UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ERICSON PINHEIRO PACHECO

DESENVOLVIMENTO DE UM PROTÓTIPO PARA VISUALIZAÇÃO DO POSICIONAMENTO DE TRENS EM UMA MALHA FÉRREA

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2018

ERICSON PINHEIRO PACHECO

DESENVOLVIMENTO DE UM PROTÓTIPO PARA VISUALIZAÇÃO DO POSICIONAMENTO DE TRENS EM UMA MALHA FÉRREA

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

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

PONTA GROSSA

2018

Dedico este trabalho à toda minha família e amigos, em especial à minha mãe que me ensinou a sorrir e sempre acreditar no meu potencial. Espero, um dia, poder retribuir tudo que a senhora fez por mim.

AGRADECIMENTOS

À minha família pelo carinho e apoio.

Ao meu orientador pela dedicação, paciência e ensinamentos.

Aos meus colegas de faculdade, colegas de trabalho e amigos.

A minha amiga Mariana, que por grande parte da minha trajetória me ajudou e apoiou sempre que precisei.

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Bacharelado em Ciência da Computação

TERMO DE APROVAÇÃO

DESENVOLVIMENTO DE UM PROTÓTIPO PARA VISUALIZAÇÃO DO POSICIONAMENTO DE TRENS EM UMA MALHA FÉRRA

por

ERICSON PINHEIRO PACHECO

Este Trabalho de Conclusão de Curso (TCC) foi apresentado às 13h30min de 24 de Maio de 2018 como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. André Pinz Borges
Prof. Orientador

_____
Prof. Dr. Gleifer Vaz Alves
Membro titular

_____
Prof. Dr. André Koscianski
Membro titular

_____
Prof. Dr. Helyane Bronoski Borges
Responsável pelo Trabalho de Conclusão de Curso

_____
Prof. Dr. Saulo Jorge Beltrão de Queiroz
Coordenador do curso

RESUMO

PACHECO, Ericson Pinheiro. **Desenvolvimento de um Protótipo para Visualização do Posicionamento de Trens em uma Malha Férrea**. 2018. 74f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2018.

O presente trabalho tem por objetivo o desenvolvimento de protótipo para a visualização do posicionamento de trens em uma malha férrea. Várias empresas de software desenvolvem soluções comerciais para gerenciamento e visualização dos posicionamentos de trens, contudo, a maioria das soluções são proprietárias e não permitem alterações em seus códigos-fonte tornando as opções de escolha muito limitadas. O protótipo foi desenvolvido utilizando o framework NetLogo, capaz de representar graficamente a simulação e interação de agentes em um determinado ambiente. Neste trabalho, os agentes compreendem os condutores de trens e o ambiente é a malha férrea onde os agentes trafegam. O uso de agentes permite simular o deslocamento de trens empregando regras de condução. Com este trabalho, foi obtido um software aberto e adaptável, que permite a visualização de um a três trens em uma malha férrea, podendo ser alterado via código fonte para possibilitar a simulação de mais trens.

Palavras-chave: Agentes. NetLogo. Condução de Trens.

ABSTRACT

PACHECO, Ericson Pinheiro. **Development of a Prototype for Visualizing the Position of Trains in a Railway Network.** 2018. 74f. Work of Conclusion Course (Graduation in Computer Science) - Federal Technology University - Paraná. Ponta Grossa, 2018.

This project has the objective of development of a prototype for the visualization of the positioning of trains in a railway network. Many software companies develop commercial solutions for managing and displaying train placements, however, most solutions are proprietary and do not allow changes to their source code, making choice very limited. The prototype was developed using the NetLogo framework, capable of graphically representing the simulation and interaction of agents in a given environment. In this work, the agents are the train drivers, and the environment is the the iron mesh where the agents travel. With this work was obtained an open and adaptable software that allows the visualization of several trains in a railway network and to assist in the management of the mesh and the visualization of train information during the simulations.

Keywords: Agents. NetLogo. Driving Trains.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ACO	Ant Colony Optimization
FIPA	Intelligent, Physical Agents
GPS	Serviço de Posicionamento Global
JADE	Java Agent DEvelopment Framework
SB	Seção de Bloqueio
SMA	Sistema Multi-Agente

SUMÁRIO

1 INTRODUÇÃO

No Brasil, o modal ferroviário teve início no século XIX com os primeiros trilhos construídos lingando Rio de Janeiro e Serra de Petrópolis. Tendo Irineu Evangelista de Souza como idealizador, as ferrovias desempenharam um importante papel desenvolvimento no transporte de cargas, sendo o primeiro modal capaz de deslocar grandes quantidades de cargas a longa distância. Durante o século XX, as ferrovias perderam parte de sua competitividade no transporte de cargas devido ao investimento no sistema rodoviário. Porém, com a privatização das ferrovias, esse cenário tem mudado com os investimentos e novas tecnologias recebidas (XAVIER, 2006).

Para que os trens possam trafegar em uma malha férrea, é necessário um controle de rastreamento para gerenciar o compartilhamento das vias e o posicionamento de todos os trens que a ocupam. Atualmente existem ferramentas que desempenham esse papel, como é o caso WebRail (SYSFER, 2017), Rail Traffic Controller (WILLSON, 2012), Railway Operation Simulator (BALL, 2017). Para realizar a simulação de um trem em um determinado trajeto, é necessário que haja um condutor para controle e tomada de decisões do veículo.

Uma forma de simular o comportamento dos maquinistas na ferrovia consiste em utilizar softwares com características de autonomia e proatividade. Tais características são alcançadas com o uso de Agentes Inteligentes. Um agente pode ser visto como uma entidade de software autônoma com objetivo bem definido, com autonomia, proatividade e comportamento social que permitem a ele executar ações para alcançar o objetivo (Wooldridge, 2009). As ações são executadas com base em estímulos recebidos via percepção do ambiente onde atuam. Os Agentes Inteligentes são capazes de tomar decisões por conta própria com base em um objetivo que é pré-definido durante sua criação. Por conta disso é possível utilizá-los em problemas que necessitem testar várias possibilidades de decisões a serem tomadas para a escolha da melhor opção (FERBER, 1999).

Um sistema de simulação pode ser composto por um conjunto de agentes, que interagem entre si cooperando uns com os outros, coordenando suas ações e, muitas vezes, negociando recursos. Neste cenário tem-se um sistema multiagente (SMA) (FERBER, 1999).

O NetLogo é um software livre que roda em uma máquina virtual Java e sua

documentação está disponível em seu site juntamente com exemplos de aplicações já desenvolvidas. O software é uma das ferramentas usadas para criação de SMAs. Netlogo permite implementar agentes e visualizar em tempo real suas interações via simulação. A ferramenta também é utilizada em diversas áreas de estudo, como as ciências biológicas, naturais e exatas, onde é necessário observar o comportamento de indivíduos atuando sem que sejam necessários comandos de um controlador (NETLOGO, 2016).

Neste trabalho será explorado o uso da ferramenta NetLogo para a implementação de uma ferramenta gráfica empregada afim de auxiliar na visualização do posicionamento dos trens de carga ao longo de uma malha férrea. A ferramenta permitirá visualizar informações de cada trem mostrando as informações em gráficos. Este trabalho permitirá visualizar, por exemplo, o comportamento do condutor elaborado por Borges (2015), o qual trata o problema sem apresentar uma interface de visualização das viagens, algo passível de implementação com NetLogo.

1.1 OBJETIVOS

Desenvolver um protótipo de um sistema para a visualização do posicionamento dos trens em uma malha ferroviária em tempo de execução.

A fim de atingir o objetivo geral, definiu-se os seguintes objetivos específicos:

- Revisar a literatura sobre Agentes Inteligentes;
- Compreender o domínio do problema acerca dos elementos que o compõe;
- Compreender formas de controlar o posicionamento dos trens em uma malha férrea;
- Revisar a literatura sobre trabalhos que utilizam NetLogo em domínios semelhantes;
- Implementar um sistema baseado em agentes utilizando NetLogo no domínio do problema;
- Analisar o comportamento dos agentes implementados via NetLogo.

1.2 JUSTIFICATIVA

Os trens detêm importante papel no transporte de cargas e podem impactar diretamente no orçamento dos produtos por conta do consumo de combustível

durante o deslocamento. Esse consumo varia com conforme é feita a condução do trem. Como cada viagem possui diferentes características, se faz necessário o uso de processos capazes de indicar quais as melhores decisões a serem tomadas durante o percurso. Além disso, é necessário controlar o posicionamento dos trens para evitar congestionamentos e paradas desnecessárias, o que também eleva o consumo de combustível (Borges, 2015).

O Brasil utiliza padrões no modal férreo que impedem que um trem invada a distância mínima de segurança de outro, garantindo assim uma distância mínima entre os trens. Este espaço é denominado Seção de Bloqueio (SB). Estudos foram realizados com o objetivo de evitar paradas desnecessárias via ultrapassagem de trens (DORDAL, 2016). Contudo, tais estudos não dispunham de uma interface gráfica que permita visualizar o posicionamento dos trens e implementar novos algoritmos de simulação.

Atualmente existem algumas arquiteturas pagas como é o caso do WebRail e Rail Traffic Controller, ou com código aberto estruturados em uma linguagem complexa como é o caso do Railway Operation Simulator (SYSFER, 2017), (WILLSON, 2012), (BALL, 2017). Com isso, faz-se necessária a implementação de um software livre e adaptável para suprir essa necessidade.

Neste contexto o NetLogo apresenta-se com uma boa solução, pois será possível visualizar graficamente o posicionamento dos trens e assim facilitar a compreensão da maneira como as locomotivas devem trafegar, obedecendo as regras de segurança ditadas pelas SB. O NetLogo também oferece uma linguagem de programação simplificada que facilita a adaptação de novas regras de condução.

O comportamento dos agentes desenvolvidos será baseado em configurações pré-estabelecida pelo usuário do *software*, como o peso, quantidade de vagões e ponto de aceleração do trem.

1.3 ESTRUTURA DO TRABALHO

Este documento está divido em cinco capítulos. O Capítulo 1 mostra uma ideia dos temas que serão abordados. O segundo capítulo fornece uma base de conhecimentos sobre trens e suas formas de condução, agentes inteligentes e frameworks para desenvolvimento de agentes. O terceiro capítulo descrevre o desenvolvimento do protótipo de simulação desenvolvido em NetLogo. O quarto

capítulo mostra os testes e resultados obtidos por meio da execução do protótipo. E por fim o quinto capitulo faz conclusões sobre o referencial teórico abordado e o desenvolvimento do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção será abordado um breve conhecimento acerca do funcionamento do controle de locomotivas, bem como os conceitos de agentes inteligentes e SMAs. Também serão apresentadas as características de ferramentas para desenvolvimento de agentes tendo como foco principal o framework NetLogo.

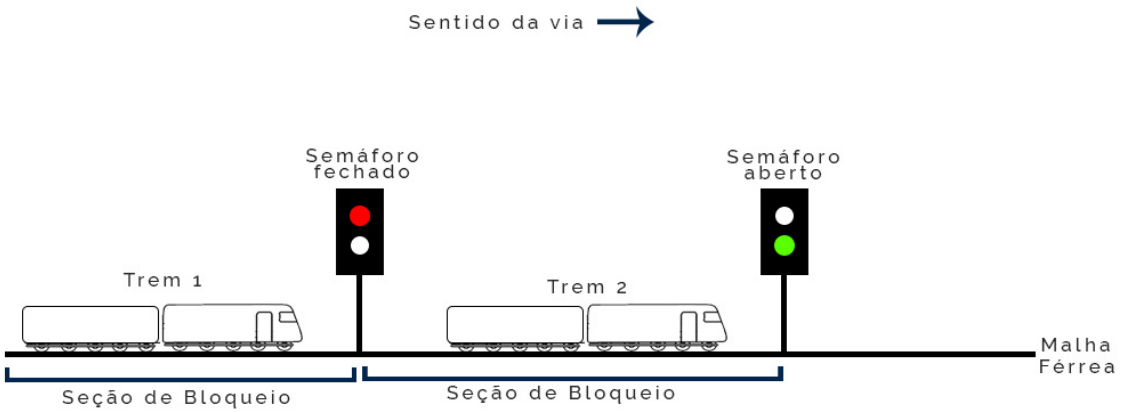
2.1 CONTROLE DE LOCOMOTIVAS

A locomoção de um trem de um ponto a outro exige do condutor o seguimento de vários protocolos que estabelecem a segurança durante a viagem. Com isso, as tomadas de decisões devem ser feitas com cautela. Por exemplo, um trem só deve começar a se mover quando todos os freios estiverem complemente soltos, deve procurar manter uma velocidade constante ao passar em certos tipos de terrenos e controlar o uso de freios e acelerações (BORGES, 2015).

Além disso, é importante conhecer do posicionamento de todos os trens que trafegam em uma determinada malha férrea. O primeiro sistema de sinalização das vias só permitia um trem andar em sua velocidade máxima quando o trem anterior tivesse dez minutos a frente. Logo se desenvolveram as sinalizações fixas que utilizam postes próximos as malhas férreas. Outro sistema criado para o auxilio da condução foi a comunicação de uma central com uma locomotiva por meio de sinais. Há também o sistema que indica, via sensores, a velocidade a ser seguida com base no trem que está trafegando mais a frente utilizando marcações durante a malha férrea (BORGES, 2015).

As seções de bloqueio foram as primeiras maneiras de evitar que trens ocupem o mesmo espaço na linha férrea, com isso, evitando colisões. As SBs são trechos delimitados da via onde apenas um trem pode trafegar em determinado momento. Quando o trem chegar em uma seção que já está sendo ocupada, ele deve aguardar até que a mesma seja desocupada. A desocupação ocorre quando um trem sai da seção de bloqueio. A forma do condutor saber se a seção esta liberada é estabelecida por um semáforo na via que fica vermelho assim que um trem entra na seção (SOLOMON, 2003), (BAJPAI et al, 2007).

Figura 1 – Representação de Seções de Bloqueio Sinalizadas por Semáforos



Fonte: Adaptado de Borges (2015).

A Figura 1 está representando uma malha férrea sinalizadas por semáforos para que as SBs não sejam invadidas por mais de um trem ao mesmo tempo. Sempre que um trem estiver trafegando em uma SB a frente, o semáforo permanecerá fechado até atingir uma distancia segura.

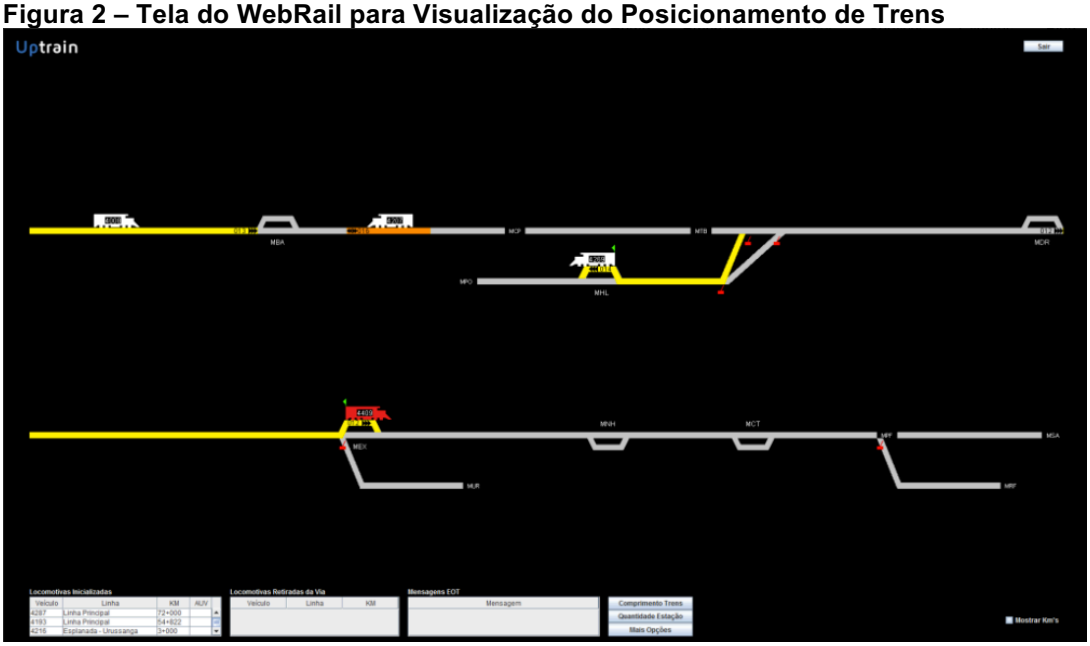
2.1.1 WebRail

O sistema WebRail tem por objetivo mostrar o acompanhamento de toda a malha férrea por meio da gestão de operações, gestão de ativos, gestão de planejamento de manutenção, controle da circulação de trens e gestão e projeção de custos e resultados. A gestão operacional trata de todo o planejamento referente às cargas que serão transportadas, equipe de tripulação e manobras que o trem deverá fazer para chegar até os terminais de clientes. Essas informações são apresentadas em uma tela com tabelas contendo os dados de cada viagem. A gestão de ativos tem como objetivo o acompanhamento de todo o patrimônio ligado a malha férrea. Esse acompanhamento é feito por meio obtenção de dados geográficos e mostrados em forma de mapa na tela do sistema. O planejamento de manutenção permite cadastrar com precisão os pontos da malha férrea onde são necessárias manutenções gerando os custos e os materiais necessários. Também é possível fazer o planejamento das manutenções pelo aplicativo *mobile* que é capaz de operar sem a necessidade de

internet. O controle de circulação é feito com o uso de equipamentos de rastreamento instalados nos trens para que seja possível acompanha-los e, se necessário, alerta-los de uma parada em vias que não sejam sinalizadas ou que necessitem de uma atenção maior. A gestão de custos permite calcular os valores para os segmentos quando os vagões estão carregados e também quando estão descarregados, sendo possível a geração de relatórios completos. Além da gestão de custos reais, também é possível projetar os custos de uma viagem apenas inserindo os valores de variáveis da viagem como parâmetros para o sistema (SYSFER, 2017).

O sistema possui várias telas, porém a que está representada na Figura 2 é mais relevante, pois este trabalho terá como base parte do conceito apresentado na imagem, a qual mostra basicamente posicionamento do veículo e trajetos secundários para desvios.

Figura 2 – Tela do WebRail para Visualização do Posicionamento de Trens



Fonte: Sysfer (2017)

2.1.2 Rail Traffic Controller

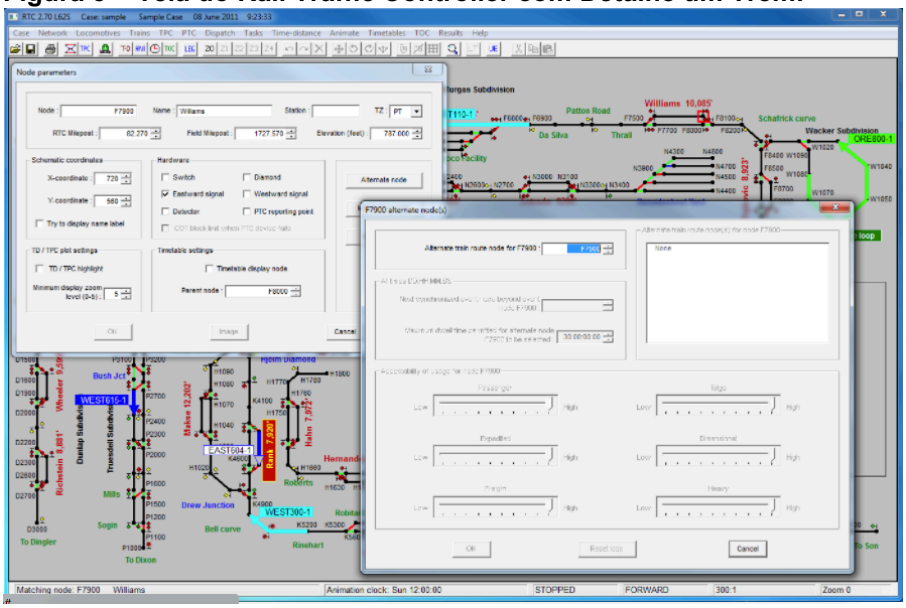
O *software* Rail Traffic Controller, é capaz auxiliar no gerenciamento na malha férrea por meio do controle de custos e de gerenciamento de tráfego nas vias. De forma geral ele é capaz de realizar todas as atividades que um expedidor humano consegue fazer, porém em uma escala maior e por longos períodos. O papel de um

expedidor é receber e transmitir informações de controle de forma que todo o sistema de condução de trens seja organizado (WILLSON, 2012).

O sistema possui uma interface gráfica gerada por meio dos resultados obtidos pela simulação. Suas principais características são as competências de exibição do fluxo dos trens com seus horários de chegada e partida, estimativa de tempo e custo de viagem e informações sobre o trem durante a viagem (WILLSON, 2012).

A Figura 3 mostra uma tela desse sistema. Ao fundo dessa imagem é mostrado graficamente toda a malha férrea com seus desvios e sinalizações. As janelas que estão abertas sobre o mapa são referentes às informações detalhadas de um comboio qualquer.

Figura 3 – Tela do Rail Traffic Controller com Detalhe um Trem.



Fonte: Willson (2012).

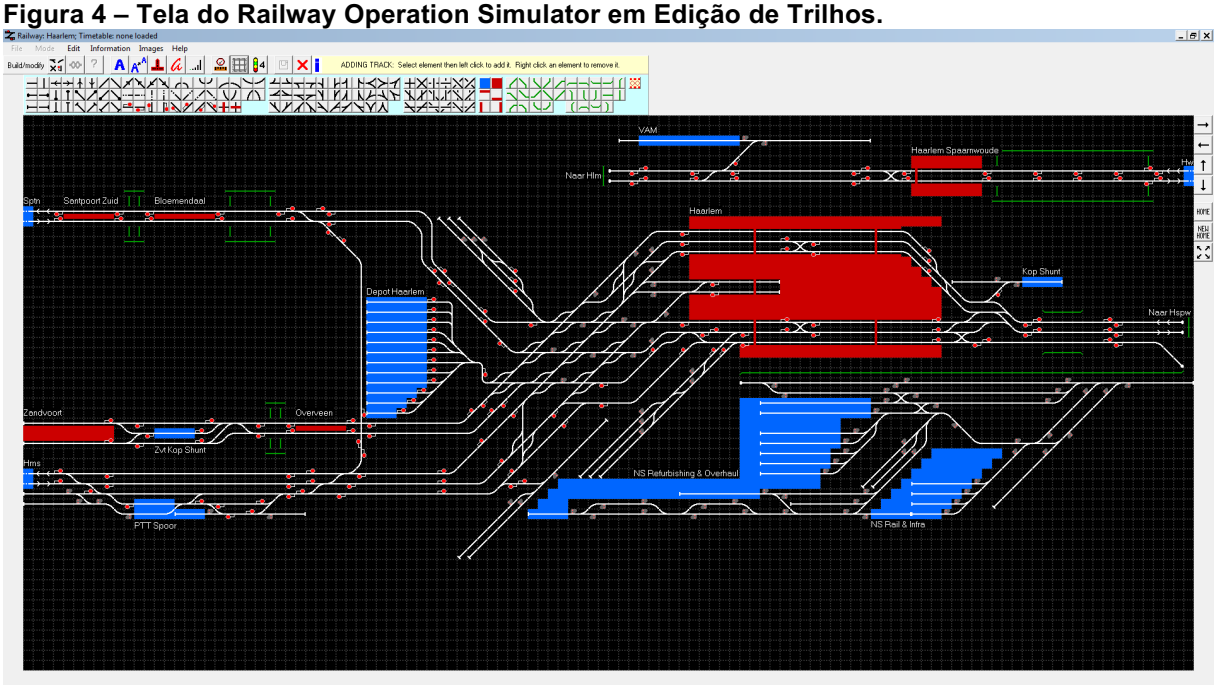
2.1.3 Railway Operation Simulator

O Railway Operation Simulator é um programa desenvolvido para executar no sistema operacional Windows. Diferentemente dos dois softwares apresentados anteriormente, a proposta deste é de oferecer a comunidade de usuários uma plataforma gratuita e adaptativa. Foi desenvolvido na linguagem C++ e seu código é aberto apenas para leitura, porém é possível submeter uma versão ou adaptação para os desenvolvedores (BALL, 2017).

A proposta desse sistema é oferecer um ambiente para aprendizado sobre a condução de trens, construção de ferrovias e também um ambiente para análise dos resultados de viagens por meio de uma planilha gerada pelo sistema (BALL, 2017).

A seção de construção de ferrovias do sistema possui um conceito que pode ser abstraído para este trabalho no momento em que os ambientes serão gerados, pois sua tela oferece ao usuário do programa opções para geração de trilhos, desvios e outras características como mostrado na Figura 4.

Figura 4 – Tela do Railway Operation Simulator em Edição de Trilhos.



Fonte: Ball (2017)

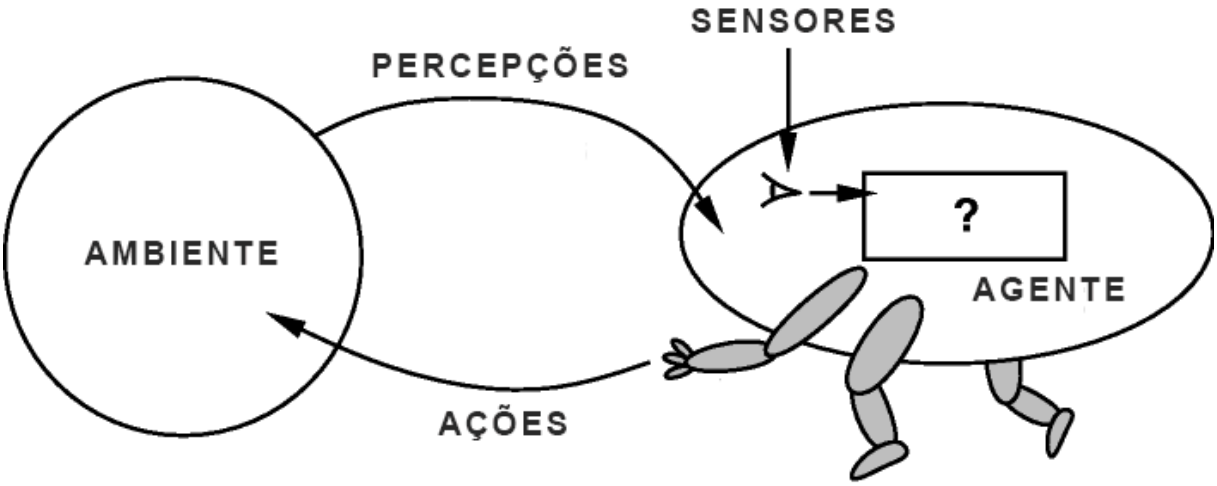
2.2 AGENTES INTELIGENTES

Genericamente um agente é descrito como algo capaz de executar ações com base em estímulos recebidos do ambiente o qual habita. Essa definição se aplica a várias áreas de estudo. Na Computação pode-se representar o agente com um programa qualquer que recebe uma entrada (estímulo) de valores, produza uma saída (ações) que, de alguma forma, interaja com o ambiente (RUSSEL; NORVIG, 2003).

Na Figura 5 apresenta um agente com sensores representados pelo desenho de um olho. Esses sensores captam as informações do ambiente, por meio de percepções, para que o agente tome uma decisão. O ponto de interrogação na imagem representa a parte pensante do agente, a qual define a ação a ser tomada.

Após tomada a decisão, a ação do agente implicará em uma possível mudança do ambiente.

Figura 5 – Exemplo de agente inteligente



Fonte: Adaptado de Russel et. al (2003)

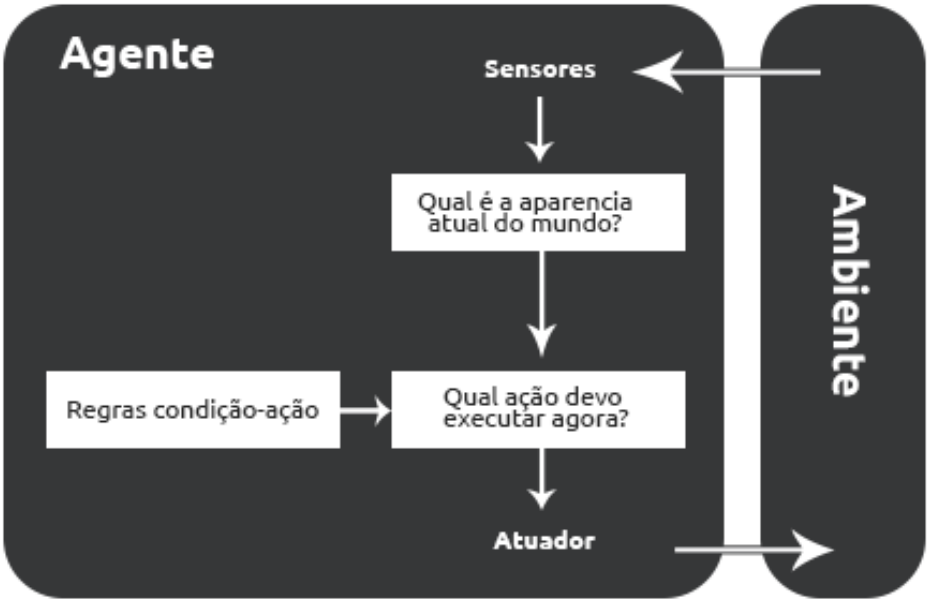
Definir-se um agente é inteligente é uma tarefa complexa. É necessário que, pelo menos, os agentes possuam os conceitos de reatividade, proatividade e habilidade social. A primeira define a capacidade do agente em perceber e responder rapidamente a estímulos provocados pelo ambiente. A segunda caracteriza a capacidade do agente em tomar suas decisões automaticamente no momento certo sem a necessidade de estímulos externos. Por fim, a habilidade social é a capacidade que o agente deve possuir de interagir com o ambiente onde atua e com outros agentes do ambiente (WOOLDRIDGE, 2002).

Para que um agente seja construído é necessário definir uma série de ações e decisões que ele poderá tomar em determinadas situações. A tarefa mais difícil que o agente enfrenta é a de escolher qual ação tomar para determinada situação (WOOLDBRIDGE, 2002). Por conta disso existem diferentes grupos de agentes que incorporam os fundamentos básicos de sistemas inteligentes, descritos a seguir: agentes reativos simples, agentes reativos baseados em modelos, agentes baseados em objetivos e agentes baseados em utilidade e agentes baseados em aprendizagem (RUSSEL et al, 2003).

2.2.1 Agentes Reativos Simples

Tipo mais simples de agentes. Ignoram todo o histórico de ações anteriores, tomando como base para suas ações apenas a percepção ambiente atual. Sua ação é definida com base em uma condição do ambiente, definida com base em um conjunto de regras no formato *condição-ação*, as quais são acionadas em função do estado atual do mundo, conforme mostra a Figura 6 (Russel et al, 2003). É importante notar que um agente isoladamente pode não representar muita importância pelo fato de apenas usar regras do tipo se A, então B. Porém quando analisados em uma comunidade de agentes tornam-se muito significativos, pois suas ações podem influenciar nas ações de outros agentes e assim modificar todo o ambiente.

Figura 6 – Diagrama de Agente Reativo Simples



Fonte: Adaptado de Russel et. al (2003)

Na Figura 6 o agente recebe uma entrada por meio dos sensores informando as percepções do ambiente naquele momento. Com isso, é feita análise do tipo se A então B da entrada e tomada a decisão de qual ação tomar com base nas regras definidas para o agente. Após escolhida a ação, é executada sobre o ambiente por meio de um atuador definido para o agente.

Em Wooldridge (2002), temos um exemplo desse tipo de agente, o qual um termostato é definido da seguinte forma:

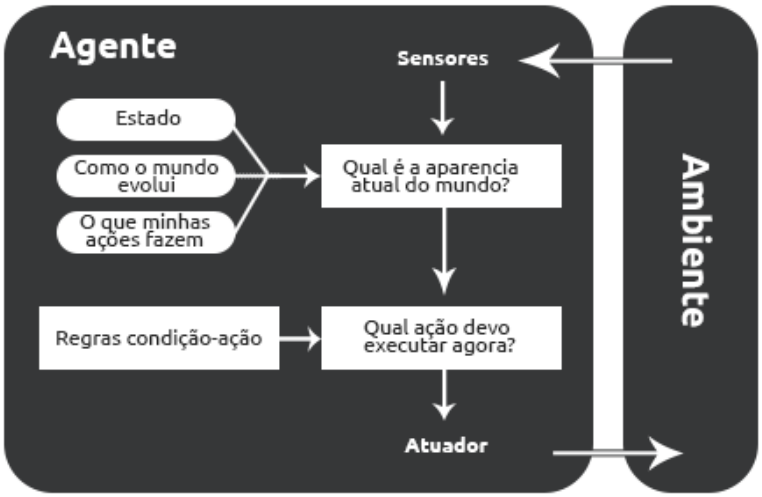
$$\mathrm{Ag(e)} = \begin{cases} \text{desligar aquecedor} & \text{Se e} = \text{temperatura normal} \\ \text{ligar aquecedor} & \text{Caso contrário} \end{cases}$$

Nesse caso, o agente percebe a temperatura do ambiente e verifica qual ação deve ser tomada. Se a temperatura estiver normal o aquecer desliga. Se a temperatura estiver abaixo do normal, o aquecedor é ligado. Essas condições se aplicam supondo que no ambiente existem apenas os estados de temperatura normal ou frio.

2.2.2 Agentes Reativos Baseados em Modelo

Ao contrário do modelo anterior, este tipo de agente utiliza como base as percepções anteriores, mantidas na forma de estados internos capazes de modelar o ambiente, para tomarem suas decisões. Estes estados guardam informações sobre como é o ambiente para auxiliar nas novas percepções (RUSSEL, 2003).

Figura 7 – Diagrama de Agente Reativo Baseado em Modelo



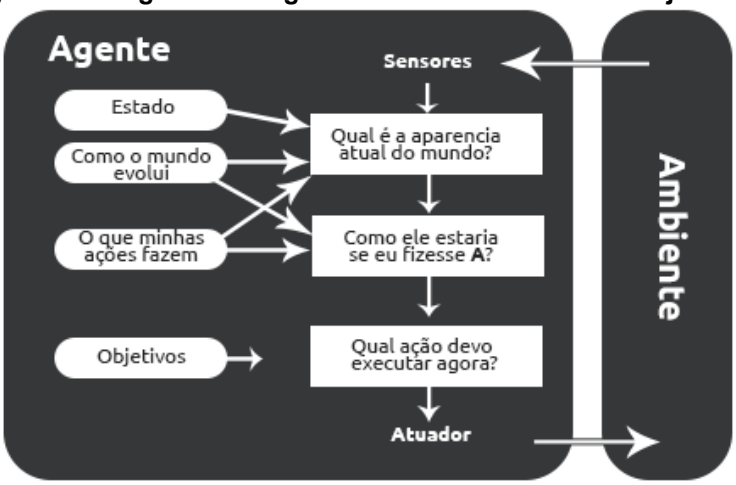
Fonte: Adaptado de Russel et. al (2003)

Na Figura 7, observa-se que o agente recebe a entrada pela percepção de mundo dos sensores assim como no modelo anterior. Na fase de verificar a aparência atual do ambiente é levado em consideração os estados já observados anteriormente para analisar se as ações tomadas mudaram de alguma forma o ambiente. Após feitas as observações é tomada a decisão de ação e executada por atuador do agente da mesma forma que o agente anterior.

2.2.3 Agentes Baseados em Objetivos

Este modelo de agente não utiliza apenas percepções atuais para obter uma resposta. São feitas combinações do resultado desejado com as possíveis tomadas de decisões. Por conta disso, esse tipo de agente tem maior capacidade de planejamento das ações, uma vez que a cada nova percepção recebida suas ações se adaptam da melhor forma para atingir um objetivo (RUSSEL, 2003).

Figura 8 – Diagrama de Agente Reativo Baseado em Objetivos



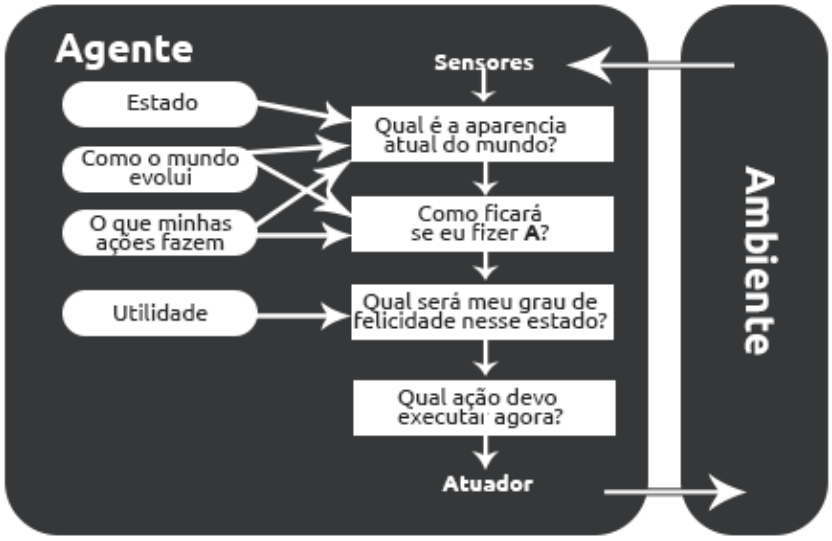
Fonte: Adaptado de Russel et. al (2003)

A Figura 8 mostra uma estrutura muito semelhante com a dos agentes baseados em modelo, porém as considerações de como o mundo evoluiu desde ultima ação e o que as ações do agente influenciaram para a mudança são consideradas na hora da escolha da ação. Após a percepção do mundo atual é feita uma verificação de como o mundo ficará depois de executada uma determinada ação com base nos estados que já foram obtidos anteriormente.

2.2.4 Agentes Baseados em Utilidade

Essa classe de agentes é uma soma da classe anterior com alguns recursos a mais. Além de combinar o resultado desejado com as tomadas de decisões, deve ser levado com consideração outros fatores que podem tornar o resultado mais satisfatório (RUSSEL, 2003).

Figura 9 – Diagrama de Agente Baseado em Utilidade



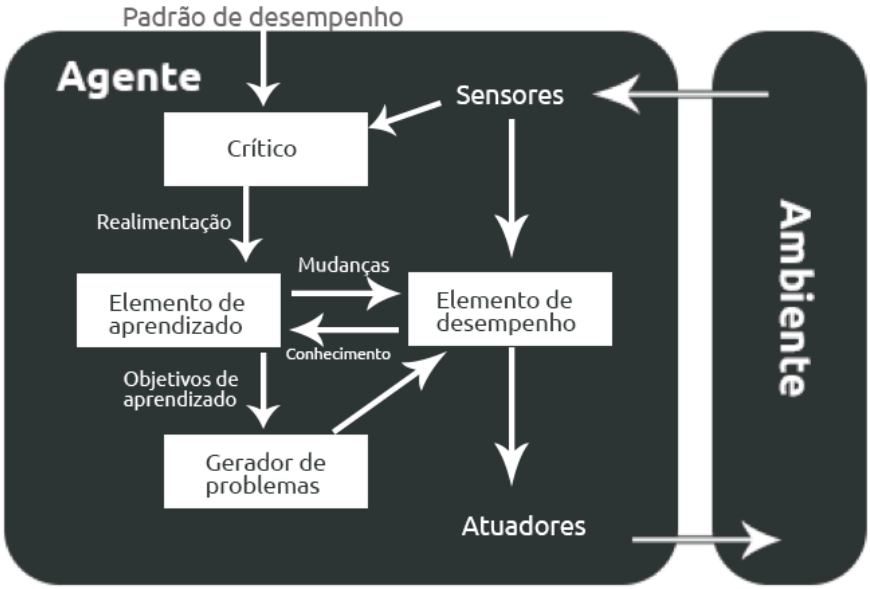
Fonte: Adaptado de Russel et. al (2003)

Na Figura 9 é mostrado que na etapa de verificação do grau de felicidade em um determinado estado escolhido é executada uma função de utilidade para determinar o resultado. Essa função varia dependendo do objetivo do agente. Em Wooldrige (2002) é ilustrado um exemplo de utilidade no mundo real, o qual supõe duas pessoas, uma com nada de dinheiro e outra com quinhentos milhões. Ao doar um milhão para a pessoa que não tem nada, isso faria uma diferença enorme. Porém ao doar um milhão para a pessoa que já tem quinhentos milhões, a diferença do valor anterior e do atual seria mínima. Pontando a utilidade seria muito melhor para a pessoa que não tem nada caso fosse executada a ação de doar um milhão.

2.2.5 Agentes Reativos Baseados em Aprendizado

Essa classe de agente possui quatro componentes principais, sendo eles, elemento de desempenho, elemento de aprendizado, elemento crítico e elemento gerador.

Figura 10 – Diagrama de Agente Baseado em Aprendizado.



Fonte: Adaptado de Russel et. al (2003)

Na Figura 10 é mostrado a ordem como os componentes são utilizados e suas interdependências. O elemento de desempenho é responsável por perceber e tomar as decisões. Já o elemento de aprendizado depende muito do funcionamento do elemento de desempenho, uma vez que o conhecimento obtido vem dele. A função do elemento crítico é fornecer ao agente um *feedback* de como ele está se saindo comparado ao padrão de desempenho fixo. Por ultimo, o elemento gerador de problemas é responsável por sugerir ações que levem o agente à obter novas experiências e com isso aprender muito mais a longo prazo (RUSSEL, 2003).

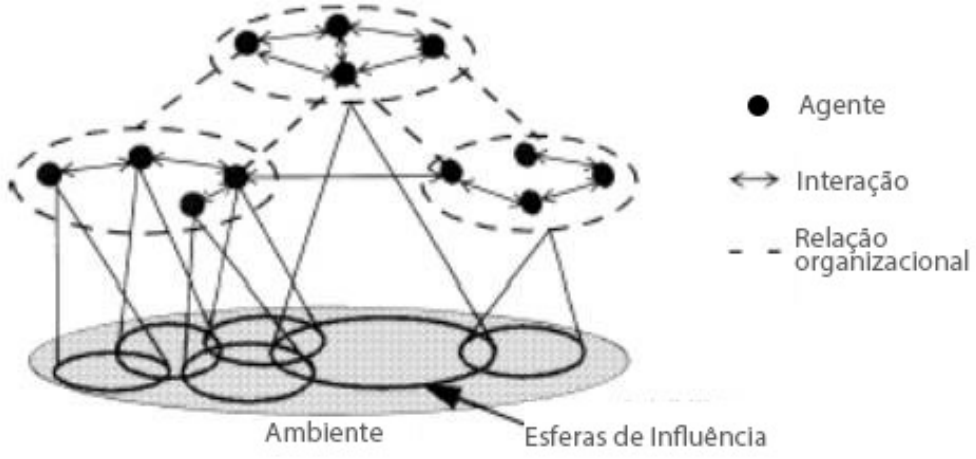
2.3 SISTEMAS MULTI-AGENTES

Para que seja feita a implementação de sistemas que simulem vários agentes, é recomendável usar agentes capazes de interagir em um mesmo ambiente. Para isso, existe a definição de Sistemas Multi-Agentes (SMA).

Em um SMA cada agente possui uma esfera de influencia que informa os recursos que ele pode usar ou modificar do ambiente. Os agentes de um mesmo sistema devem ter a capacidade de influenciar o ambiente de alguma forma, porém, respeitando as regras de interação do ambiente. Segundo as regras, caso, em algum momento, a esfera de influencia coincida com a de outro agente simultaneamente um dos agentes deve ser o predominante para a ação. Para isso isso, podem ser

declaradas regras de hierarquia entre agentes como representadas na Figura 11 (WOOLDRIDGE, 2002).

Figura 11 – Agentes em um ambiente multi-agente.



Fonte: Adaptado de Wooldridge (2002)

Cada ponto preto da Figura 11 representa um agente. Estão conectados a outros agentes por setas, as quais representam a interação entre eles. Na imagem os pontos estão separados em grupos circulados por pontilhados. Esses grupos podem influenciar o ambiente como um todo ou, como no caso dos agentes do grupo da esquerda, cada um pode influenciar individualmente o ambiente. Pode-se notar que as esferas de influencia colidem-se no ambiente e, para isso devem ser definidas as hierarquias dos agentes para que uma ação seja predominante a outra de forma que não gere conflito. Os agentes mais acima estão em um nível predominante na hierarquia com relação aos agentes mais abaixo.

Em um grupo de agentes cada um possui algum objetivo ou preferência de como o ambiente deve ser. Portanto, é necessário que cada resultado que possa ser obtido em um cenário qualquer seja atribuído ao agente junto com a qualidade desse, ou seja, o quão bom será para o agente se esse resultado for gerado. Para isso, usamos funções de utilidade que nos dão esses resultados. Tais funções verificam o grau de satisfação dos estados para cada agente.

2.4 FRAMEWORKS PARA DESENVOLVIMENTO DE AGENTES

A seguir serão apresentados dois frameworks para desenvolvimento de sistemas com agentes, bem como as suas características, linguagem, vantagens e desvantagens.

2.4.1 Java Agent Development Framework

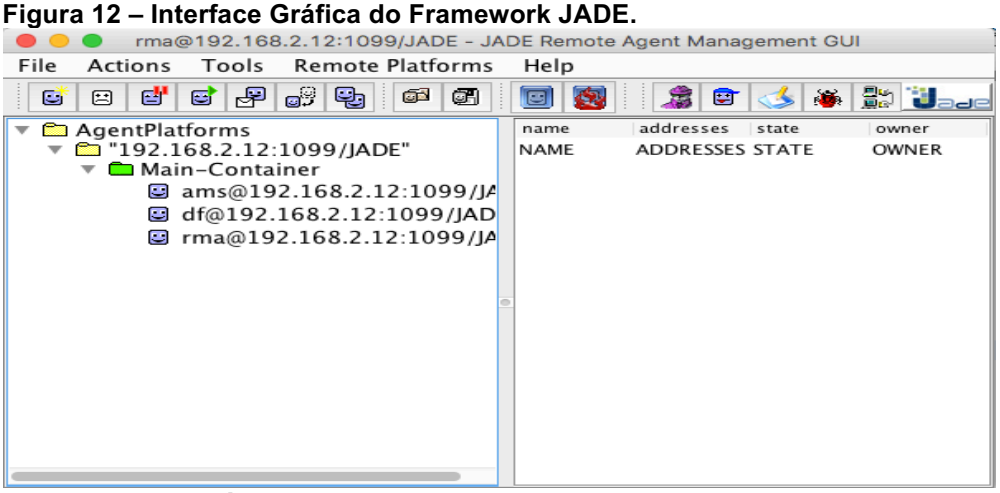
O Java Agent Development Framework (JADE) é um software desenvolvido em Java com código aberto. Segue os padrões da Foundation For Intelligent Physical Agents (FIPA), a qual é uma fundação que desenvolve especificações para a implementação de plataformas de agentes (TELECOM ITALIA, 2017).

O JADE tem como objetivo principal a facilitação na manipulação de agentes, bem como funcionalidades de integração com outros programas. Por ser desenvolvido em Java, é capaz de rodar em praticamente todos os sistemas operacionais no mercado tornando facilmente acessível a sua utilização. Pelo fato dos agentes serem facilmente construídos pelo framework, o desenvolvedor pode focar seus esforços apenas no negócio da aplicação, tornando o desenvolvimento mais ágil (TEIXEIRA, 2010).

O JADE segue quatro princípios principais. A Interoperalidade, a qual um agente deve ser capaz de se comunicar com outro agente fora do ambiente JADE. A Uniformidade e Portabilidade que oferece algumas APIs que são independentes da versão do Java. Facilidade de uso, a qual a complexidade da comunicação com outras plataformas é sustentada por um simples conjunto de APIs. Por fim, JADE emprega a filosofia *pas-you-go*, que possibilita ao programador a escolha do que será utilizado do *middleware,* evitando a sobrecarga de recursos sendo utilizada pelo computador (BELLEIFEMINE, 2003).

Na Figura 12 está representada a tela inicial do JADE.

Figura 12 – Interface Gráfica do Framework JADE.



Fonte: Autoria Própria.

2.4.2 NetLogo

O NetLogo é uma ferramenta de desenvolvimento gratuita utilizada para a simulação de ambientes multi-agentes, capaz de representar sistemas complexos e estruturados na linguagem Java. O *framework* implementa agentes baseados em modelo. Não faz uso dos padrões FIPA. Durante a simulação, o NetLogo representa graficamente o comportamento dos agentes em meio a um ambiente, os quais são denominados *turtles e patches*, respectivamente (TISUE, 2004).

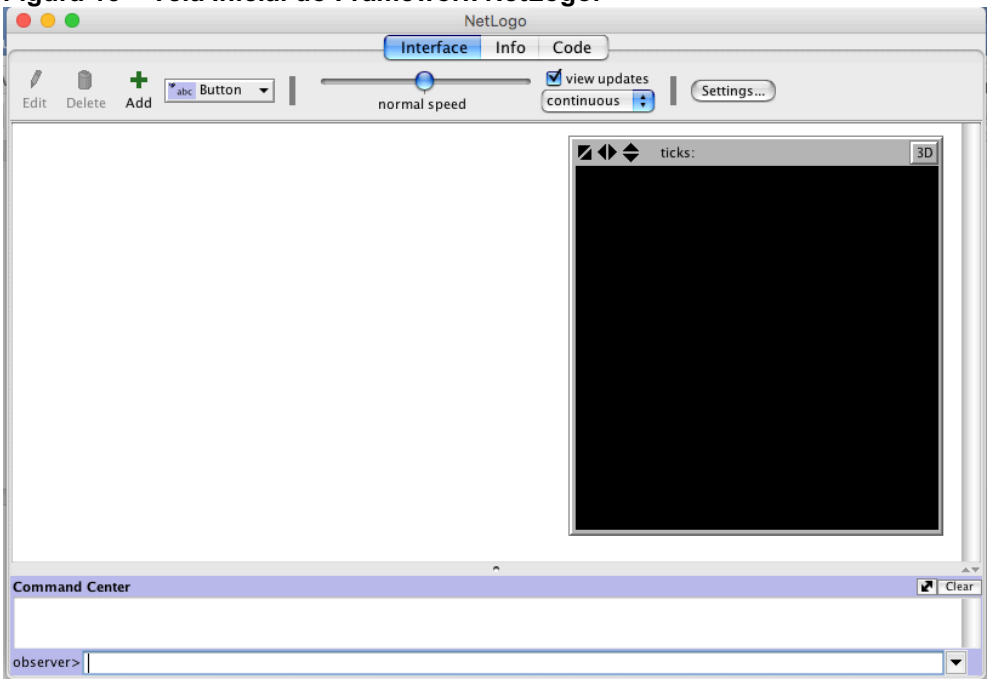
Pelo fato de ser desenvolvido na linguagem Java, o NetLogo é capaz de executar em quase todos os sistemas operacionais disponíveis, além de sua estrutura ser totalmente programável. O framework também permite representar visualizações bidimensionais e tridimensionais, e controlar a velocidade de execução do ambiente (NETLOGO, 2016).

Sua estrutura pode ser incorporada a outros aplicativos, sendo possível a adição de novos recursos ao NetLogo (NETLOGO, 2016). Este conjunto de características permitirá, por exemplo, a integração com o softwares que possuem diferentes regras de condução de trens.

A interface do NetLogo, ilustrada na Figura 13, é simples e intuitiva. Inicialmente o programa carrega a tela principal (Interface), nessa tela será exibido o ambiente de simulação, representado pelo quadro preto, o qual os agentes podem interagir. Nessa tela também é possível configurar os itens de interface de entrada do programa, como botões e textos. Na barra superior é possível controlar a velocidade de simulação, adicionar novos comandos de entrada e fazer ajustes de *layout* da tela de simulação. Na barra inferior é possível executar ações previamente definidas por

meio de linhas de comando.

Figura 13 – Tela Inicial do Framework NetLogo.

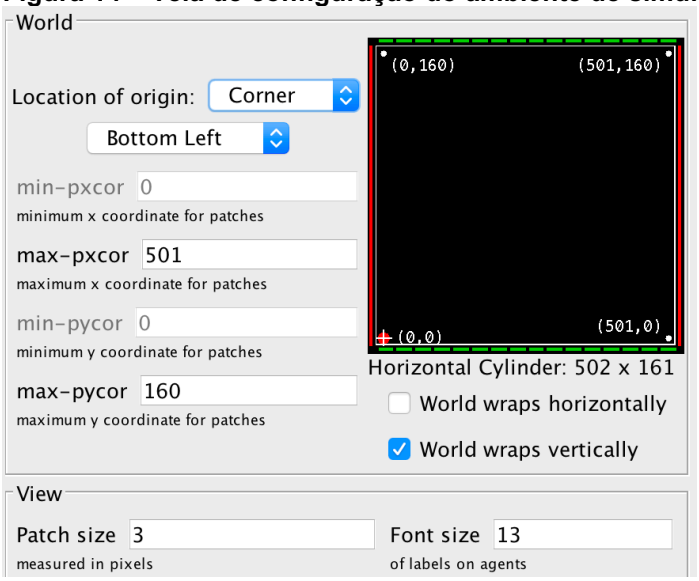


Fonte: Autoria Própria.

Existem nove tipos de itens de interface, sendo eles: *Button*, responsável por capturar um evento de clique do mouse. *Slider*, cria uma linha com um intervalo de valores predefinidos e arrastável pelo *mouse*. *Switch,* retorna um valor verdadeiro se estiver selecionado *On* e falso se estiver selecionado *Off. Chooser,* responsável por criar uma lista de valores predefinidos para seleção. *Input,* capaz de receber um valor de entrada digitado pelo teclado. *Monitor,* responsável por monitorar algum valor de variável definido. *Plot,* cria um gráfico para monitoramento de variáveis do programa. *Output,* responsável por criar um campo para exibir algum valor definido no código, sendo possível a criação de apenas um campo. E por fim, *Note,* responsável por criar uma área de texto livre.

O ambiente de simulação utiliza coordenadas *x* e *y* para construir o universo dos agentes. Sendo assim, é possível ajustar o tamanho de várias formas como mostra a Figura 14.

Figura 14 – Tela de configuração do ambiente de simulação Netlogo

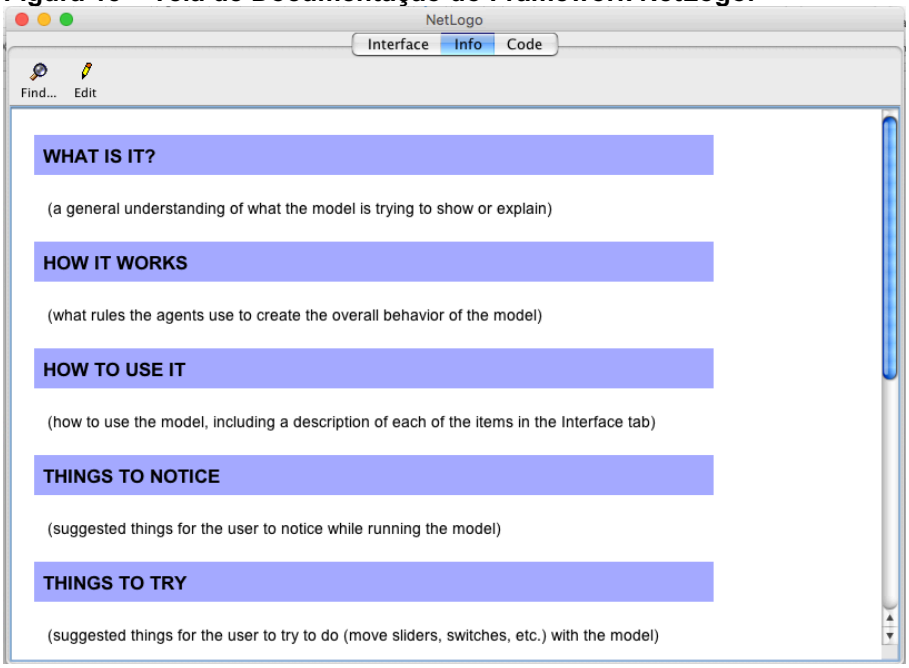


Fonte: Autoria Própria.

A configuração *Location of origin* possibilita definir qual será o ponto de origem, ou seja, o ponto em que o eixo x e o eixo y são iguais a zero. Para este trabalho foram utilizadas as coordenas iniciando no canto inferior esquerdo, assim como mostra a Figura 14. As configurações de *max-pxcor* e *max-pycor* definem o tamanho da tela de simulação em pontos, onde cada ponto pode ter 1 pixel ou mais, definido logo abaixo em *Patch-size.* Para este trabalho foi utilizado o tamanho de eixo y com 160 pontos e o eixo x foi definido como estático, pois é necessário uma adaptação do tamanho de ambiente para qualquer tamanho de malha férrea.

Na segunda tela do NetLogo (Info), representada na Figura 15, é definida a documentação programa desenvolvido, onde fica a critério do desenvolvedor a forma como é escrita. O framework possui por padrão uma estrutura que serve como sugestão de como deve ser escrita a documentação. Porém ela poderá ser alterada clicando no botão de editar da barra superior.

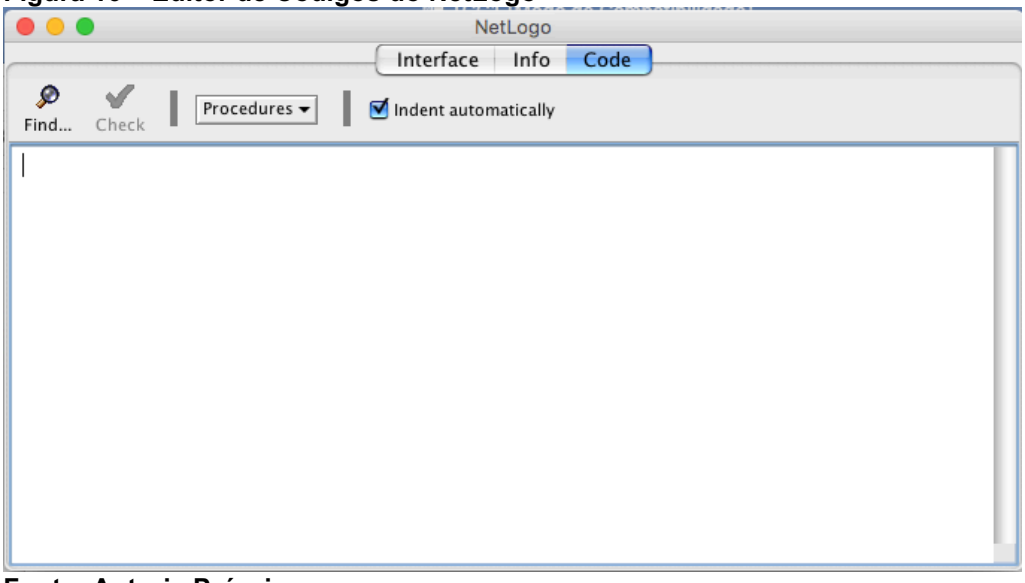
Figura 15 – Tela de Documentação do Framework NetLogo.



Fonte: Autoria Própria.

A terceira e última aba do Netlogo é onde todo o código do programa será declarado. Está tela possui um editor de código, conforme mostrado na Figura 16. A barra superior possui a opção de identação automática e busca rápida pelas funções declaradas no código.

Figura 16 – Editor de Códigos do NetLogo



Fonte: Autoria Própria.

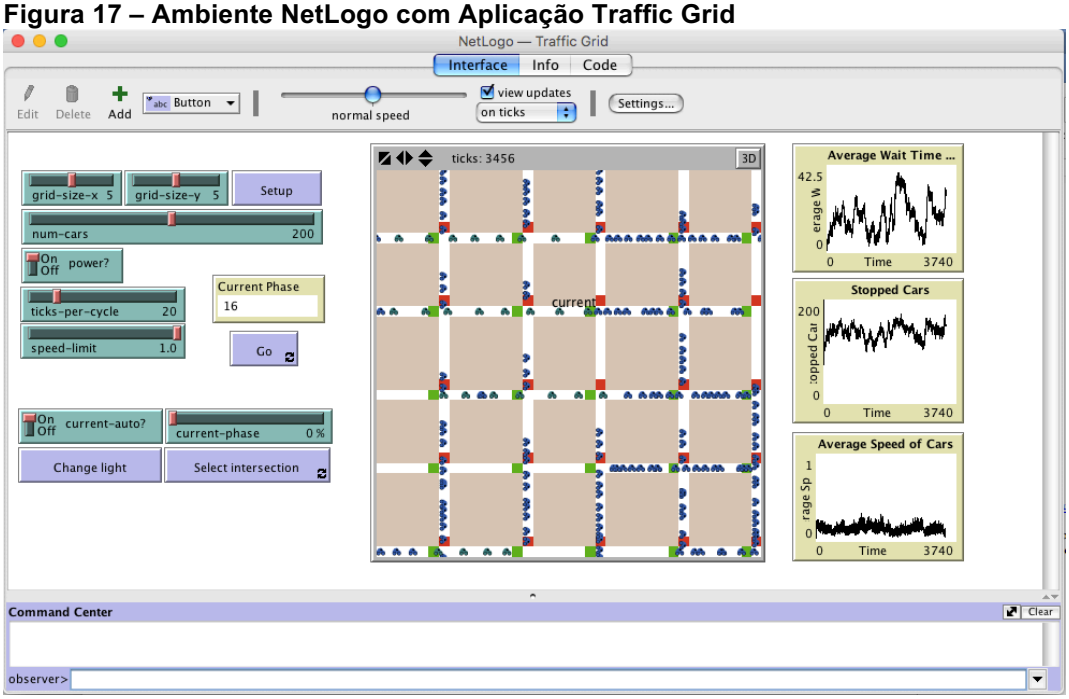
O NetLogo possui por padrão alguns modelos já implementados para consulta e aprendizado. A seguir serão apresentados dois destes modelos que se destacam com mais relevância para esse trabalho por serem próximos ao domínio da aplicação.

2.4.2.1 Modelo Traffic Grid do NetLogo

O Modelo Traffic Grid representa uma cidade na forma de grade e permite controlar os semáforos, limites de velocidade e quantidade de veículos por meio de variáveis globais predefinidas.

A cada cruzamento existem semáforos para as duas vias e cada via possui um único sentido, conforme é possível visualizar na Figura 17.

Figura 17 – Ambiente NetLogo com Aplicação Traffic Grid



Fonte: Wilensky (2003).

Os veículos aceleram quando estão abaixo da velocidade máxima e a mantém quando atingem até que seja necessária uma parada por conter um veículo a frente ou por um semáforo no estado fechado (WILENSKY, 2003).

Os veículos podem trafegar na vertical e na horizontal, portanto existe uma função que estabelece randomicamente qual será o sentido do veículo com base no número de agentes já ocupando a mesma via. Também é estabelecido o *setup* dos semáforos, pois quando agentes no sentido horizontal estiverem transitando, os do

sentido vertical devem aguardar a liberação do semáforo e vice-versa. (WILENSKY, 2003).

Para a execução do programa, é necessário executar o comando *setup* que gera os agentes em posições aleatórias nas linhas que representam as ruas. Para iniciar a simulação é utilizado o comando *go.* Durante a execução é possível observar a movimentação dos veículos e três gráficos que mostram a velocidade média dos veículos, o tempo de espera e a quantidade de veículos parados em determinado momento. Foram declarados alguns comandos que podem alterar a entrada para o programa como a quantidade de ruas na vertical e horizontal, quantidade de veículos, opção de desabilitar semáforos, tempo de mudança do semáforo, velocidade limite dos veículos e uma opção para o usuário controlar a mudança de um semáforo escolhido.

A seguir, na Figura 18, está representado o código da função *setup*. Essa função, tem por objetivo inicializar todos os componentes da simulação. Primeiramente é usado um comando da linha 2 para limpar todos os dados da simulação, caso tenha sido executado algo antes. Após isso, são inicializadas as variáveis globais e o ambiente de simulação por meio da chamada das funções *setup-globals* e *setup-patches* nas linhas 3 e 4, respectivamente. As variáveis globais possuem os valores de entrada do programa, como o número de carros e a quantidade de ruas. A função para geração do ambiente de simulação utiliza os recursos do NetLogo para colorir de forma correta o mapa para que cada elemento fique em sua posição correta. Com essa função também são inicializados os semáforos. Para que a quantidade de veículos não seja maior do que o espaço disponível é verificado se a quantidade de carros é menor que o espaço nas ruas, como mostrada a linha 9, e caso não seja possível inserir os veículos no ambiente o programa dispara uma mensagem e é parado com os comandos das linhas 11 e 17, respectivamente. Por fim, após criado o ambiente, são gerados os veículos por meio da função nativa do NetLogo *create-turtles*. Essa função permite inserir os atributos ao agente como mostrado da linha 22 até 24. Também são definidas as velocidades do veículos na linha 17.

Figura 18 – Código de Inicialização da Aplicação Traffic Grid do NetLogo

```
to setup
    clear-all
    setup-globals
    setup-patches
    make-current one-of intersections
    label-current
    set-default-shape turtles "car"

    if(num-cars > count roads)
    [
      user-message (word "There are too many cars for the amount of"
                         "road. Either increase the amount of roads "
                         "by increaing the GRID-SIZE-X or "
                         "GRID-SIZE_Y sliders, or decrease the"
                         "number of cars by lowering the NUMBER slider.\n"
                         "The setup has atopped." )
      stop
    ]

    create-turtles num-cars
    [
        setup-cars
        setup-car-color
        record-data
    ]

    ask turtles [ set-car-speed ]

    reset-ticks
end
```

Fonte: Wilensky (2003).

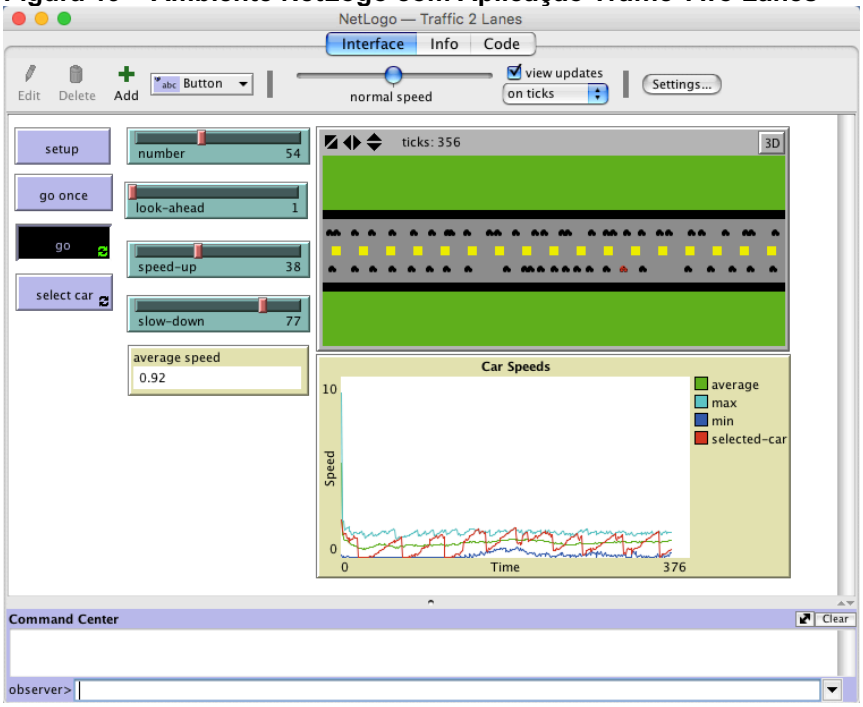
2.4.2.2 Modelo Traffic Two Lanes do NetLogo

O Modelo Traffic Two Lines possui uma via com duas faixas para que os veículos tenham a possibilidade de alterná-las, visando diminuir o congestionamento formado. O ambiente de simulação oferece o controle do número de veículos, a proximidade máxima entre dois veículos e o controle de velocidade (WILENSKY, 1998).

No *setup* dos agentes é sorteado a via em que cada um iniciaria a simulação. São definidas duas vias, porém os veículos podem transitar entre elas durante a passagem de uma para outra. Portanto, o espaçamento entre as vias deve comportar veículos. Todos os agentes possuem uma função *drive* que estabelece a velocidade e o controle contra colisões. Nessa função há a possibilidade de o veículo transitar de uma via para outra definida randomicamente (WILENSKY, 1998).

O modelo e controles da simulação podem ser observados na Figura 19, onde é possível observar as variáveis de controle usadas na simulação, como o número de veículos (*number*), a velocidade(*avarege speed*) e a distancia mínima entre os veículos.

Figura 19 – Ambiente NetLogo com Aplicação Traffic Two Lanes



Fonte: Wilensky (1998).

Na Figura 20 é mostrado o código para definição da função de *setup* do programa *Traffic Two Lanes.* Assim como no modelo anterior todas as configurações de inicialização do ambiente e dos agentes são executadas nessa função. Na linha 3 é feita uma chamada da função *draw-road*, a qual executa todos os comandos para inicialização do ambiente. Essa função é equivalente a função *setup-patches* do modelo *Traffic Grid*. O NetLogo possui algumas imagens para representar os agentes por padrão. Na linha 4 é mostrado que todos os agentes serão definidos usando a imagem de um carro. Na linha seguinte são criados os agentes em si com base na quantidade dada como parâmetro. Na linhas seguintes (6 e 7) é feita uma seleção aleatória de algum agente e definido a cor como vermelha para facilitar o acompanhamento visual desse agente se locomovendo.

Figura 20 – Código de Inicialização da Aplicação Traffic Two Lanes do NetLogo

```
1  to setup
2      clear-all
3      draw-road
4      set-default-shape turtles "car"
5      create-turtles number [ setup-cars ]
6      set selected-car one-of turtles
7      ask selected-cart [ set color red ]
8      reset ticks
9  end
```

Fonte: Wilensky (1998).

2.4.3 NetLogo e JADE

Existem outros frameworks no mercado para implementação de sistemas multi-agentes. O JADE e NetLogo possuem algumas semelhanças, como a linguagem em que foi desenvolvido e o código aberto. O JADE foi apresentado pois seu uso é muito significativo pela comunidade de desenvolvedores. Porém a escolha do NetLogo para o desenvolvimento desse trabalho se faz pelo fato de possuir o foco no desenvolvimento de simulações, pois sua interface apresenta nativamente um espaço gráfico para representação do ambiente e dos agentes.

2.5 TRABALHOS RELACIONADOS

A seguir, na Figura 21, será apresentado um diagrama de Venn para relacionar projetos e trabalhos de pesquisa semelhantes com este.

Figura 21 – Diagrama de Venn para Trabalhos Relacionados

Fonte: Autoria Própria.

Sistema de Visualização e Agentes (A) – Em um sistema que faz utilização do NetLogo, no artigo de Jerry et. al (2015) é proposto um novo sistema para controle de tráfego de veículos, onde diferentemente de outros sistemas, os automóveis não se comunicarão com um sistema central para receber coordenadas. Eles atuarão como agentes comunicando-se entre si, com isso, evitando tanto a sobrecarga de um servidor central como os atrasos dos comandos por conta da distância. O mesmo artigo mostra a implementação do sistema, tanto utilizando a técnica *Ant Colony Optimization* (ACO) que trata de um algoritmo que procura caminhos em grafos

inspirado no comportamento das formigas em busca de alimento, como não utilizando. O sistema gera uma matriz dimensão *n* definida pelo usuário. Também é possível definir a quantidade de veículos (agentes) transitando durante a simulação.

Agentes e Condução de Veículos (B) - Em artigo Dordal et. al (2001) propõem uma solução para o desperdício de combustível das locomotivas utilizando sistemas multi-agentes para a condução. Como o maior desperdício de combustível é gerado quando há necessidade de parada do trem, o sistema proposto utiliza cruzamentos para que possa ser feita a ultrapassagem. Para o desenvolvimento foram utilizados três tipos de agentes: (i) agentes de condução, especialistas em controle dos trens; (ii) agentes de licença, responsáveis pela gestão dos trechos com pelo menos um cruzamento e (iii) agente de ambiente, capaz de controlar a ferrovia. Todo esse sistema contribui, além da redução do consumo de combustível, para o aumento da quantidade de trens trafegando ao mesmo tempo, redução na duração da viagem, segurança e redução de poluentes gerados por meio da queima de combustível.

Sistema de Visualização e Condução de Veículos (C) – Em Weigang, Li et al (2001) é apresentado o desenvolvimento de um sistema para rastreamento e visualização do posicionamento de ônibus. O intuito do projeto foi oferecer a comunidade de usuários do transporte coletivo uma maneira de informar os horários dos ônibus de forma dinâmica e em tempo real. O sistema também traz benefícios aos administradores do serviço transporte, pois gera mais controle sobre os veículos que estão trafegando. Para o desenvolvimento foi utilizado o Serviço de Posicionamento Global (GPS), sistema de informações geográficas, serviços relacionados a bando de dados e serviços de telefonia para comunicação com os veículos. Para os usuários do sistema foi disponibilizada uma página web com informações sobre o posicionamento. Outras formas de obtenção de informação sobre o rastreamento foram desenvolvidas. Uma delas mostra em forma de texto informações como velocidade, local, número da linha e posição do veículo em um sistema local. A outra mostra em forma gráfica o posicionamento do veiculo em um mapa.

Agentes, Sistema de Visualização e Condução de Veículos (D) - Em pesquisa relacionada gestão de transportes, Silveira e Pasin (2014) apresentam uma solução para a gestão de veículos utilizando um sistema multi-agentes. O objetivo é facilitar o entendimento dos problemas de transporte urbano por meio de simulação. Foi utilizada a ferramenta de Siafu, de licença gratuita, escrita em Java e capaz de

modelar sistemas multi-agentes. Durante a implementação foram utilizados mapas reais com suas informações definidas. Cada veículo possui um agente atuando como motorista com comportamento único. Por fim, este projeto foi capaz de oferecer um meio para a detecção de comboios, geração de relatórios e informações sobre os veículos atrasados ou sobrecarregados.

3 DESENVOLVIMENTO

Neste capitulo será demonstrado o processo de desenvolvimento do ambiente de simulação, bem como os recursos utilizados por meio do framework NetLogo e suas particularidades. Inicialmente será apresentado como os elementos que compõe a malha férrea foram tratados no NetLogo. Também serão mostrados os funcionamentos dos componentes de interação e acompanhamento de um determinado trem de amostra. Por fim, será mostrado a forma de locomoção do trem na malha férrea.

3.1 ELEMENTOS QUE COMPÕE A MALHA FÉRREA

As informações de uma malha férrea são representadas por meio de um arquivo em formato *xml*, o qual contém informações de vários pontos da via. Para o desenvolvimento desse projeto, o arquivo da malha utilizado possui informações sobre a malha fornecidos a cada 20 metros de distância.

A Figura 22 demonstra o arquivo *xml* com seu cabeçalho e o exemplo de um determinado ponto da via com suas respectivas informações. As marcações dos campos *identificador*, *distanciaPonto*, *bitolaLinha* e *velocidadeMedia* representam, respectivamente, o nome da via, e distância entre uma marcação e outra em metros, o valor da bitola (distância entre os dois trilhos em metros) e a velocidade média da via. As marcações *pontoDeMedida* possuem algumas informações sobre o ponto da via em questão. As informações relevantes para ilustrar o posicionamento do trem na via são: a quilometragem atual (representada por *km*), a inclinação da via (representada por *rampa*) e o raio da curva (representado por *raioCurva*). As demais informações são usadas nos cálculos de deslocamento, como, ângulo central (ac), grau 20 (g20), altitude (altitude) e localização geográfica (localização).

Figura 22 – Arquivo xml com informações sobre a malha férrea

```xml
<viaFerrea>
    <identificador>Preencher nome</identificador>
    <distanciaPonto>20</distanciaPonto>
    <bitolaLinha>1.60</bitolaLinha>
    <velocidadeMedia>100</velocidadeMedia>
    <listaDePontosDeMedida>
        <pontoDeMedida>
            <!-- Pasta de origem rampa = km204-->
            <!-- Pasta de origem curva = km204-->
            <id>1</id>
            <velocidadeMax>60</velocidadeMax>
            <km>204</km>
            <rampa ini="0.0" fim="20.0">-0.98642856</rampa>
            <raioCurva ini="0.0" fim="20.0">0.0</raioCurva>
            <ac ini="0.0" fim="20.0">0.0</ac>
            <g20 ini="0.0" fim="20.0">0.0</g20>
            <altitude ini="0.0" fim="20.0">527.573</altitude>
            <localizacao>
                <latitude>0</latitude>
                <longitude>0</longitude>
            </localizacao>
        </pontoDeMedida>
        .
        .
        .
    </listaDePontosDeMedida>
</viaFerrea>
```

Fonte: Borges (2015)

Para que esta via seja mostrada, primeiramente é necessário que o arquivo seja lido pelo framework Netlogo. Como não há suporte para a leitura desse tipo de arquivo de modo nativo no NetLogo, fez-se necessária a conversão dos dados do arquivo *xml* para um arquivo texto. Nele, cada ponto da malha é representado em uma linha com seus respectivos valores separados por um espaço. Dessa forma, é possível ler o arquivo e armazenar cada linha em uma lista. A Figura 23 mostra um exemplo do ponto de medida *xml* convertido para o formato texto.

Figura 23 – Comparação de um ponto da via em xml e texto.

ARQUIVO XML

```xml
<pontoDeMedida>
    <!-- Pasta de origem rampa = km204-->
    <!-- Pasta de origem curva = km204-->
    <id>1</id>
    <velocidadeMax>60</velocidadeMax>
    <km>204</km>
    <rampa ini="0.0" fim="20.0">-0.98642856</rampa>
    <raioCurva ini="0.0" fim="20.0">0.0</raioCurva>
    <ac ini="0.0" fim="20.0">0.0</ac>
    <g20 ini="0.0" fim="20.0">0.0</g20>
    <altitude ini="0.0" fim="20.0">527.573</altitude>
    <localizacao>
       <latitude>0</latitude>
       <longitude>0</longitude>
    </localizacao>
</pontoDeMedida>
```

ARQUIVO TEXTO

```
1 80 1 60 204 -0.98642856 0.0 527.573
```

Fonte: Autoria Própria

A conversão do arquivo para texto pode ser feita em qualquer linguagem, pois o resultado obtido deve ser o mesmo. Para este trabalho foi utilizado um script na linguagem PHP, mostrado no Algoritmo 1.

Algoritmo 1 – Script PHP para conversão de arquivo xml em texto

```
01  <?php
02      $eixo_x = 1;
03      $eixo_y = 80;
04      $xml = simplexml_load_file('via01.xml');
05      $resultado = fopen("via.txt", "w");
06      foreach($xml->listaDePontosDeMedida->pontoDeMedida as $registro):
07
08      $linha =
09             $eixo_x . " " . $eixo_y . " " .
10             $registro->id .    " " .
11             $registro->velocidadeMax . " " .
12             $registro->km . " " . $registro->rampa . " " .
13             $registro->raioCurva . " " .
14              $registro->altitude . "\n";
15      fwrite($resultado, $linha);
16      $eixo_x++;
18      endforeach;
19      fclose($resultado);
20  ?>
```

Fonte: Autoria Própria

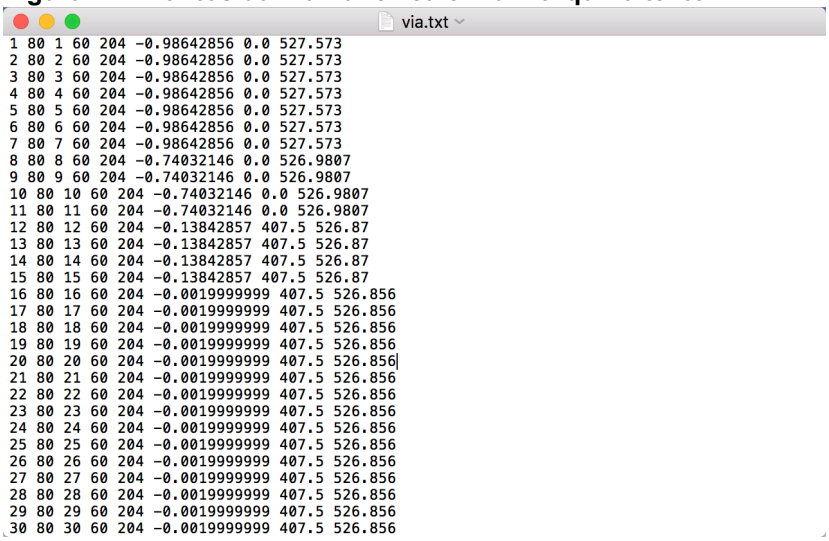
Nas linhas 2 e 3 foram declaradas as variáveis dos eixos *x* e *y*, respectivamente, do ambiente de simulação. O eixo *x* inicia em um, pois será o primeiro ponto da malha férrea. Já o eixo *y* deve ser sempre 80 para que a malha férrea seja exibida verticalmente ao meio do ambiente de simulação.

A variável *$xml* utiliza uma função para abertura do arquivo *xml* com o conteúdo da malha férrea. A variável *$resultado* será utilizada para armazenar os pontos da malha no arquivo texto.

Com as variáveis definidas, um laço de repetição percorre todos os pontos do arquivo *xml* e armazena em uma variável denominada $linha. Essa variável é responsável por armazenar os valores um ponto da via para ser inserida no arquivo de saída.

A Figura 24 mostra os pontos iniciais da via como resultado obtido por meio da execução do *script* desenvolvido, no seguinte formato: número do ponto no eixo x, número do ponto no eixo y, id, velocidade máxima no ponto, quilômetro atual, inclinação, raio da curva e altitude.

Figura 24 –Pontos da malha férrea em um arquivo texto



Fonte: Autoria Própria

Para que esses valores sejam passados aos *patches* do NetLogo, foi necessário a criação de variáveis para cada entrada. O Algoritmo 2 mostra a declaração dessas variáveis. O procedimento *patches-own* é padrão do NetLogo para declarações de variáveis que podem ser utilizadas por qualquer ponto de ambiente na simulação.

Algoritmo 2 –Declaração das variáveis de ambiente no NetLogo

```
01 patches-own [
02     id
03     inclinacao
04     semaforo
05     raio_da_curva
06     altitude
07     velocidade_permitida
08     desvio_ativo
09     desvio_id
10     inicio_desvio
11     curva1_desvio
12     curva2_desvio
13     parada_desvio
14     fim_desvio
15     fechamento_semaforo
16 ]
```

Fonte: Autoria Própria

A seguir é mostrado como é feita a leitura do arquivo, armazenagem dos valores nas variáveis de ambiente e montagem gráfica da via no NetLogo.

3.1.1 Malha Férrea Principal

A malha férrea principal é a via a qual o trem pode transitar e ler as informações contidas em cada ponto, mostrados na seção anterior. Os valores de cada ponto da via a ser construída são obtidos do arquivo texto citado anteriormente. O procedimento que realiza a montagem gráfica da via principal no ambiente de simulação é representado no Algoritmo 3.

Algoritmo 3 – Procedimento de plotagem da malha férrea – via principal

```
01 to setup-malha
02     let largura tamanho_malha
03     resize-world 0 largura 0 160
04     file-open "via.txt"
05     let id_desvio 1
06     set quilometro  -1
07     while [not file-at-end?]
08     [
09         let items read-from-string (word "["file-read-line "]")
10         ifelse (item 0 items != -1)
11         [
12             ask patches with [pxcor = item 0 items and pycor = item 1
13 items]
14 
15             [
16                if quilometro  != item 4 items [
17                    set plabel word quilometro  "KM"
18                    set quilometro  item 4 items
19                ]
20                if (round(item 5 items) = 0)
21                [
22                   set pcolor gray
23                ]
24                if (round(item 5 items) > 0)
25                [
26                    set pcolor yellow
27                ]
28                if (round(item 5 items) < 0)
29                [
30                    set pcolor sky
31                ]
32                set inclinacao item 5 items
33             ]
34         ]
35 
36     [
37         ;Caso a linha seja de desvio
38     ]
39 end
```

Fonte: Autoria Própria

O primeiro parâmetro que deve ser verificado antes de iniciar a plotagem da malha férrea é o tamanho do ambiente definido por *tamanho_malha*, que é o espaço

disponível para a inserção de elementos de simulação, como agentes e pontos da via. Para isso, foi inserido na interface um *input* que recebe como entrada um número inteiro. Este número é a quantidade de pontos que a malha férrea possui, com exceção dos desvios. Na função *setup-malha* representada no Algoritmo 3, é definida a variável local do tamanho da malha na linha 02 com o nome de largura. O valor a ser atribuído à variável largura é originado do *input* citado acima, que possui o nome *tamanho_malha*. Com o valor definido, é utilizado o comando *resize-world,* na linha 03*,* para redefinir o tamanho do ambiente com base no valor de entrada.

Com a definição do tamanho do ambiente de simulação, é executado o comando *file-open* de abertura do arquivo texto com os dados da via. Após isso, duas variáveis recebem valores antes do início da leitura do arquivo. A variável local *id_desvio* define, caso exista, o id do primeiro desvio da malha que será demonstrado na próxima seção deste trabalho. A variável global *quilometro* é responsável por receber o valor da quilometragem atual da via. Como no início da execução nenhuma linha do arquivo foi lida, a variável recebe um valor simbólico -1, definida por *quilometro.*

Na linha 06 do Algoritmo 2 é iniciado o laço de repetição que percorre todo o arquivo da via. Cada iteração do laço recebe uma linha do arquivo texto. Então, é criada uma variável local com o nome de *items* responsável por armazenar uma lista. Para que a linha seja convertida em uma lista, os valores precisar estar entre colchetes, por conta disso é feita uma concatenação dos colchetes com a linha que está sendo lida por meio do comando *word.*

A cada iteração do laço de repetição é verificado se os dados são referentes a malha férrea principal ou desvio. Nos casos em que o primeiro valor da lista é diferente de -1, os dados referem-se a via principal. O valor -1 foi estipulado neste trabalho para diferenciar o trecho principal de um desvio. Sendo assim, são feitas algumas verificações antes de plotar o ponto no ambiente gráfico.

A primeira verificação, na linha 15, valida se o *label* de quilometragem já foi inserido utilizado a variável *quilometro* citada anteriormente. A segunda, terceira e quarta verificação (linhas 19, 23 e 27), são responsáveis por escolher a cor da via com base na inclinação atual. Isso é feito por meio de um arredondamento do valor da inclinação utilizando o comando *round*. Nos casos em que os valores são arredondados para zero, a malha recebe cor cinza (*gray*). Quando são arredondados para valores menores e maiores que zero, recebem cor azul (*sky*) e amarelo (*yellow*),

respectivamente. Por fim, na linha 29 do Algoritmo 2, a variável *inclinação* do *patch* atual recebe o valor que se encontra no sexto elemento da lista.

3.1.2 Manipulação dos Desvios

Os desvios não são representados no arquivo *xml*, uma vez que não foram mapeados, portanto devem ser inseridos manualmente. Para isso, é necessário que o primeiro valor da linha seja -1. Com isso, a verificação da linha 09 do Algoritmo 3 deve ser falsa. Dessa forma, é executado o trecho de código do Algoritmo 4.

Algoritmo 4 – Procedimento de plotagem da malha férrea – desvios

```
01 to setup-malha
02   [
03         let tamanho_desvio (item 2 items - item 1 items)
04         let tamanho_desvio_s_c (tamanho_desvio - 8)
05         let metade_desvio (tamanho_desvio_s_c / 2)
06         let percorre_desvio 0
07         let percorre_curva 0
08         let eixo_x item 1 items
09         let eixo_y 80
10         ask patches with [pxcor = eixo_x and pycor = eixo_y]
11         [
12             set inicio_desvio true
13         ]
14         ask patches with  [pxcor = (eixo_x + 5) and pycor = (eixo_y +
15 5)]
16           [
17              set curva1_desvio true
18         ]
19         ask patches with [pxcor = (eixo_x + tamanho_desvio - 3) and
20 pycor = (eixo_y + 5) ]
21        [
22            set curva2_desvio true
23        ]
24        ask patches with [ pxcor = (eixo_x + tamanho_desvio + 2) and
25 pycor = eixo_y]
26        [
27             set fim_desvio true
28        ]
29        ask patches with [pxcor = (tamanho_desvio_s_c + eixo_x) and
30 pycor = (eixo_y + 9)]
31        [
32            set plabel word "ID: d" id_desvio
33            set plabel-color orange
34        ]
35       while [percorre_curva <= 4] [
36            ask patches with [pxcor = eixo_x and pycor = eixo_y]
37            [
38                set desvio_ativo false
39                set desvio_id word "d" id_desvio
40                set pcolor orange
41             ]
42             set eixo_x (eixo_x + 1)
```

```
43              set eixo_y (eixo_y + 1)
44               set percorre_curva (percorre_curva + 1)
45          ]
46          while [percorre_desvio <= tamanho_desvio_s_c]
47          [
48             ifelse metade_desvio = percorre_desvio
49             [
50                ask patches with [pxcor = eixo_x and pycor = eixo_y]
51
52                 [
53                    set pcolor orange
54                    set parada_desvio true
55                     set desvio_ativo false
56                     set desvio_id word "d" id_desvio
57                  ]
58               ]
59               [
60                  ask patches with [pxcor = eixo_x and pycor = eixo_y]
61
62                  [
63                     set desvio_ativo false
64                     set desvio_id word "d" id_desvio
65                     set pcolor orange
66                   ]
67                ]
68                set eixo_x (eixo_x + 1)
69                set percorre_desvio (percorre_desvio + 1)
70           ]
71           set percorre_curva 0
72           while [percorre_curva <= 4]
73           [
74              set eixo_y (eixo_y - 1)
75              ask patches with [pxcor = eixo_x and pycor = eixo_y]
76
77              [
78                 set desvio_ativo false
79                 set desvio_id word "d" id_desvio
80                 set pcolor orange
81              ]
82               set eixo_x (eixo_x + 1)
83               set percorre_curva (percorre_curva + 1)
84               ]
85          set id_desvio (id_desvio + 1)
86       ]
87  end
88
```

Fonte: Autoria Própria

Desta vez a estrutura da lista é diferente. O primeiro item é o indicador de desvio. O segundo item é o ponto inicial do desvio (eixo x). O terceiro item é o ponto final do desvio (eixo x). O quarto, quinto e sexto item representam a inclinação, raio da curva e altitude, respectivamente.

Nas linhas 02 a 08 do Algoritmo 3 são definidas as variáveis utilizadas para o cálculo e plotagem do desvio no ambiente gráfico. A variável *tamanho_desvio* utiliza os pontos final e inicial para obter o tamanho total do desvio, por meio do calculo do

ponto final menos o ponto inicial. A variável, *tamanho_desvio_s_c* é responsável por armazenar tamanho da reta do desvio, sem considerar o tamanho da primeira e segunda curva que possuem quatro pontos cada uma. Essas retas e curvas são definidas apenas para critério de visualização e não possuem necessariamente raios de curva maiores que zero nos momentos em que são plotadas as curvas visualmente. A variável *metade_desvio* é utilizada para definir um ponto de semáforo exatamente no meio do desvio para que seja possível a parada do trem quando solicitado pelo controlador do painel. As variáveis *percorre_desvio* e *percorre_curva* são auxiliares para execução dos laços de repetição que percorrem o plotam o desvio no ambiente gráfico. A variável *eixo_x* recupera do arquivo lido o valor do ponto inicial do desvio no eixo x. A variável *eixo_y*, como padrão para este projeto, possui o valor fixo de 80 que representa o meio do ambiente gráfico.

Durante a simulação o agente deve estar alinhado para percorrer o ambiente de maneira horizontal. Dessa forma, para que seja possível a entrada em algum desvio, a direção do veículo deve ser alterada em quarenta e cinco graus para cima. Já para a saída do desvio a direção deve ser alterada quarenta e cinco graus para baixo. Portanto, é necessária marcação de pontos no desvio que indiquem ao agente a necessidade de realizar uma curva no aspecto visual da simulação. As linhas 09 à 26 do Algoritmo 3, são responsáveis por realizar esta marcação utilizando as variáveis do *patch inicio_desvio, curva1_desvio, curva2_desvio* e *fim_desvio*.

Para que o usuário do protótipo saiba qual é o nome de cada desvio, é importante mostrar na tela os identificadores, dessa forma, facilitar o controle de abertura e parada dos trens. As linhas 27 a 32 do Algoritmo 3 fazem essa marcação em forma de texto por meio da concatenação do identificador e do valor da variável *id_desvio* definida no início da procedimento *setup-malha* no Algoritmo 2.

Por fim, a plotagem do desvio do ambiente gráfico é feita utilizando três laços de repetição no Algoritmo 4. O primeiro laço, na linha 36, realiza a plotagem a primeira curva gráfica contendo os quatro primeiros pontos do desvio transversalmente direcionados quarenta e cinco graus para cima. O segundo laço, na linha 48, antes de realizar a plotagem da reta, verifica se o ponto atual está localizado no meio do desvio, inserindo o valor *true* na variável *parada_desvio* caso a condição resulte verdadeira. O terceiro laço, na linha 74, funciona da mesma forma que o primeiro, porém faz a plotagem dos pontos transversalmente direcionada para baixo. Em todos os laços de

repetição as características de cor, identificador do desvio e as variáveis *desvio_ativo*, *inclinacao*, *raio_da_curva* e *altitude* são configuradas.

Com a execução do procedimento mostrado no Algoritmo 4, é possível ver na tela, além da via principal, os desvios lidos do arquivo texto.

3.1.3 Seções de Bloqueio

As seções de bloqueio são áreas que utilizam semáforos para sinalizar que há um trem trafegando a frente, sendo necessário que o trem que está atrás aguarde até que a passagem seja liberada, como foi mencionado no Capítulo 2.1 deste trabalho. Da mesma forma que os desvios, as seções de bloqueio não possuem representações no arquivo *xml* utilizado. Portanto, para realizar os testes e simulações, foi definido por meio de código que os semáforos ficarão localizados a cada 100 pontos da malha férrea e indicarão luz vermelha após a passagem de um trem por 60 pontos da via, ou seja, 1200 metros.

A ativação de um semáforo é feita no momento em que um trem passa por ele. Como deve permanecer fechado durante o tráfego do trem nos próximos sessenta pontos da via, foi necessário um procedimento para verificar o estado atual de cada semáforo e abri-lo quando necessário. O procedimento utilizado para essa verificação está demonstrado no Algoritmo 5.

Algoritmo 5 – Procedimento de fechamento dos semáforos

```
01 to verifica-semaforo
02     ask patches with [semaforo = true ]
03     [
04         ifelse (fechamento_semaforo != 0)
05         [
06          set fechamento_semaforo (fechamento_semaforo - 1)
07         ]
08         [ set pcolor green ]
09 ]
10 end
```

Fonte: Autoria Própria

Primeiramente são chamados todos os *patches* que representam semáforos. Na linha 03 é verificado se algum semáforo possui valor de fechamento diferente de 0, de forma que o semáforo está fechado e o trem ainda não percorreu os 60 pontos para abertura. Sendo assim é decrementado o valor um da variável de fechamento.

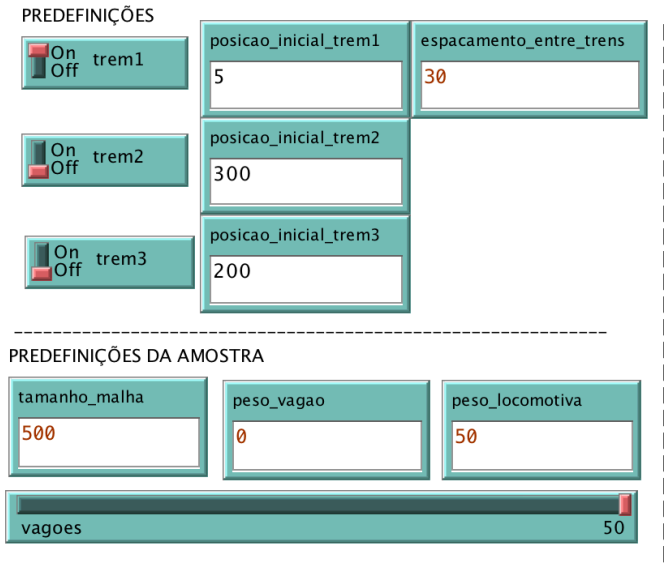
Esse procedimento ocorre até que a condição verificada na linha 03 seja falsa e a abertura do semáforo seja feita.

3.2 ELEMENTOS DE ACOMPANHAMENTO E CONTROLE DA VIA

Os elementos que compõe a interface do protótipo de simulação foram organizados em quatro colunas na interface gráfica, de forma que na primeira coluna constam os botões de controle da simulação, na segunda as variáveis de configurações iniciais, na terceira os gráficos de acompanhando da malha férrea e trem e na quarta as variáveis que podem ser controladas durante a simulação.

3.2.1 Predefinições

Figura 25 – Elementos das predefinições da interface gráfica do protótipo



Fonte: Autoria Própria

Os controladores de predefinições estão distribuídos em 11 *inputs* de 3 tipos diferentes. Esses *inputs* precisam ser configurados antes do início da simulação, uma vez que possuem informações de como os agentes devem se comportar. Os primeiros *inputs* a serem definidos são os dos trens. Para este trabalho foram definidos *inputs* na interface que permitem a criação de até três trens, definindo sua existência ou não por meio de *switches* e definido seu posicionamento inicial na via por meio de *input* de número. Como um dos trens é definido como amostra para ser utilizado

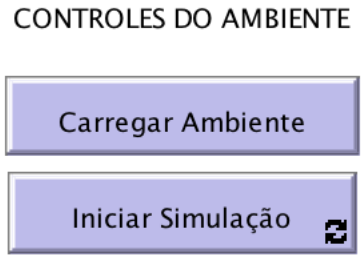
acompanhando os dados da malha e velocidade do trem, devem ser atribuídos os valores de peso de vagão, peso de locomotiva e quantidade de vagões nos *inputs peso_vagao, peso_locomotiva e vagões,* respectivamente. Além disso, também deve ser definido o *input tamanho_malha,* o qual deve ser um número inteiro que represente a quantidade de pontos que o arquivo da malha férrea possui. Esse valor é utilizado para redefinir o tamanho do ambiente de simulação.

Todas estas pré-definições são usadas nos cálculos de deslocamento do trem na via. Elas representam as características iniciais do trem e que variam de composição para composição. Outros valores necessários, como valor da área frontal, número de eixos, base rígida e bitula são atribuídos como constantes e são a todos os trens uma vez que o modelo de locomotivas e vagões usados na simulação são os iguais.

3.2.2 Controles do Ambiente

Com todas as predefinições configuradas corretamente, é possível carregar a malha férrea e dar início a simulação por meio dos botões mostrados na Figura 26.

Figura 26 – Elementos dos controles de ambiente da interface gráfica do protótipo



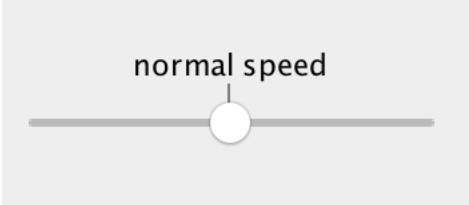
Fonte: Autoria Própria

Para dar início a simulação, é necessário ativar primeiro o botão *Carregar Ambiente* que chama o procedimento *setup* e depois ativa o botão *Iniciar Simulação* responsável por chamar a função *go*.

O botão para dar início na simulação possui um ícone de *looping* no canto inferior direito, isso significa que ao clicar no botão ele permanecerá executando o procedimento *go* em *looping* infinito. A velocidade em que o *looping* é executado

depende da velocidade de simulação definida no *slider* nativo do NetLogo representado na Figura 27.

Figura 27 – Slider de controle de velocidade de execução do NetLogo



Fonte: Autoria Própria

Cada procedimento executado pelos botões citados acima, executam uma séria de comandos.

3.2.2.1 Botão Carregar Ambiente

O botão *Carregar Ambiente* é responsável por chamar a função *setup* mostrada no Algoritmo 6.

Algoritmo 6 – Procedimento de criação do ambiente de simulação

```
01  to setup
02     clear-all
03     setup-malha
04     setup-trem
05     reset-ticks
06  end
```

Fonte: Autoria Própria

A segunda linha executada é o comando *clear-all,* nativo do NetLogo, faz a limpeza de todas as informações da tela de simulações anteriores. O segundo comando, na linha 03, chama a procedimento *setup-malha* que foi demonstrada na seção 3.1.2 deste trabalho. Na linha 04, a procedimento *setup-trem* é chamada.

Essa procedimento está demonstrada no Algoritmo 7 e é responsável por criar o(s) agente(s) e atribuir sua(s) respectiva(s) característica(s). Por fim, na linha 05 é executado o comando nativo do NetLogo denominado *reset-ticks*, responsável por reiniciar a contagem da plotagem de todos os gráficos do ambiente.

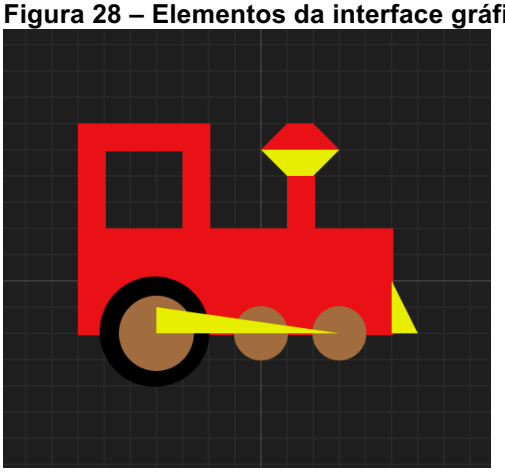
Algoritmo 7 – Procedimento de criação dos trens

```
01  to setup-trem
02     set-default-shape turtles "train"
03     if(trem1 = true) [
04        create-turtles 1  [
05            set size 15
06            set xcor posicao_inicial_trem1
07            set ycor 80
08            set heading 90
09            set color white
10            set label vagoes
11            set label-color yellow
12            set movimentar 1
13            espacamento-entre-trens
14        ]
15     ]
16     set amostra one-of turtles
17  end
```

Fonte: Autoria Própria

O NetLogo possui alguns modelos de figuras para serem utilizadas nos agentes, porém não existe nenhum com a aparência de trem. Dessa forma, foi criado manualmente uma figura utilizando a ferramenta *Turtle Shapes Editor* do NetLogo e atribuindo a ela o nome de *train.* Para acessar o editor de figura dos agentes, é necessário acessar o menu superior *tools* o clicar em *Turtle Shapes Editor*. O resultado consta na Figura 28.

Figura 28 – Elementos da interface gráfica do protótipo



Fonte: Autoria Própria

Na segunda linha do Algoritmo 7 é utilizado o comando *set-default-shapes* para definir que todos os agentes devem utilizar a figura *train.* Na linha 03 é verificado se o *switch trem1* está ativo e executa as demais linhas caso resulte verdadeiro. No Algoritmo 7 foi demonstrado para o trem 1, porém o trecho de código é válido para os

demais *switches* definidos nas predefinições da seção 3.2.1. A linha 04 executa o comando de criação do agente. A linha 05 até a linha 12 são definidas as características de posição inicial (recuperada da variável de predefinições *posição_inicial_trem1*), orientação em que está direcionado, cor, quantidade de vagões com base no *input* já mencionado e a quantidade de movimentos que fará a cada execução. Nesse caso, o valor definido sempre será um, para o que trem consiga passar por todos os pontos da malha férrea obtendo suas informações. Na linha 15, o agente é atribuído à variável global *amostra* para que possa ser acessado por outros procedimentos, como por exemplo a plotagem dos gráficos.

Por fim, durante a criação do agente, na linha 13, é feita a chamada do procedimento *espacamento-entre-trens* que em caso de criação de mais de um agente, verifica e reposiciona-o para que não ocorram sobreposições. O trem é reposicionado a quantidade de pontos definida na variável *espacamento_entre_trens*, definida nas predefinições. Isso ocorre de forma recursiva para que seja possível utilizar a imagem independentemente da quantidade de agentes criados. Essa verificação é mostrada no Algoritmo 8.

Algoritmo 8 – Procedimento de verificação de posicionamento dos trens

```
01  to espacamento-entre-trens
02     if any? other turtles-here [
03        forward 30
04        espacamento-entre-trens
05     ]
06  end
```

Fonte: Autoria Própria

Na linha 02 do Algoritmo 8, é verificado se algum agente está ocupando a mesma posição do agente que está sendo inserido. A verificação é feita utilizando o comando nativo do do NetLogo *any? Other turtles-here*. Se o agente for inserido na mesma posição, será deslocado a quantidade de pontos contida na variável *espacamento_entre_trens* na direção em que o agente está apontando, definido na variável de agente *heading.*

3.2.2.2 Botão Iniciar Simulação

Após a execução do procedimento *setup,* por meio do botão *Carregar Ambiente,* a simulação pode ser iniciada. Para isso, o procedimento *go,* representado no Algoritmo 9, precisa ser executado em *looping* por meio do botão *Iniciar Simulação.*

Algoritmo 9 – Procedimento de execução da simulação.

```
01  to go
02      verifica-estado-dos-semaforos
03      verifica-status-desvio
04      entra-desvio
05      verifica-parada
06      abre-fecha-semaforo
07      calcula-velocidade
08      ask turtles [
09        fd movimentar
10      ]
11      tick
12  end
```

Fonte: Autoria Própria

Na linha 02 no Algoritmo 9, o procedimento *verifica-estado-dos-semaforos* é chamado. Esse procedimento é responsável por verificar se os semáforos da via estão fechados, e abri-los quando necessário. O Algoritmo 10 mostra este procedimento.

Algoritmo 10 – Procedimento verificação de semáforos fechados.

```
01  to verifica-estado-dos-semaforos
02     ask patches with [semáforo = true]
03     [
04        ifelse (fechamento_semaforo != 0)
05        [ set fechamento_semaforo (fechamento_semaforo  -1) ]
06        [ set pcolor green ]
07     ]
08  end
```

Fonte: Autoria Própria

Na linha 02 do Algoritmo 10 é feita uma chamada por todos os pontos de ambiente que possuem configuração de semáforo, definida na função *setup-malha* apresentada anteriormente. Com isso, na linha 04 é verificado se a variável de ambiente *fechamento_semaforo* possui valor diferente de zero, que nesse caso significa que o semáforo em questão está fechado e precisa ser decrementando em um até atingir o valor zero. A cada execução do procedimento *go,* o valor do semáforo é decrementado, portanto quando zerar a variável, a linha 06 é executada e o

semáforo abre. A quantidade de vezes movimentos que o trem deve fazer para que o semáforo é definido na linha 13 do Algoritmo 9.

Na linha 03 do Algoritmo 9 é verificado quais desvios estão ativos. O procedimento de verificação está detalhado na seção 3.2.4 deste trabalho.

Na linha 04 do Algoritmo 9 , o procedimento *entra-desvio* é chamado para definir se o trem deve entrar no desvio. O Algoritmo 11 mostra detalhadamente como são feitas as verificações.

Algoritmo 11 – Procedimento verificação de entrada em desvios.

```
01  to entra-desvio
02     ask turtles [
03        if(desvio_ativo = true)
04        [
05           if(inicio_desvio = true)
06           [ set heading 45 ]
07           if(curva1_desvio = true)
08           [ set heading 90 ]
09           if(curva2_desvio = true)
10           [ set heading 135 ]
11           if(fim_desvio = true)
12           [ set heading 90 ]
13        ]
14     ]
15  end
```

Fonte: Autoria Própria

Na linha 02 do Algoritmo 11, todos os trens da simulação são chamados com o comando *ask turtles.* Após isso, na linha 03 é feita uma verificação se o ponto atual em que o trem está passando possui um desvio ativo. Se o desvio estiver ativo, o trem deve entrar no desvio em questão. Para isso, são necessárias algumas mudanças de direção do trem durante o percurso no desvio.

Como o os desvios possuem quatro pontos em que há necessidade do trem mudar a direção, são feitas quatro verificações de qual ponto o trem está passando. Caso o trem esteja na via principal, o primeiro ponto de desvio a ser encontrado é o ponto *inicio_desvio,* em que o trem deve ser direcionado com *heading* 45 para se movimentar em sentido de quarenta e cinco graus com relação aos eixos *x* e *y.* Os próximos pontos a serem encontrados são a *curva1_desvio, curva2_desvio* e *fim_desvio* que direcionam o trem com valor de *heading* 90, 135 e 90, respectivamente.

Na linha 05 do Algoritmo 9, o procedimento responsável por verificar a necessidade de parada dos trens é chamado. Seu código fonte está detalhado no Algoritmo 12.

Algoritmo 12 – Procedimento verificação da necessidade de parada dos trens.

```
01  to verifica-parada
02     ask turtles [
03        let auxiliar_movimentar 1
04        ask patch-ahead 1
05        [
06           if (pcolor = red)
07             [set auxiliar_movimentar 0]
08        ]
09        set movimentar auxiliar_movimentar
10     ]
11  end
```

Fonte: Autoria Própria

Na linha 02 do Algoritmo 12 todos os trens da simulação são chamados com o comando *ask turtles.* Após isso, uma variável temporada é criada como nome de *auxiliar_movimentar* com valor um. Na linha 04, é chamado um ponto de ambiente a frente de cada agente da simulação e na linha 06 é verificado se esse ponto possui cor vermelha. Caso seja verificação resulte em verdadeira, a linha 07 é executada e variável temporária recebe valor zero. Na linha 09 a variável *movimentar* de cada agente é definida com o valor da variável temporária.

Na linha 06 do Algoritmo 9, é executado procedimento que realiza o fechando do semáforo com a passagem de um trem. O Algoritmo 13 mostra o funcionamento do procedimento.

Algoritmo 13 – Procedimento de fechamento de semáforo.

```
01  to abre-fecha-semaforo
02     ask turtles [
03        if (semaforo = true)
04        [
05           let id_semaforo_aux id
06           ask patches with [id = id_semaforo_aux]
07           [
08             set pcolor red
09             set fechamento_semaforo 60
10           ]
11         ]
12      ]
13  end
```

Fonte: Autoria Própria

Na linha 02 do Algoritmo 13, todos os trens da simulação são chamados e na linha 03 é verificado se o ponto de ambiente em que o agente está passando é semáforo. Caso seja um semáforo, na linha 05 é definida uma variável temporária *id_semaforo_aux* que recebe o valor do *id* do semáforo. Na linha 06 são chamados todos os pontos de ambiente que possuem o mesmo id armazenado na variável temporária e assim, respectivamente na linha 08 e 09, são definidos a cor vermelha e a variável *fechamento_semaforo* com valor sessenta. O valor da variável *fechamento_semaforo* define quantas execuções do procedimento *go* serão feitas até que o semáforo abra-se novamente.

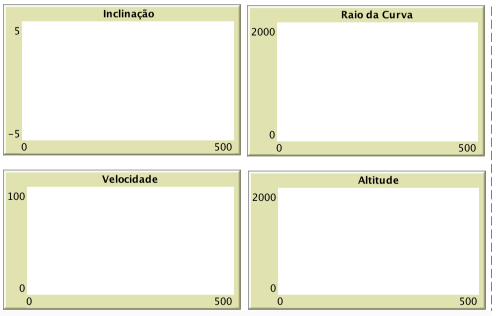
Na linha 07 do Algoritmo 9, é chamado o procedimento para cálculo de velocidade atual do trem. As fórmulas utilizadas para o cálculo podem ser encontradas na seção 3.2.3.2 deste trabalho.

Por fim, na linha 08 do Algoritmo 9 são chamados todos os trens da simulação para que na linha 09 o comando *fd,* responsável por mover os agentes, movimente o valor armazenado na variável de agente *movimentar*, que nesse caso pode ser zero ou um. Na linha 11 é utilizado um comando nativo no NetLogo denominado *tick,* responsável por atualizar todos os gráficos.

3.2.3 Análise

Durante a simulação, a cada ponto movimentado pelo trem, é feito um apanhado de dados da via para o acompanhamento por meio de gráficos. As informações que são recuperadas da via pelo agente são a inclinação atual, o raio da curva, velocidade do trem e a altitude da via e são exibidos nos gráficos da Figura 29.

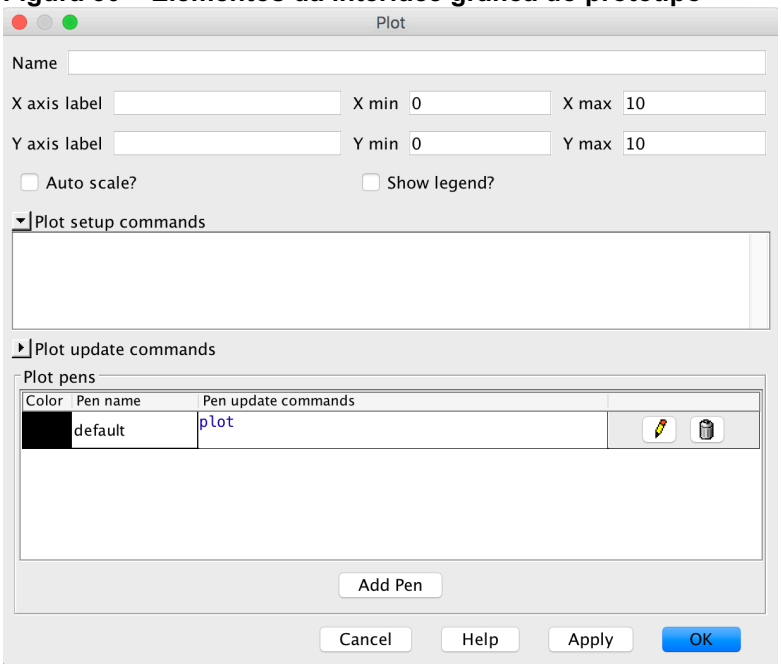
Figura 29 – Elementos de análise dos dados da via e do trem.



Fonte: Autoria Própria

As configurações dos gráficos devem ser feitas na tela de configuração de cada um, acessando a opção *Edit* ao clicar com o botão direito do *mouse* sobre o gráfico a ser configurado. A Figura 30 mostra a interface da tela de configurações de um gráfico.

Figura 30 – Elementos da interface gráfica do protótipo



Fonte: Autoria Própria

O campo *Name* informa o título do gráfico. Os campos *X axix label* e*Y axis label* não são obrigatórios e definem o título de cada eixo. Os campos *X min, X max, Y min e Y max* informam os valores intervalo de cada eixo. Essa opção é obrigatória, porém o valor pode ser alterado no campo *Plot set commands* para adaptar-se ao tamanho de ambiente desejado. A lista de *Plot pens* representa cada linha traçada no gráfico e qual o valor que será traçado. Nesse caso, podem ser traçadas mais de uma linha no mesmo gráfico, porém, para esse trabalho foi utilizado um gráfico para cada informação a ser mostrada para facilitar o entendimento do observador. Os comandos de plotagem devem ser inseridos no campo de *Pen update commands,* o qual define quais variáveis o gráfico deve utilizar para plotagem de uma linha.

3.2.3.1 Inclinação, raio de curva e altitude da via

Os valores de inclinação, raio de curva e altitude da via ficam armazenados em cada *patch* do ambiente, portanto é possível fazer a leitura destes valores no momento da passagem do agente.

Primeiramente foram definidos os valores do eixo *x* iniciando em zero e terminando com o mesmo tamanho da via, por meio da variável *tamanho_malha* mencionada anteriormente. Já os valores do eixo *y* são diferentes para inclinação, raio de curva e altitude. A inclinação possui as extremidades do eixo *y* iniciando em cinco negativo e terminando em cinco positivo. O raio de curva inicia em zero e termina em dois mil. A altitude inicia zero e termina em mil e quinhentos. Os valores atribuídos aos eixos y foram escolhidos com base na análise dos valores máximos e mínimos fornecidos pelo arquivo *xml* mostrado no início deste capitulo.

O Algoritmo 14 mostra os campos de configuração dos eixos tendo como exemplo o caso do gráfico de inclinação.

Algoritmo 14 – Configuração de tamanho dos eixos do gráfico de inclinação

```
01 set-plot-x-range  0 tamanho_malha
02 set-plot-y-range  -5 5
```

Fonte: Autoria Própria

Para realizar a plotagem do gráfico de inclinação durante a simulação, foi necessário acessar o agente que faz a leitura do valor. O Algoritmo 15 mostra o como é feito esse procedimento para o gráfico de inclinação.

Algoritmo 15 – Plotagem de inclinação do gráfico

```
01 ask amostra [ if movimentar != 0 [plot inclinacao]]
```

Fonte: Autoria Própria

O primeiro comando utilizado seleciona o agente do qual a informação será retirada. A variável *amostra* é um *turtle* que foi selecionada nas configurações dos agentes para ser utilizado como referência para os cálculos de velocidade e obtenção de informações da via. Após selecionar o agente, é realizada uma verificação em sua variável de movimento, caso o trem esteja parado, de forma que quando a variável *movimentar* for igual a zero, o gráfico não deve ser plotado. Caso o trem esteja em movimento, o comando *plot* vai inserir no gráfico o valor de inclinação do *patch* que

está em contato com o agente. O mesmo procedimento funciona para os gráficos de raio de curva e altitude.

3.2.3.2 Velocidade do trem

O procedimento para plotagem do gráfico de velocidade é o mesmo mostrado na seção anterior, porém primeiramente é necessário obter o valor final da velocidade. Para isso, foram necessários alguns cálculos com base em algumas equações físicas. A Tabela 1 mostra as equações utilizadas para o cálculo de velocidade em sua forma matemática e convertida para NetLogo.

Após todos os cálculos necessários para obtenção da velocidade final do trem em uma determinada situação, o gráfico utiliza a variável *Vf* para plotar o ponto em questão.

Em uma situação real, os vagões encontram-se em posições diferentes das locomotivas, portanto seria necessário fazer a leitura da via para cada posição de vagão. Para este trabalho os valores obtidos da malha foram considerados os mesmos para os vagões e locomotivas.

3.2.4 Controladores

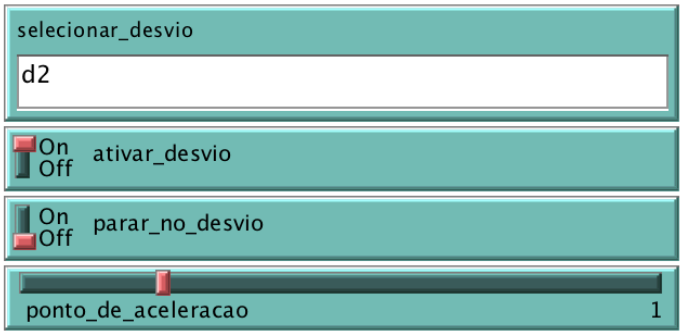
Os controladores são itens da interface gráfica que podem ser manipulados pelo usuário durante a simulação. Os três primeiros itens da Figura 31 são utilizados para controle dos desvios e o último item para controle do ponto de aceleração do trem de amostra.

Tabela 1 – Equações para o cálculo de velocidade do trem

Equação	Equação em NetLogo	Descrição
$v_F = \sqrt{\left\| v^2 + \frac{(F_{ac} \times 20)}{(4 \times W)} \right\|}$	**let** auxiliar_raiz ((Va * Va) + ((Fac * 20) / (4 * Wt))) **if** (auxiliar_raiz < 0) [**set** auxiliar_raiz (auxiliar_raiz * (-1))] **set** Vf **sqrt**(auxiliar_raiz)	Velocidade final
$F_{ac} = F_t - R_T$	**set** Fac (Ft - Rtotal)	Força de aceleração disponível (em kgf)
$F_t = \frac{273.24 \times 0.82 \times HP}{v}$	**set** Ft ((273.24 * 0.82 * HP) / Va)	Força trator (em kgf)
$R_T = \sum_{i=0}^{n_l} (R_{nl} + R_{cl} + R_\gamma + R_i) + \sum_{i=0}^{n_v} (R_{nv} + R_{cv} + R_\gamma + R_i)$	**set** Rtotal ((n_locomotivas * (Rnl + Rcl + Ry + Ri)) + (n_vagoes * (Rnv + Rcv + Ry + Ri)))	Resistência total do trem
$R_{cv} = \frac{500 \times b}{R}$	**set** Rcv ((500 * b) / R)	Resistência de curva do veículo (em kgf)
$R_{nv} = 1.3 + \frac{29}{w} + 0{,}045 \times v + \frac{0{,}0024 \times A \times v^2}{w \times n}$	**set** Rnv (1.3 + (29 / wv) + (0.045 * Va) + ((0.0024 * A * (Va * Va)) / (wv * n_eixos)))	Resistência normal dos vagões (em kgf)
$R_i = 10 \times i$	**set** Ri (10 * i)	Resistência de tampa do veículo (em kgf)
$R_\gamma = 4 \times \frac{v_F^2 - v^2}{\ell}$	**set** Ry (4 * (((Va + 2) - Va) / 20))	Resistência inercial (em kgf)
$R_{cl} = 0{,}2 + \frac{100}{R} \times (br + b + 3{,}8)$	**set** Rcl (0.2 + (100 / R) * (br + b + 3.8))	Resistência de curva de locomotiva (em kgf)
$R_{nl} = 1{,}3 + \frac{29}{w} + 0{,}03 \times v + \frac{0{,}0024 \times A \times v^2}{w \times n}$	**set** Rnl (1.3 + (29 / wl) + (0.03 * Va) + ((0.0024 * A * (Va * Va)) / (wl * n_eixos)))	Resistência normal da locomotiva (em kgf)

Fonte: Adaptado de Borges (2015).

Figura 31 – Elementos de controle da via e trem.



Fonte: Autoria Própria

O controle de abertura e fechamento de desvios possui três *inputs* que podem ser alterados. O primeiro *input selecionar_desvio* é um campo de texto o qual deve receber o valor digitado pelo usuário do desvio que deverá ser controlado, a verificação do desvio selecionado está representada na linha 02 e na linha 07 do Algoritmo 16. Para saber o nome do desvio corretamente, basta olhar para os títulos dos desvios carregados no ambiente de simulação. Com o desvio selecionado, podem ser feitas duas alterações. A primeira, manipulada pelo *ativar_desvio,* define se o desvio estará aberto para o próximo trem que passar por ele, essa verificação é feita na linha 04 do Algoritmo 16 e define a variável de ambiente *desvio_ativo* como verdadeira ou falsa. A segunda alteração que pode ser feita no desvio, é manipulada pelo *input parar_no_desvio,* que define se o trem deve parar ao chegar no meio do desvio. Essa verificação é feita pela linha 11 do Algoritmo 16 e em caso resultar verdadeiro o ponto central do desvio recebe cor vermelha.

Algoritmo 16 – Manipulação de desvios ativos

```
01 to verifica-status-desvio
02    ask patches with [desvio_id = selecionar_desvio ]
03     [
04       ifelse(ativar_desvio = true)
05           [ set desvio_ativo true ]
06           [ set desvio_ativo false ]
07       ]
08       ask patches with [desvio_id = selecionar_desvio and
09 parada_desvio = true ]
10        [
11          ifelse(parar_no_desvio = true)
12              [ set pcolor red ]
13              [ set pcolor orange ]
14        ]
15 end
```

Fonte: Autoria Própria

O controlador *ponto_de_aceleracao* pode ser modificado entre o valor negativo menos um (desaceleração) até o valor positivo oito. Cada ponto selecionado representa um valor de potencia *HP* utilizado para o cálculo de velocidade do trem. A Tabela 2 mostra qual o valor de potência para cada ponto de aceleração selecionado.

Tabela 1 – Pontos de aceleração e seus respectivos valores de potência.

Ponto de aceleração (*ponto_de_aceleracao*)	**Valor de Potência (HP)**
-1	0
0	0
1	100
2	275
3	575
4	960
5	1440
6	1930
7	2500
8	2940

Fonte: Adaptado de Borges (2015).

Ao definir o *ponto_de_aceleracao* com valor negativo menos um, será feita realizada uma desaceleração de dois quilômetros por hora do trem a cada vinte metros percorridos, de forma que a cada chamada do procedimento *go,* será feito esse decremento. A verificação é feita no trecho de código contido no Algoritmo 17. Esse trecho de código está dentro do procedimento de *calcula-velocidade,* citado anteriormente.

Algoritmo 17 – Verificação de desaceleração

```
01 ask amostra [
02    if (velocidade_atual != 0)
03    [
04       ifelse(ponto_de_aceleracao = -1)
05       [
06          set Va (velocidade_atual - 2)
07       ]
08       [
09          set Va velocidade_atual
10       ]
11    ]
12 ]
```

Fonte: Autoria Própria

4 EXPERIMENTOS E RESULTADOS

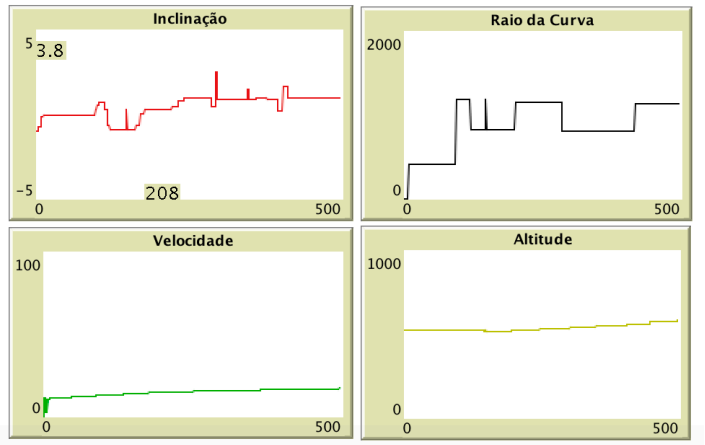
Para realizar os experimentos afim de verificar os resultados obtidos, foram realizados testes com cenários diferentes. Em cada cenário de teste as variáveis de predefinições foram alteradas antes do início da simulação e as variáveis de controle foram manipuladas durante a simulação. Em todos os experimentos foi utilizada uma malha férrea com quinhentos pontos, dessa forma a variável *tamanho_malha* se manteve sempre com o mesmo valor.

4.1 CENÁRIOS

O primeiro experimento foi realizado com apenas um trem transitando na via. Para esse experimento foram feitos três testes diferentes.

Para o primeiro teste os valores de predefinições do trem de amostra foram: *input trem1* ativado, *posicao_inicial_trem1* com valor cinco, *peso_vagao* com valor trezentos, *peso_locomotiva* com valor quinhentos, *vagões* com valor cinquenta. Durante toda a simulação o *ponto_de_aceleracao* se manteve com valor um. O resultado obtido nos gráficos está demonstrado na Figura 32.

Figura 32 – Gráficos do primeiro teste do primeiro experimento.

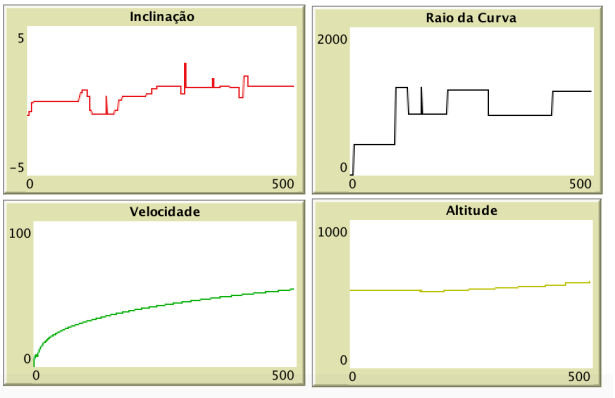


Fonte: Autoria Própria

Foi possível perceber que a velocidade do trem não teve um aumento significativo, isso se deve ao fato do ponto de aceleração exercer uma potência baixa.

Para o segundo teste foram mantidos os mesmos valores nas variáveis, porém durante toda a simulação o valor de *ponto_de_aceleracao* foi definido com valor oito. O resultado dos gráficos está demsonstrado na Figura 33.

Figura 33 – Gráfico do segundo teste do primeiro experimento.

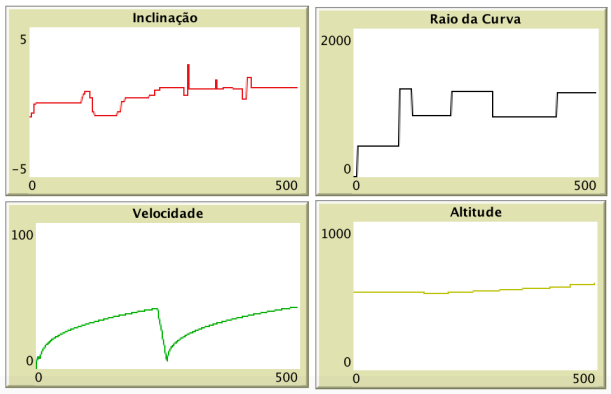


Fonte: Autoria Própria

Para este segundo teste o incremento na velocidade é bastante perceptível com parado ao primeiro. Nos dois casos de testes a velocidade se manteve em crescimento, porém a rapidez com que ela cresceu foi diminuindo ao final do gráfico. Isso se deve ao fato da inclinação possuir valores maiores ao final da via.

O terceiro teste foi executado com *ponto_de_aceleracao* definido com valor oito até metade da via e com valor negativo um durante aproximadamente duzentos metros. Após isso o *ponto_de_aceleracao* foi definido com valor oito novamente. O resultado está demonstrado na Figura 34.

Figura 34 – Gráfico do terceiro teste do primeiro experimento.

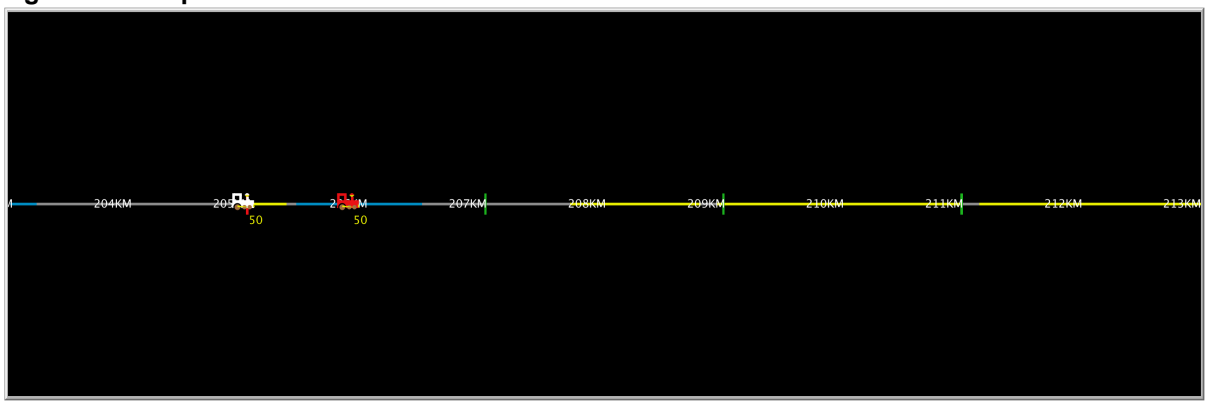


Fonte: Autoria Própria

É perceptível observando os gráficos que a desaceleração acorre de forma rápida, uma vez que a cada vinte metros (um ponto da via) percorridos ocorre uma desaceleração de dois quilômetros por hora.

Com os testes de gráficos realizados, o objetivo do segundo experimento é verificar o comportamento de mais de um trem transitando ao mesmo tempo na via. Para o primeiro teste deste experimento, as variáveis *peso_vagao, peso_locomotiva* e *vagões* são irrelevantes. Já as variáveis *trem1* e *trem2* recebem valor verdadeiro com o *switch* ativo. A posição inicial do primeiro e segundo trem recebem os valores zero e trinta, respectivamente. A simulação está representada na Figura 35.

Figura 35 – Experimento com dois trens na via.

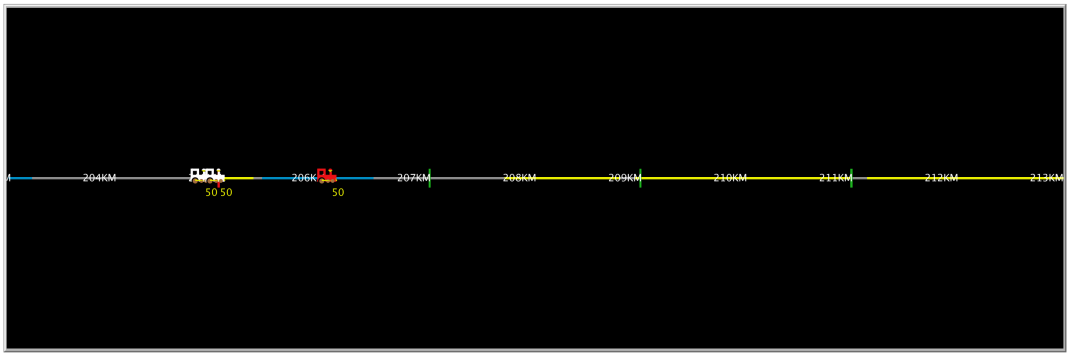


Fonte: Autoria Própria.

No momento em que o primeiro trem passou pelo primeiro semáforo da via, o semáforo permaneceu fechado durante os próximos sessenta pontos percorridos pelo trem. Quando o segundo trem chegou ao primeiro semáforo, ele ainda estava fechado, portanto ouve uma espera até que fosse aberto novamente. Nenhuma colisão ocorreu durante toda a simulação.

O segundo teste realizado neste cenário contou com a presença de mais um trem transitando na via. A variável *trem3* foi ativada e a variável *posicao_trem_3* recebeu valor sessenta. As demais variáveis permaneceram da mesma forma demonstrada no primeiro teste. A simulação do segundo teste está demonstrada na Figura 36.

Figura 36 – Experimento com três trens na via.



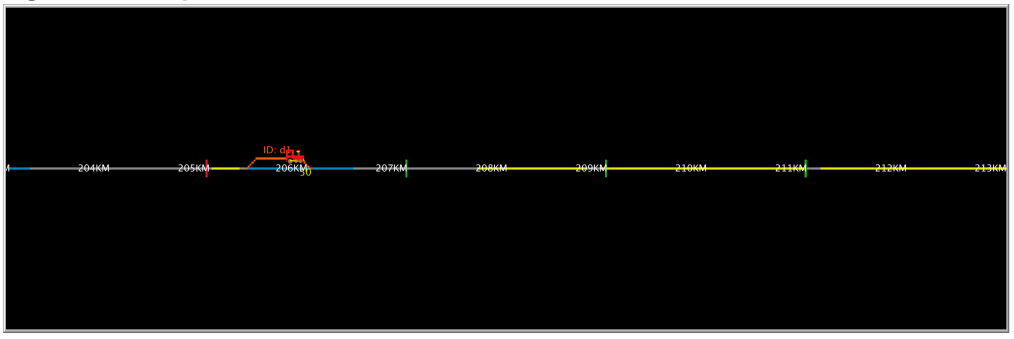
Fonte: Autoria Própria.

Os dois primeiros trens se comportaram da mesma forma do primeiro teste. Já o último trem que estava percorrendo a via colidiu com o segundo que estava em espera pela abertura do semáforo. Isso se deve ao fato dos três trens iniciarem a locomoção muito próximos uns aos outros.

O objetivo do terceiro experimento é demonstrar a entrada de um trem em um desvio da via. O desvio foi carregado pelo arquivo de texto com inicio no ponto cento e vinte e término no ponto cento e cinquenta. Para esse experimento foram desconsiderados os valores das variáveis de predefinições e apenas a variável *trem1* ativa com *posicao_inicial_trem1* com valor zero.

Para o primeiro teste do experimento o desvio *d1* foi ativo por meio do *switch ativar_desvio,* porém o *switch parar_no_desvio* não foi ativo. A simulação está demonstrada na Figura 37.

Figura 37 – Experimento com um desvio ativo na via.

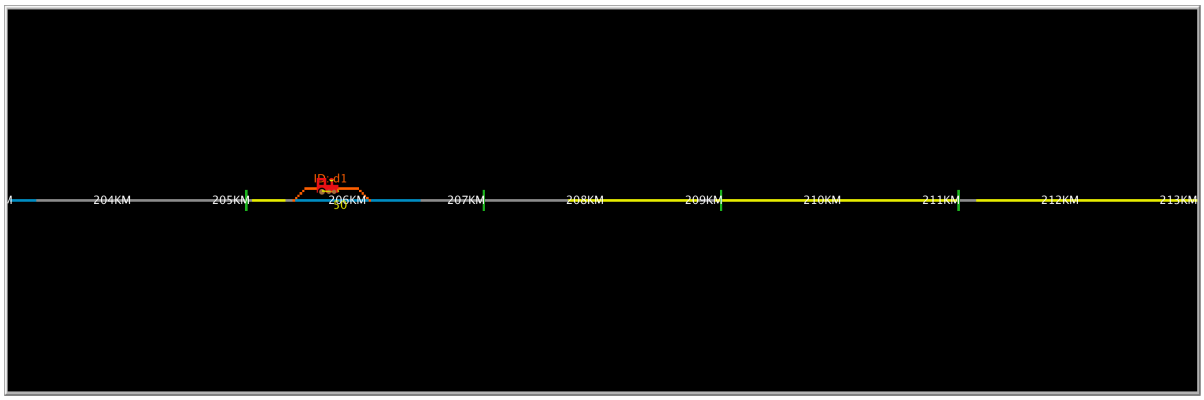


Fonte: Autoria Própria

O trem pode entrar e sair do desvio fazendo as curvas necessárias sem nenhum tipo de interrupção.

O segundo teste foi realizado na mesma forma que o primeiro, porém o *switch parar_no_desvio* foi ativo. A simulação está demonstrada na Figura 38.

Figura 38 – Experimento com desvio e parada ativos.



Fonte: Autoria Própria

Ao chegar ao ponto central do desvio, o trem se deparou com um o ponto marcado em vermelho, portanto permaneceu parado até que o *switch parar_no_desvio* fosse alterado novamente.

5 CONCLUSÃO

O objetivo deste trabalho foi desenvolver um protótipo para visualização de posicionamento de trens em uma malha férrea utilizando o *framework* NetLogo.

Por meio do referencial teórico foi possível compreender acerca das características e funcionamentos dos agentes, bem como dos ambientes em que os agentes devem atuar. Desta forma foram implementados os funcionamentos dos trens com e as propriedades malha férrea. Também foi possível entender o funcionamento da forma de condução de trens bem como a interpretação dos dados que uma malha férrea contém. Além disso, foi obtido um breve conhecimento de alguns *frameworks* existentes para implementação de agentes e um conhecimento mais aprofundado no *framework* Netlogo.

A analise do comportamento dos trens no protótipo desenvolvido permitiu com facilidade a observação dos dados na tela em tempo de execução bem como o posicionamento atual dos trens. Foram implementados botões de interação com o protótipo para que o usuário consiga com facilidade alterar alguns dados necessários para o inicío da simulação e também para controle durante a simulação.

Conclui-se que por meio da fundamentação teórica e do desenvolvimento do código fonte foi possível cumprir os objetivos específicos por meio da revisão da literatura sobre agentes, compreensão do domínio acerta dos elementos que o compõe, compreensão das formas controlar o posicionamento dos trens observando seu deslocamento na malha férrea e compreensão do funcionamento do NetLogo para implementação de agentes. Dessa forma o objetivo geral foi cumprido.

5.1 TRABALHOS FUTUROS

Para os trabalhos futuros, sugere-se a implementação de novos recursos ao protótipo de simulação. A primeira sugestão é a adaptação do ambiente para suportar quantidades maiores de trens transitando na via ao mesmo tempo, isso pode ser feito com a adaptação dos *inputs* de entrada da interface. A segunda sugestão é a possibilidade e escolher qual trem da via terá seus dados visualizados nos gráficos, ou até mesmo a exibição dos dados de mais de um trem ao mesmo tempo. A terceira

sugestão é a implementação de trens seguindo em sentidos diferentes, nesse caso é necessária uma adaptação também nos semáforos da via ou até mesmo implementação de novos recursos de segurança. E por fim, a ultima sugestão é o acompanhamento do tempo de viagem de um determinado trem de um ponto a outro.

Existem várias outras possibilidades de melhoria do protótipo para que se aproxime cada vez mais de um *software* utilizado no mundo real.

REFERÊNCIAS

BAJPAI, A. et al. An Introduction to the Train Management System Of Western Railway, 2007.

BALL, Albert. Railway Operation Simulator. 2017. Disponível em: <http://www.railwayoperationsimulator.com/>. Acesso em: 9 jun. 2017.

BELLEIFEMINE, F. et al. **JADE –** A White Paper. 2003.

BORGES, A. P. **Uma contribuição para geração de política de ações para condução de trens de carga usando raciocínio baseado em casos**. 2015. 234 f. Tese de Doutorado. Pontifícia Universidade Católica do Paraná, 2015.

DORDAL, O. B. et al. Strong reduction in fuel consumption driving trains in bi-directional single line using crossing loops. In: **Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on**. IEEE, 2011. p. 1597-1602.

FERBER, Jacques. **Multi-agent systems: an introduction to distributed artificial intelligence**. Reading: Addison-Wesley, 1999.

JERRY, Kponyo et al. NetLogo implementation of an ant colony optimisation solution to the traffic problem. **IET Intelligent Transport Systems**, v. 9, n. 9, p. 862-869, 2015.

NetLogo. User Manual**,** version 5.3.1. Disponível em: <https://ccl.northwestern.edu/netlogo/docs/>. Acesso em: 16 out. 2016.

RACHEL, Flávio Monteiro. **Proposta de um controlador automático de trens utilizando lógica nebulosa preditiva**. 2006. Tese de Doutorado. Universidade de São Paulo.

RUSSELL, Stuart Jonathan et al. **Artificial intelligence: a modern approach**. Upper Saddle River: Prentice hall, 2003.

SILVEIRA, Thiago Lopes Trugillo el al. Provimento de informações sobre transporte público urbano para empresas concessionárias: Simulação e avaliação apoiada por

um sistema multi-agentes. **Proceedings of the X Simpósio Brasileiro de Sistemas de Informação**, 2014.

SOLOMON, B. Railroad Signaling. MBI Publishing Company, 2003.

SYSFER. Sistemas WebRail, 2017. Disponível em http://sysfer.net.br/site/produtos/sistemas-webrail/>. Acesso em 2 de jun. 2017.

TEIXEIRA, Fábio V. Jade: **Java Agent Development Framework.** Disponível em: <http://www.dca.fee.unicamp.br/~gudwin/courses/IA009/artigos/IA009_2010_12.pdf>

TELECOM ITALIA. JADE. Disponível em: <http://jade.tilab.com/>. Acesso em: 9 jun. 2017.

TISUE, Seth et al. Netlogo: A simple environment for modeling complexity. In: **International conference on complex systems**. 2004.

WEIGANG, Li et al. Implementação do Sistema de Mapeamento de uma Linha de Ônibus para um Sistema de Transporte Inteligente. Em: **Anais do XXI Congresso da Sociedade Brasileira de Computação, Seminário Integrado de Software e Hardware**. 2001.

Wilensky, U. (1998). NetLogo Traffic 2 Lanes model. http://ccl.northwestern.edu.netlogo/models/Traffic2Lanes. Center for Connected Learning and Computer-Base Modeling, Northwestern University, Evanston, IL.

Wilensky, U. (2003). NetLogo Traffic Grid mode.l http://ccl.northwestern.edu.netlogo/models/Traffic2Lanes. Center for Connected Learning and Computer-Base Modeling, Northwestern University, Evanston, IL.

WILLSON, E. Rail Traffic Controller (RTC). **Berkeley Simulation Software**, 2012.

WOOLDRIDGE, Michael. **An introduction to multiagent systems**. John Wiley & Sons, 2002.

XAVIER, Marcio. F. **A importância do modal ferroviário no transporte de carga no Brasil utilizando a intermodalidade**. Monografia – Tecnólogo em Logística com ênfase em Transporte, Faculdade de Tecnologia da Zona Leste. São Paulo, 2006.