da complexidade da simulação

As etapas previamente citadas constroem as características mais importantes do sistema. A partir deste ponto, são adicionados aspectos que contribuem com a complexidade da simulação.

O primeiro incremento é o aumento no número de pistas das vias. Inicialmente os testes foram feitos com somente vias simples de mão única. Para incrementar a simulação, foi adicionada a capacidade de simular vias de mão dupla com múltiplas pistas em cada sentido. Desta forma, a interseção deve autorizar que dois veículos percorram o cruzamento ao mesmo tempo se estes seguem em vias paralelas, ou seja, não tem possibilidade de colisão.

A adição de uma maior quantidade de agentes possibilita analisar a capacidade de gerência do cruzamento. É necessário que, apesar de filas, os veículos tenham um baixo tempo de espera no cruzamento e atravessem-nos com segurança. Além disso, veículos que estão na mesma via devem tomar cuidado com colisões com os veículos que estão à sua frente e notificar veículos que estão atrás sobre frenagens ou reduções na velocidade.

Para que fosse possível adicionar agentes com facilidade, foi adicionada a opção de criá-los por meio de um arquivo JSON padronizado que tenha o mesmo nome do arquivo de ambiente, porém com extensão `.agents.json`. A existência desse arquivo é verificada após a leitura do mapa, seguida do início da simulação. A especificação do uso dos arquivos JSON pode ser encontrada no APÊNDICE A.

Para avaliar o desempenho dos veículos conectados, o ambiente foi adaptado para suportar veículos dirigidos em um ambiente com semáforos. Para isso, foi criada a classe `DrivedVehicle`, que herda de `Vehicle`, assim como `Car`. Também foi adicionada a classe `Semaphore`, que herda de `Intersection` e a substitui nos ambientes não conectados. Diferente dos veículos conectados, os veículos dirigidos observam apenas a distância do veículo ou semáforo à frente para decidirem por acelerar ou frear. Para indicar se a simulação deve ser com os veículos conectados, o arquivo JSON de especificação do ambiente recebeu uma variável `autonomous`, que recebe `true` ou `false` para indicar o tipo de simulação.

3.3.7 Operacionalidade da implementação

Para que o usuário possa executar uma simulação sem ter a necessidade de codificar, são necessários dois arquivos do tipo JSON. Estes arquivos especificam o ambiente e os agentes da simulação.

A seguir são descritas especificações dos arquivos e como utilizá-los no simulador. Valores padrão para as características do ambiente são definidos no APÊNDICE A - Quadro

17. As unidades utilizadas no arquivo seguem o padrão internacional de unidades, sendo m (metros) para distâncias (o que inclui coordenadas), m/s (metros por segundo) para velocidade e m/s² (metros por segundo ao quadrado) para aceleração e desaceleração.

Para o ambiente, o arquivo de definições contêm atributos como os nomes das ruas presentes no mapa, os vértices – normalmente cruzamentos de vias – e os trechos que os conectam. Cada rua, além do nome, pode ter um limite de velocidade especificado. Para os cruzamentos basta especificar sua coordenada. Cada trecho precisa ter pelo menos três atributos especificados, que indicam a rua a qual ele pertence e quais vértices ele liga, sendo utilizado o índice da listagem como valor. A direção especificada indica o sentido em que cresce a numeração do trecho. Um arquivo simplificado pode ser visualizado no Quadro 4 - p. 45 e um exemplo utilizado em um dos testes pode ser encontrado no APÊNDICE A - Quadro 15.

O segundo arquivo representa os agentes que o ambiente irá simular. Este arquivo deve ter o mesmo nome do primeiro, porém deve ter a extensão `.agents.json`, se diferenciando de somente `.json` no anterior. Para este arquivo existem duas opções de adicionar agentes ao ambiente. A primeira opção é definir individualmente os agentes, indicando o endereço onde se encontram e opcionalmente qual o destino que devem alcançar e algumas características do veículo. Esta opção se encontra no atributo `spawn` do arquivo JSON. Com o segundo atributo da raiz (`autospawn`) é possível definir vias em que veículos serão criados automaticamente. Neste atributo deve-se definir qual a rua em que os veículos serão criados e um intervalo em milissegundos que será utilizado na geração de veículos como intervalo entre as criações. Demais parâmetros opcionais podem ser observados em um arquivo simplificado no Quadro 8, além de um arquivo completo utilizado em um dos exemplos que pode ser encontrado no APÊNDICE A - Quadro 16.

Quadro 8 - Exemplo de arquivo JSON de agentes

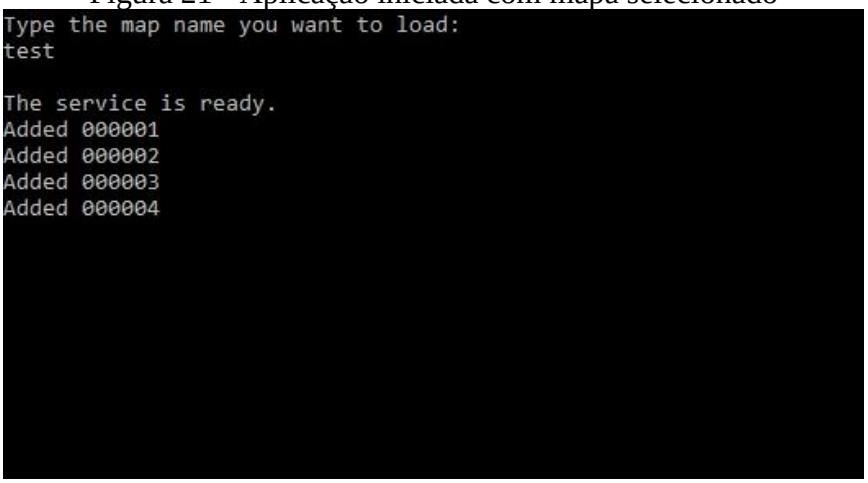
```
{
  "spawn": [ {                  // Criar agentes
    "address": {
      "street": "Nome da Rua",
      "number": 100,
      "lane": 1                 // opcional
    },
    "vehicle": {                // opcional
    },
    "destination": {            // opcional
      "street": "Nome da Rua",
      "number": 700,
      "lane": 1                 // opcional
    },
    "delay": 0                  // opcional
  } ],
  "autospawn": [ {        // Criação automática de agentes
    "road": "Nome da Rua",
    "minInterval": 4000,
    "maxInterval": 8000,  // opcional
    "spawnSpeed": 10,           // opcional
    "lanes": [ 1, 1 ]           // opcional
  } ]
}
```

Fonte: elaborado pelo autor.

Tendo os dois arquivos corretamente formatados e nomeados na pasta Maps, o usuário pode executar a aplicação. Primeiramente é necessário iniciar o roteador Boris (visto na p. 41 - Figura 17) divulgando a porta 1234, definida no código por ser a porta padrão da ferramenta. Em seguida, o usuário deve iniciar a aplicação `CavSim.exe` em seu computador e será aberta uma janela do tipo *console*. Esta janela solicita, ao iniciar, o nome do mapa a ser carregado. Deve-se então digitar o nome do arquivo na pasta sem a sua extensão. Neste momento a aplicação carrega o mapa e os agentes a serem simulados e inicia a simulação. A Figura 21 apresenta a ferramenta iniciada no qual o usuário optou pelo ambiente definido nos arquivos `test`.

Para que a simulação possa rodar o Web Service em uma rede local, foi optado pela definição de um IP de rede padrão para o serviço, sendo este `192.168.0.100`, que deve ser o endereço local da máquina durante a simulação. Mesmo que a visualização gráfica seja opcional, o sistema tem a necessidade de utilizar o endereço fixo para definir o serviço. Em alguns casos há a necessidade de iniciar a aplicação em modo administrador para a inicialização do serviço.

Figura 21 - Aplicação iniciada com mapa selecionado



Fonte: elaborado pelo autor.

Com as condições prévias corretamente atendidas, basta rodar a aplicação `CavSimInterface.exe` que a visualização gráfica será aberta. Dentro da aplicação é possível movimentar a câmera com as setas do teclado, e para acompanhar um dos agentes é necessário apertar um número do teclado numérico para selecionar o agente e `Esc` para desselecioná-lo. A Figura 22 mostra a interface gráfica com os agentes simulados.

Figura 22 - Interface gráfica da simulação acompanhando um agente

Fonte: elaborado pelo autor.

3.4 ANÁLISE DE RESULTADOS

O sistema multiagente desenvolvido permite a simulação de veículos conectados em cruzamentos e a comparação destes com veículos dirigidos utilizando um semáforo na

interseção. A simulação pode ser visualizada por meio de uma interface 2D com informações em tempo real e os dados podem ser extraídos para um arquivo de `log` para uma avaliação mais detalhada. Com isso, foram efetuadas simulações comparando o mesmo ambiente com veículos conectados e com veículos dirigidos.

Na seção 3.4.1 são especificados os testes executados para avaliar o sucesso dos veículos conectados. A seção 3.4.2 avalia os testes a partir dos arquivos de `log` gerados, incluindo dados estatísticos. As características práticas observadas nos dados estatísticos são apontadas na seção 3.4.3. Por fim, a seção 3.4.4 apresenta uma comparação deste trabalho com os trabalhos correlatos citados na seção 2.4.

3.4.1 Especificação dos testes

Para avaliar o desempenho dos veículos conectados em diferentes situações, optou-se pela execução de testes em ambientes distintos variando alguma característica do ambiente. A diferença se dá principalmente na quantidade de pistas do cruzamento. As especificações do computador que rodou os testes estão descritas no Quadro 9, sendo que estes foram executados dentro da ferramenta de desenvolvimento: Visual Studio Community 2017.

Quadro 9 - Características do computador de testes

Processador Intel i7-7700 3,6 GHz
Placa-mãe Asus ROG Strix B250F Gaming
Memória Kingston HyperX 8 GB 2400 MHz
Placa de vídeo EVGA GeForce GTX 970 SC 4 GB GDDR5
Sistema operacional Windows 10 Pro 64-bit versão 1703 build 15063.332

Fonte: elaborado pelo autor

As características definidas como padrão podem ser vistas no Quadro 10. Uma imagem da simulação do caso padrão para veículos conectados pode ser observada na Figura 23.

Figura 23 - Caso de teste padrão

Fonte: elaborado pelo autor.

Quadro 10 - Características padrão dos casos de teste

Vias: - 2 vias com 2 pistas cada (1 em cada sentido) - comprimento: 800 m cada, com ponto de encontro no centro - limite de velocidade: 15 m/s (54 km/h) Veículos: - aceleração máxima: 2,5 m/s² - desaceleração máxima: 4 m/s² - velocidade máxima: 30 m/s - comprimento: 5 m Semáforos: - duração do verde: 12 s - duração do amarelo: 3 s - duração do vermelho: 15 s (soma dos anteriores) Criação de veículos: - aleatoriamente em todas as pistas - intervalo entre 4000 e 8500 ms - velocidade inicial: 0 m/s

Fonte: elaborado pelo autor.

A primeira variação foi quanto ao número de pistas nas vias. Para o caso 1 foi diminuído o número de pistas, e para o caso 2 foram utilizadas o dobro de pistas em cada sentido. A especificação pode ser encontrada no Quadro 11. Em seguida, foi alterada a velocidade limite de cada via, novamente uma acima e uma abaixo do padrão, tendo-o como referência. O Quadro 12 apresenta a especificação destes testes. Como terceiro ponto, foi variada a quantidade de veículos simulada, diminuindo e aumentando o intervalo de criação de veículos. As características destes testes podem ser observadas no Quadro 13.

Quadro 11 - Especificação dos testes de variação de quantidade de pistas

Caso 1: 2 vias de mão única	Padrão: 2 vias com 2 pistas cada (sendo 1 em cada sentido)	Caso 2: 2 vias com 4 pistas cada (sendo 2 em cada sentido)

Fonte: elaborado pelo autor.

Quadro 12 - Especificação dos testes de variação de limite de velocidade

Caso 3: Limite: 10 m/s (36 km/h)	Padrão: Limite: 15 m/s (54 km/h)	Caso 4: Limite: 20 m/s (72 km/h)

Fonte: elaborado pelo autor.

Quadro 13 - Especificação dos testes de variação de quantidade de veículos

Caso 5: Intervalo de criação: 2000 a 5000 ms	Padrão: Intervalo de criação: 4000 a 8500 ms	Caso 6: Intervalo de criação: 8000 a 15000 ms

Fonte: elaborado pelo autor.

3.4.2 Comparação de resultados de testes

Para avaliar os casos de teste, foram executados os mesmos ambientes apenas com a alteração da variável `autonomous`, comparando assim os veículos conectados com os dirigidos.

Em cada teste foi aguardado que um número mínimo de 100 veículos chegasse ao seu destino. Não ocorreram colisões nos cruzamentos, cumprindo o requisito de garantir a segurança dos veículos e passageiros. Foram utilizados os arquivos de saída das simulações em uma breve aplicação Web desenvolvida com propósito de extrair o tempo de cada veículo para chegar ao seu destino. Esses tempos foram então adicionados a uma tabela no software Microsoft Excel, no qual foram feitos cálculos de média, mediana, variância e desvio padrão dos tempos. As tabelas a seguir apresentam os resultados dos testes com tempo especificado em milissegundos.

Tabela 1 - Estatísticas dos testes de variação de quantidade de pistas

	Caso 1		Padrão		Caso 2	
	Conectados	Dirigidos	Conectados	Dirigidos	Conectados	Dirigidos
Quantidade	110	112	123	118	155	155
T Mínimo	58896	58389	58964	58405	58200	58365
T Médio	59374,11	69284,30	59717,37	69867,89	60632,36	69161,88
T Mediano	59078	69307,5	59229	70103,5	60589	69506
T Máximo	60699	77360	62087	77441	62956	77472
Variância	306543,2	18073530,1	704863,3	17935029,4	1286847,5	20354292,1
Desvio Padrão	553,66	4251,30	839,56	4234,98	1134,39	4511,57

Fonte: elaborado pelo autor.

Tabela 2 - Estatísticas dos testes de variação de limite de velocidade

	Caso 3		Padrão		Caso 4	
	Conectados	Dirigidos	Conectados	Dirigidos	Conectados	Dirigidos
Quantidade	121	128	123	118	120	130
T Mínimo	83901	83347	58964	58405	47297	46696
T Médio	85312,44	94145,56	59717,37	69867,89	47775,43	58189,13
T Mediano	84838	94383,5	59229	70103,5	47393	58436,5
T Máximo	88778	102324	62087	77441	49916	66265
Variância	1838022,5	20552871,23	704863,3	17935029,4	345702,93	19205378
Desvio Padrão	1355,74	4533,53	839,56	4234,98	587,97	4382,39

Fonte: elaborado pelo autor.

Tabela 3 - Estatísticas dos testes de variação de quantidade de veículos

	Caso 5		Padrão		Caso 6	
	Conectados	Dirigidos	Conectados	Dirigidos	Conectados	Dirigidos
Quantidade	125	114	123	118	112	109
T Mínimo	58970	58690	58964	58405	58957	58367
T Médio	60705,77	78686,24	59717,37	69867,89	59334,12	65646,21
T Mediano	60697	78433	59229	70103,5	59081	64551
T Máximo	62621	90564	62087	77441	61387	77120
Variância	1407452,3	54168260,3	704863,3	17935029,4	275921,8	37208148
Desvio Padrão	1186,36	7359,91	839,56	4234,98	525,28	6099,85

Fonte: elaborado pelo autor.

3.4.3 Principais características observadas

A partir das estatísticas apresentadas na seção anterior, é possível identificar alguns pontos de destaque na comparação entre os veículos conectados e os dirigidos. Com relação ao tempo mínimo, há uma diferença de cerca de meio segundo em todos os casos, que se dá pela distribuição das fatias tempo aos agentes pelo processador, podendo ser ignorada. Já em relação ao tempo máximo é notável a diferença de tempo, chegando a mais de 14 segundos em todos os casos. Essa diferença é, principalmente, causada pelo tempo que o veículo aguarda no semáforo e que precisa para novamente alcançar a velocidade máxima da via após ter reduzido a velocidade. As filas causadas no semáforo podem ser visualizadas na Figura 24, que apresenta o caso 5 para veículos dirigidos.

Também é possível observar a maior consistência no tempo dos veículos conectados, com um desvio padrão muito inferior, com no máximo 30% do valor dos veículos dirigidos no caso 3. Sem a necessidade de espera, os veículos conectados precisam apenas regular a sua aceleração para garantir que cheguem em um momento autorizado pelo cruzamento, o que garante um tempo menor e mais consistente.

Outro detalhe observado a partir dos testes foi que os veículos conectados, uma vez que chegam a sua velocidade máxima, reduzem de velocidade apenas quando param em seu destino. Ou seja, a sua velocidade tem pouca variação durante o trajeto e chegam no cruzamento com uma velocidade próxima da máxima permitida. Com o conhecimento antecipado do momento permitido para cruzarem a interseção, pequenas regulagens na aceleração e velocidade são suficientes para chegarem com segurança e rapidez a seus destinos.

Figura 24 - Filas no cruzamento no caso de teste 5

Fonte: elaborado pelo autor.

A partir da análise dos casos apresentados, é possível comparar as características do ambiente e destacar os pontos em que os veículos conectados são mais eficientes. Sendo assim, os veículos têm melhor eficiência quanto a:

a) quantidade de pistas: quando o número é menor. Menos pistas indica uma menor quantidade de veículos cruzando, ou seja, mais intervalos livres e menor necessidade de rearranjar as solicitações pela interseção;
b) limite de velocidade: quando o limite é maior. A maior velocidade do veículo tem vantagens não somente por este poder mais facilmente trabalhar sua aceleração para chegar à velocidade limite, mas principalmente por fazer com que os veículos permaneçam por menos tempo na interseção;
c) quantidade de veículos: quando é menor. Assim como na quantidade de pistas, menos veículos significa mais intervalos livres para atravessar a interseção.

Uma dificuldade observada na simulação quanto aos veículos conectados pode ser observada quando o número de veículos é muito alto e a distância entre eles é curta. Como os veículos conectados utilizam apenas da sua comunicação para projetarem sua rota, em alguns casos dois veículos que estão na mesma via e pista recebem momentos de travessia muito próximos, e a distância entre suas coordenadas se torna menor do que o comprimento dos veículos. Esta situação pode ser verificada na Figura 25.

Figura 25 - Problema com sobreposição de veículos

Fonte: elaborado pelo autor.

Uma característica relacionada aos arquivos se dá na criação de mapas. A estrutura permite que diferentes cenários sejam montados, incluindo opções com mais de um cruzamento. Porém, em alguns casos de teste específicos foi observado que os veículos conectados sofrem um pequeno atraso na visualização quando se movem de um cruzamento para outro. O tempo de chegada permanece correto e o sistema evita colisões apesar do atraso.

Outro problema identificado se faz presente quando o número de agentes é alto e a capacidade de processamento do computador não é compatível com a simulação. A apresentação em tempo real pode ter um grande custo computacional, que, nos testes apresentados com a máquina citada no Quadro 9 não teve problemas. Porém, executando alguns casos em uma máquina com características inferiores foi notado um baixo número de ciclos de atualização por segundo, deixando a simulação imprecisa.

3.4.4 Comparação com trabalhos correlatos

Para comparação deste trabalho com os citados na seção 2.4, foi montado o Quadro 14, relacionando algumas características com os trabalhos citados, incluindo o sistema desenvolvido neste trabalho.

Inicialmente, observa-se que a utilização de SMA é comum para a simulação de veículos, sendo utilizada em todos os trabalhos listados, em contraste com a linguagem de programação, que apresenta diversas variedades. A simulação de grande escala é executada somente pelo SUMO (HERTKORN et al., 2002) e, consequentemente, o iTETRIS (CARTOLANO et al., 2008), que utiliza o SUMO como parte de seu projeto.

A otimização do tempo é uma característica adotada somente nos trabalhos que utilizam veículos conectados. Em especial, destaca-se o trabalho de Jin et al. (2012) pela semelhança do teste apresentado no artigo com os testes executados neste projeto. Ambos

analisam o desempenho de veículos conectados em um cruzamento, comparando-o com veículos dirigidos. Porém, o projeto de Jin et al. (2012) não permite personalização das vias ou uma visualização da simulação.

Quadro 14 - Comparação entre trabalhos correlatos e sistema desenvolvido

	Jin et al. (2012)	Hertkorn et al. (2002)	Freire (2004); Ranghetti (2007)	Cartolano et al. (2008)	Sistema desenvolvido (2017)
Utilização de SMA	Sim	Sim	Sim	Sim	Sim
Linguagem em que foi programado	C++	C++, Python	Object-Pascal	C++	C#
Simulação em grande escala (cidades)	Não	Sim	Não	Sim	Não
Veículos conectados	Sim	Não	Não	Sim	Sim
Otimização de tempo de espera e viagem	Sim	Não	Não	-	Sim
Personalização da malha rodoviária	Não	Sim	Sim	Sim	Sim
Visualização	Não	2D	2D / 3D	2D	2D
Visualização em tempo real	Não	Não	Não	Não	Sim

Fonte: elaborado pelo autor.

Um ponto de destaque deste projeto é a busca pela simulação dos veículos baseando-se no tempo do computador, ou seja, o tempo que os veículos utilizam para acelerar, frear ou chegar a um destino é próximo do tempo real que veículos com os mesmos atributos. Essa característica não pode ser observada nos outros projetos, que utilizam de outras técnicas para otimizar a simulação, dificultando assim sua capacidade de visualização e interpretação do ambiente.

4 CONCLUSÕES

A principal proposta do trabalho foi simular e comparar veículos conectados e tripulados em cruzamentos em um ambiente multiagente. Este objetivo foi alcançado com sucesso, observando-se diversas diferenças nos casos de teste descritos na seção 3.4, o que reitera as afirmações lidas em outros trabalhos sobre o futuro da tecnologia de comunicação em veículos autônomos.

O trabalho se mostra importante no meio acadêmico por trazer uma solução diferenciada para a simulação de veículos conectados, buscando ter uma apresentação de fácil interpretação pelo usuário com a simulação apresentada em tempo real. Esse atributo pode ter um custo computacional com o crescimento do número de veículos, exigindo uma capacidade de processamento coerente com a escala da simulação.

Na avaliação das ferramentas notou-se diversos pontos positivos e negativos, citados no Quadro 2. A opção pelo desenvolvimento de um *framework* próprio garantiu que não houvessem limitações por parte das ferramentas, habilitando a otimização deste sistema especificamente para simulação de veículos conectados.

No desenvolvimento a plataforma Boris.NET mostrou-se bastante fácil de implementar. Apesar de ser relativamente simples quando comparada às demais ferramentas estudadas, proveu a estrutura necessária para a comunicação eficiente entre os agentes simulados, com agilidade na troca de mensagens.

Outra vantagem apresentada pelo sistema é a de apresentar um sistema para veículos conectados com capacidade de alteração da malha viária, com capacidade de simular múltiplos cruzamentos. Porém, hoje o simulador limita-se a um tipo de veículo que segue somente em linha reta e somente vias retas verticais e horizontais. A adição de maior variedade de veículos, vias oblíquas e possibilidade de mudança de direção dos veículos poderia deixar a solução mais realista.

Com os testes executados, foram verificadas vantagens apresentadas pelos veículos conectados, comprovando uma de suas vantagens, citada no item 2.2. A consistência e eficiência do transporte foi comprovada sem detrimento da segurança nos cruzamentos. Também a validade de um simulador para verificar questões como essa foi reafirmada.

Como conclusão, é possível afirmar que o trabalho conclui seus objetivos, desenvolvendo um sistema para a simulação de veículos conectados e validando características destes.

4.1 EXTENSÕES

Possíveis extensões para este trabalho incluem:

a) possibilidade de simulação distribuída entre diversas máquinas, para maior poder computacional;
b) revisão do código de agendamento na interseção, para aprimorar a otimização dos intervalos de cruzamento;
c) maior complexidade nas vias, incluindo ruas com trechos em sentidos opostos, quantidade de pistas diferentes e trechos não lineares;
d) possibilidade dos veículos mudarem de pista;
e) possibilidade dos veículos mudarem de direção;
f) maior variedade de veículos simulados;
g) melhorias na interface gráfica, exibindo mais informações sobre a simulação;
h) possibilidade de controlar um agente selecionado;
i) interface para seleção do arquivo de simulação e inicialização da interface gráfica.

REFERÊNCIAS

ALBUQUERQUE, Marcos C. C. O custo econômico do congestionamento. **Folha de S. Paulo**, São Paulo, [S.n.], 14 abr. 2008. Disponível em: <http://marcoscintra.org/mc/custo-economico-congestionamento/>. Acesso em: 09 set. 2016.

ANFAVEA. **Estatísticas**. São Paulo, 2016. Disponível em: <http://www.anfavea.com.br/estatisticas-2016.html>. Acesso em: 05 set. 2016.

AZEVEDO, Rita. As principais causas de mortes no Brasil (e como evitá-las). **Exame.com**, São Paulo, [S.n], 17 jun. 2015. Disponível em: <http://exame.abril.com.br/brasil/as-principais-causas-de-mortes-no-brasil-e-como-evita-las/>. Acesso em: 07 jun. 2017.

BARCELÓ, Jaime et al. **Smartest:** Simulation modelling applied to road transport european scheme tests. Maio, 1999. 112 p. Disponível em: <http://www.its.leeds.ac.uk/projects/smartest/deliv6f.html>. Acesso em: 28 out. 2016.

BAUZA, Ramon et al. iTETRIS: a modular simulation platform for the large scale evaluation of cooperative ITS applications. **Simulation modelling practice and theory**, [S.l.], v. 34, [S.n.], p. 99-125, mar. 2013. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1569190X1300018X>. Acesso em: 6 set. 2016.

BAKER, Albert; CHAUHAN, Deepika. In: International conference on Autonomous agents, 2., 1998, [S.l.]. **Proceedings...** Minneapolis: ACM, 1998, p. 100-107. Disponível em: <http://dl.acm.org/citation.cfm?doid=280765.280782>. Acesso em: 8 maio 2017.

BECKER, Henrik et al. Autonomous vehicles: the next jump in accessibilities? **Research in Transportation Economics**, Zurich, [S.v.], [S.n.], p. 1-12, Apr. 2017. Disponível em <http://www.sciencedirect.com/science/article/pii/S0739885917300021>. Acesso em 24 abr. 2017.

BÉLANGER, Jean; VENNE, Philippe; PANQUIN, Jean-Nicolas. The What, Where and Why of Real-Time Simulation. **IEEE Power and Energy General Meeting**, Minneapolis, p. 37-49, Jul. 2010. Disponível em <http://sites.ieee.org/pes-resource-center/files/2013/10/11TP255E.pdf>. Acesso em: 16 jun. 2017.

BERGERO, Federico; KOFMAN, Ernesto. PowerDEVS: a tool for hybrid system modeling and real-time simulation. Simulation, v. 87, n. 1-2, p. 113-132, Jan. 2011. Disponível em <http://dl.acm.org/citation.cfm?id=1899673.1899679>. Acesso em: 16 jun. 2017.

BORDINI, Rafael H.; HÜBNER, Jomi F.; WOOLDRIDGE, Michael. **Programming multi-agent systems in AgentSpeak using Jason.** West Sussex: John Wiley & Sons, 2007.

BOUDETTE, Neal E. GM expands self-driving car operations in Silicon Valley. **The New York Times**, [S.n.], 13 apr. 2017. Disponível em <https://www.nytimes.com/2017/04/13/business/gm-expands-self-driving-car-operations-to-silicon-valley.html>. Acesso em: 24 abr. 2017.

CALDERON, Maria et al. Vehicle to Internet communications using the ETSI ITS GeoNetworking protocol. **Transactions on Emerging Telecommunications Technologies**, [S.l.], v. 27, n. 3, p. 373–391, Mar. 2016. Disponível em: <http://onlinelibrary.wiley.com/doi/10.1002/ett.2895/full>. Acesso em: 26 abr. 2017.

CARTOLANO, Fabio et al. iTETRIS: an integrated wireless and traffic platform for real-time road traffic management solutions. In: WIRELESS WORLD RESEARCH FORUM, 21., 2008, Estocolmo. **Proceedings...** Zurich: Wireless World Research Forum, 2008. p. 1-7. Disponível em: <http://www.ict-itetris.eu/documents/conferences/iTETRIS_Integrated_Wireless_Traffic.pdf>. Acesso em: 3 set. 2016.

CHENG, Nan et al. Connected vehicles: solution and challenges. **IEEE Internet of Things Journal**, [S.l.], v. 1, n. 4, p. 289-299, Aug. 2014. Disponível em: <http://ieeexplore.ieee.org/document/6823640/>. Acesso em: 16 ago. 2016.

CRAIG, Donald C. **Extensible hierarchical object-oriented logic simulation with an adaptable graphical user interface**. 1996. Dissertação (Master of Science) - Department of Computer Science, Memorial University of Newfoundland, St. John. Disponível em: <http://web.cs.mun.ca/~donald/msc/thesis.html> Acesso em: 16 jun. 2017.

DONIEC, et al. A behavioral multi-agent model for road traffic simulation. **Engineering Applications of Artificial Intelligence**, New York, v. 21, n. 8, p. 1443-1454, Dec. 2008. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0952197608000456>. Acesso em: 18 ago. 2016.

DOWLING, Richard et al. Performance Benefits of Connected Vehicles for Implementing Speed Harmonization. In: International Symposium on Enhancing Highway Performance, [S.n], 2016. **Proceedings…** Berlin: Elsevier B.V., 2016. p. 459-470. Disponível em <http://www.sciencedirect.com/science/article/pii/S2352146516305713>. Acesso em: 16 jun. 2017.

FIGUEIREDO, Miguel C. **An approach to simulation of autonomous vehicles**. 2009. 90 f. Dissertação (Major in Telecommunications) - Integrated Master in Electrotechnical and Computer Engineering, Faculdade de Engenharia da Universidade do Porto, Porto. Disponível em <https://repositorio-aberto.up.pt/handle/10216/60543>. Acesso em: 4 abr. 2017.

FREIRE, Jocemar J. **Simulação de controle de tráfego de automóveis em uma malha rodoviária urbana**. 2004. 47 f. Dissertação (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://dsc.inf.furb.br/arquivos/tccs/monografias/2007-1maycoandreyranghettivf.pdf>. Acesso em: 23 ago. 2016.

HERTKORN, Georg et al. **SUMO (Simulation of Urban Mobility)**: an open-source traffic simulation. 2002. SUMO. Disponível em <http://sumo.dlr.de/pdf/dkrajzew_MESM2002_SUMO.pdf>. Acesso em: 20 ago. 2016.

HESSE, Tobias. **Highly and fully automated driving**: How can the driver spendthetime? In: THE ROAD TO AUTOMATED DRIVE, [S.n.], 2014, Stuttgart. Disponível em: <http://elib.dlr.de/90165/1/AU_TS_Presentation_AutonomatedDrive_Stuttgart_140526_v1.pdf>. Acesso em: 8 maio 2017.

HORLING, Bryan; LESSER, Victor. A survey of multi-agent organizational paradigms. **The Knowledge Engineering Review**, Cambridge, v. 19, n. 4, p. 281-316, 2005. Disponível em: <https://www.cambridge.org/core/journals/knowledge-engineering-review/article/survey-of-multiagent-organizational-paradigms/A07BCCB1379F001DE995F3E5476EE4AB>. Acesso em: 20 abr. 2017.

HÜBNER, Jomi F. **Migração de agentes em sistemas multi-agentes abertos**. 1995. 126 f. Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: <http://www.lume.ufrgs.br/handle/10183/25032>. Acesso em: 27 out. 2016.

IEEE. **Foundation for Intelligent Physical Agents**, [S.l.], 2017. Disponível em <http://www.fipa.org>. Acesso em: 8 maio 2017.

INTEL. **Como começar a desenvolver para direção automática**, [S.l.], 2017. Disponível em <https://software.intel.com/pt-br/articles/automated-driving/get-started>. Acesso em: 8 maio 2017.

INTEL CORPORATION. **Intel GO™ autonomous driving solutions:** autonomous driving, accelerated. [S.l.], 2017. Disponível em <http://www.intel.com/content/dam/www/public/us/en/documents/platform-briefs/go-automated-accelerated-product-brief.pdf>. Acesso em: 9 maio 2017.

JASON. **Getting started with Jason**. 2017. Disponível em <http://jason.sourceforge.net/mini-tutorial/getting-started/>. Acesso em: 22 maio 2017.

JIN, Qiu et al. Advanced intersection management for connected vehicles using a multi-agent systems approach. In: INTELLIGENT VEHICLES SYMPOSIUM, 2012, Alcalá de Henares. **Proceedings…** Piscataway: IEEE PSPB, 2012. p. 932-937. Disponível em: <http://ieeexplore.ieee.org/document/6232287/>. Acesso em: 16 ago. 2016.

KOMADA, Yoko et al. Short sleep duration, sleep disorders, and traffic accidents. **IATSS Research**, [S.l.], v. 37, n. 1, p. 1-7, Jul. 2013. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0386111213000149>. Acesso em: 28 out. 2016.

KRAUSS NETO, Artur. **Cenário brasileiro para os veículos elétricos**. 2013. 134 f. Dissertação (Mestrado em Engenharia Elétrica) - Centro de Ciências Tecnológicas, Universidade Regional de Blumenau, Blumenau. Disponível em <http://www.bc.furb.br/docs/DS/2013/359264_1_1.pdf>. Acesso em: 9 set. 2016.

LANDAU, Deb M. 5G: a comunicação é fundamental para a direção autônoma. **Intel IQ**, [S.n.], 7 mar. 2017. Disponível em <https://iq.intel.com.br/5g-a-comunicacao-e-fundamental-para-a-direcao-autonoma/>. Acesso em: 8 maio 2017.

LEE, Joyoung; PARK, Byungyu. Development and evaluation of a cooperative vehicle intersection control algorithm under the connected vehicle environment. **IEEE Transactions on Intelligent Transportation Systems**, [S.l.], v. 13, n. 1, p. 81-90, Mar. 2012. Disponível em: <http://ieeexplore.ieee.org/document/6121907/>. Acesso em: 16 ago. 2016.

LIDEN, Daniel. **What is a driverless car? [S.l.]**, 2016. Disponível em: <http://www.wisegeek.com/what-is-a-driverless-car.htm>. Acesso em: 2 nov. 2016.

MERSKY, Avi S.; SAMARAS, Constantine. Fuel economy testing of autonomous vehicles. **Transportation Research Part C: Emerging Technologies**, Pittsburgh, v. 65, [S.n.], p. 31-48, Apr. 2016. Disponível em <http://www.sciencedirect.com/science/article/pii/S0968090X16000024>. Acesso em: 16 jun. 2017.

NEWMAN, Daniel. Autonomous cars: the future of mobility. **Forbes**, [S.n.], 27 set. 2016. Disponível em <https://www.forbes.com/sites/danielnewman/2016/09/27/autonomous-cars-the-future-of-mobility/>. Acesso em: 20 abr. 2017.

OLFATI-SABER, Reza; FAX, J. Alex; MURRAY, Richard M. Consensus and cooperation in networked multi-agent systems. **Proceedings of the IEEE**, [S.l.], v. 95, n. 1, p. 215-233, Jan. 2007. Disponível em: <http://ieeexplore.ieee.org/document/4118472/>. Acesso em: 18 ago. 2016.

PANAIT, Liviu; LUKE, Sean. Cooperative multi-agent learning: the state of the art. **Autonomous Agents and Multi-Agent Systems**, Heidelberg, v. 11, n. 3, p. 387–434, Nov. 2005. Disponível em: <http://cs.gmu.edu/~eclab/papers/panait05cooperative.pdf>. Acesso em: 29 out. 2016.

RANGHETTI, Mayco A. **Simulador de tráfego de automóveis em uma malha rodoviária**: versão 2.0. 2007. 64 f. Dissertação (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.bc.furb.br/docs/MO/2008/329231_1_1.pdf> Acesso em: 23 ago. 2016.

RUSSEL, Stuart; NORVIG, Peter. **Artificial Intelligence:** a modern approach. 2. ed. Upper Saddle River: Pearson Education, 2003.

SILVA, Marcos B. **Sistema multiagentes para gerenciamento de tráfego urbano**. 2005. 88 f. Dissertação (Mestrado em Ciência da Computação), Universidade Federal do Maranhão, São Luís. Disponível em: <http://www.tedebc.ufma.br/tde_arquivos/10/TDE-2008-11-18T174317Z-257/Publico/Marcos%20Barros%20e%20Silva.pdf>. Acesso em: 27 out. 2016.

SUMO. In: SUMO2016, 30., 2016, Berlin-Adlershof. **Proceedings...** Braunschweig: Deutsches Zentrum für Luft- und Raumfahr, 2016. Disponível em: <http://www.dlr.de/ts/en/Portaldata/16/Resources/veranstaltungen/2016/SUMOconference_proceedings_2016.pdf>. Acesso em: 10 nov. 2016.

SUMO. **Screenshots**, [S.l.], 2011. Disponível em: <http://sumo.dlr.de/wiki/Screenshots>. Acesso em: 20 ago. 2016.

SUMO. **Simulation of Urban MObility**, [S.l.], 2017. Disponível em: <http://sumo.dlr.de/wiki/ Simulation_of_Urban_MObility_-_Wiki>. Acesso em: 15 jun. 2017.

US DOT. What Are the Benefits of Connected Vehicles? **Intelligent Transport Systems Joint Program Office**, Washington, [2017]. Disponível em <https://www.its.dot.gov/cv_basics/cv_basics_benefits.htm>. Acesso em: 16 jun. 2017.

WADHWA, Vivek. Self-driving cars should leave us all unsettled. Here's why. **The Washington Post**, Washington, 24 apr. 2017. Disponível em: <https://www.washingtonpost.com/news/innovations/wp/2017/04/24/self-driving-cars-should-leave-us-all-unsettled-heres-why/>. Acesso em: 8 maio 2017.

APÊNDICE A – Especificação do uso dos arquivos JSON

A seguir, no Quadro 15, encontra-se um arquivo JSON utilizado para descrever um ambiente de exemplo que possui duas vias sendo uma horizontal e uma vertical, ambas de mão dupla com duas pistas em cada sentido. Cada via possui 800 metros de comprimento e tem sua interseção bem ao centro.

Quadro 15 - Arquivo JSON de ambiente teste-exemplo.json

```
{
  "name": "Teste Exemplo",
  "name": "Matias G Henschel",
  "log": "C:\\logs\\teste-exemplo.txt",
  "autonomous": true,
  "roads": [ {
      "name": "Rua Horizontal"
  }, {
      "name": "Rua Vertical",
      "speedLimit": 11.0
  } ],
  "nodes": [
    { "x": 0, "y": 400 },   // 0
    { "x": 400, "y": 400 }, // 1
    { "x": 400, "y": 0 },   // 2
    { "x": 800, "y": 400 }, // 3
    { "x": 400, "y": 800 }  // 4
  ],
  "edges": [ {
      "road": "Rua Horizontal",
      "from": 0,
      "to": 1,
      "direction": 2,
      "length": 400,
      "lanes": [ 2, 2 ]
  },{
      "road": "Rua Horizontal",
      "from": 1,
      "to": 3,
      "direction": 2,
      "length": 400,
      "lanes": [ 2, 2 ]
  },{
      "road": "Rua Vertical",
      "from": 2,
      "to": 1,
      "direction": 2,
      "length": 400,
      "lanes": [ 2, 2 ]
```

```
  }, {
      "road": "Rua Vertical",
      "from": 1,
      "to": 4,
      "direction": 2,
      "length": 400,
      "lanes": [ 2, 2 ]
  } ]
}
```

Fonte: elaborado pelo autor.

O arquivo deste mesmo teste utilizado para especificar os agentes encontra-se no Quadro 16 e define os dois tipos de criação de veículos.

Quadro 16 - Arquivo JSON de agentes teste-exemplo.agents.json

```
{
  "spawn": [ {
    "address": {
      "road": "Rua Horizontal",
      "number": 100,
      "lane": 1
    },
    "vehicle": {
      "maxSpeed": 20,
      "maxAcceleration": 2.5,
      "maxDeceleration": 4,
      "color": "blue"
    },
    "destination": {
      "road": "Rua Horizontal",
      "number": 700,
      "lane": 1
    },
    "delay": 10000
  } ],
  "autospawn": [ {
    "road": "Rua Vertical",
    "minInterval": 4000,
    "maxInterval": 8000,
    "spawnSpeed": 10,
    "lanes": [ 1, 1, 1, 1 ]
  } ]
}
```

Fonte: elaborado pelo autor.

A seguir são descritos os campos com os valores padrão dos arquivos de criação para o caso de itens não especificados. O Quadro 17 é divido em três partes, sendo a primeira a legenda, a segunda é a especificação do arquivo de ambiente e a terceira a do arquivo de agentes.

Quadro 17 - Valores padrão para os arquivos JSON de criação

(>) "campo" : tipo (#opcional) ({ restrições }) ([valor padrão])
"name" : string #opcional "author" : string #opcional "log" : string #opcional "autonomous" : string #opcional { true, false } [true] "roads" : array { pelo menos um item necessário } > "name" : string > "speedLimit" : double #opcional [14] "nodes" : array { pelo menos um item necessário } > "x" : double > "y" : double "edges" : array { pelo menos um item necessário } > "road" : string > "from" : int > "to" : int > "direction" : int #opcional { 1 = ida, -1 = volta, 2 = dois sentidos } [2] > "length" : double #opcional [distância em linha reta entre os vértices ligados] > "lanes" : array[int] { deve ter 2 itens [ida, volta], sobrescreve o especificado em "direction" }
"spawn" : array #opcional > "address" : array > "road" : string > "number" : double > "lane" : int #opcional > "vehicle" : array #opcional > "maxSpeed" : double #opcional [40] > "maxAcceleration" : double #opcional [2.5] > "maxDeceleration" : double #opcional [2.5] > "color" : string #opcional ["red"] > "destination" : array #opcional { não negativo, mesmos campos de "address" } [final da rua] > "delay" : int #opcional { em milissegundos } [0] "autospawn" : array #opcional > "road" : string > "minInterval" : int > "maxInterval" : int #opcional [4 * "minInterval"] > "spawnSpeed" : double #opcional [0] > "lanes" : array[0/1] #opcional { um item para cada pista, 1 indica que devem ser criados veículos na pista, ordem da pista mais externa contra para a pista mais externa a favor } [todos itens 1] > "vehicle" : array #opcional { mesmos campos de "spawn > vehicle" }