UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

SIMULADOR DE TRÁFEGO DE AUTOMÓVEIS EM UMA MALHA RODOVIÁRIA: VERSÃO 2.0

MAYCO ANDREY RANGHETTI

BLUMENAU
2007

2007/1-30

MAYCO ANDREY RNAGHETTI

SIMULADOR DE TRÁFEGO DE AUTOMÓVEIS EM UMA MALHA RODOVIÁRIA: VERSÃO 2.0

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. José Roque Voltolini da Silva – Orientador

BLUMENAU
2007

2007/1-30

SIMULADOR DE TRÁFEGO DE AUTOMÓVEIS EM UMA MALHA RODOVIÁRIA: VERSÃO 2.0

Por

MAYCO ANDREY RANGHETTI

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

Presidente:	Prof. José Roque Voltolini da Silva – Orientador, FURB
Membro:	Prof. Maurício Capobianco Lopes – FURB
Membro:	Prof. Paulo Cesar Rodacki Gomes – FURB

Blumenau, 10 de julho de 2007

Dedico este trabalho a minha família, em especial Liana, que sempre esteve próximo de mim mesmo nos momentos mais difíceis.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

À minha família que sempre esteve presente.

Aos meus amigos, pelos empurrões e cobranças.

Ao meu orientador por ter acreditado na conclusão deste trabalho.

A todos que de uma forma ou de outra contribuíram para o desenvolvimento deste trabalho.

Não confunda jamais conhecimento com sabedoria. Um o ajuda a ganhar a vida; o outro a construir uma vida.

Henry Ford

RESUMO

Este trabalho consiste em uma extensão do Simulador de Controle de Tráfego de Veículos em uma Malha Rodoviária desenvolvido por Jocemar Freire (FREIRE, 2004). Foram realizadas adaptações ao código fonte desenvolvido por Freire para estender algumas funcionalidades e também para corrigir inconsistências observadas no simulador. A principal extensão realizada foi a criação de um veículo controlado pelo usuário e a introdução da terceira dimensão (3D) na definição de trechos, possibilitando assim a representação e a visualização de viadutos. No desenvolvimento do simulador foi utilizado o ambiente de programação Delphi 7, utilizando as rotinas da biblioteca OpenGl para a representação gráfica das ruas e veículos.

Palavras-chave: Simulador. Controle de tráfego.

ABSTRACT

This work consists of an extension to the Vehicles Traffic Control Simulator in a Road Mesh developed by Jocemar Freire (FREIRE, 2004). Adaptations to the code source developed for Freire had been to extend some functionalities and also to correct inconsistencies observed in the simulator. The main extension was the creation of a controlled vehicle for the user and the introduction of the third dimension (3D) in the definition of stretches, thus making possible the representation and the visualization of viaducts. In the development of the simulator the programming environment Delphi 7 was used using the routines of the OpenGl library for the graphical representation of the streets and vehicles.

Key-words: Simulator. Traffic control.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

> Quanto mais as cidades crescem, mais o homem quer conforto e comodidade. Para tanto, usa cada vez mais os sistemas de transporte. Esse ciclo vicioso gera cada vez mais problemas no trânsito, pois apesar de estar em constante crescimento, as vias de acesso da cidade não podem simplesmente ser 'alargadas'. (SILVA, 2005, p. 11).

Muitas propostas tentam resolver desconfortos que os usuários de veículos possam ter, citando como exemplo o congestionamento. Há propostas que usam dispositivos em tempo real para analisar o tráfego em uma via urbana (ZANUZ, 2005), sistemas controlados por multiagentes (SILVA, 2005) e outras capazes de prover um trajeto onde o fluxo de veículos seja menor (HOFFEMANN; AEBI; BORTOLETO, 2006), evitando desta forma a incidência de congestionamento de veículos.

Uma alternativa proposta em Freire (2004) é um simulador de tráfego de veículos, considerado pelo autor como um sistema útil para o planejamento de malhas rodoviárias. Conforme Freire (2004, p. 13), "para verificação da validade de melhorias propostas numa malha viária antes de sua implantação no mundo real, um simulador de trânsito torna-se útil." Através do simulador de trânsito pode-se estudar mudanças nas vias, objetivando amenizar congestionamentos e controlar temporização dos semáforos.

No protótipo desenvolvido por Freire (2004), os veículos transitam na malha rodoviária de forma robotizada[1]. O usuário define parâmetros iniciais, como a malha rodoviária e quantidade de veículos, e o protótipo simula o fluxo de veículos nas vias.

Freire (2004) desenvolveu o protótipo utilizando processos concorrentes, onde para cada veículo circulando na malha, existe um processo (*thread*) instanciado e todos são executados simultaneamente, de forma concorrente. Os veículos não tem uma rota pré-definida. As mesmas são determinadas aleatoriamente.

Na tentativa de se obter resultados precisos e garantir que todas as rotas e cruzamentos sejam testados no protótipo de Freire (2004), propõe-se a criação de um veículo controlado, onde o usuário, através de uma interface de entrada, possa determinar as trajetórias a serem seguidas. Ainda, propõem-se melhorias na interface e uma verificação geral no código fonte será realizada.

O trabalho aqui proposto é uma extensão ao trabalho desenvolvido por Freire (2004). O ambiente de programação será o mesmo utilizado em Freire (2004): o Delphi 7 com a biblioteca gráfica OpenGl.

1.1 OBJETIVOS DO TRABALHO

O objetivo desta proposta é estender o trabalho desenvolvido por Freire (2004), acoplando novas funcionalidades e reformulando (com adaptações ou correções) algumas funções.

Os objetivos específicos do trabalho são:

a) disponibilizar um veículo guiado pelo usuário no simulador;
b) acrescentar a terceira dimensão nas coordenadas dos trechos da malha rodoviária;
c) reimplementar a parte de visualização da malha, considerando a coordenada z.

1.2 ESTRUTURA DO TRABALHO

O primeiro capítulo deste trabalho traz uma introdução ao tema abordado. O segundo capítulo descreve os fundamentos teóricos para a realização do simulador de tráfego rodoviário em uma malha rodoviária. No terceiro capítulo é mostrado como o desenvolvimento foi realizado e o quarto e último capítulo traz as conclusões e resultados obtidos.

[1] A movimentação dos veículos é de forma automatizada, sem a interferência do usuário.

2 FUNDAMENTAÇÃO TEÓRICA

Nas seções a seguir são abordados os temas relacionados ao trabalho, os quais são: sistema de controle de trânsito, apresentação do simulador de controle de tráfego desenvolvido por Freire (2004), rotinas matemáticas usadas em representação de malhas viárias, computação gráfica usando a biblioteca gráfica OpenGl, processos concorrentes e trabalhos correlatos.

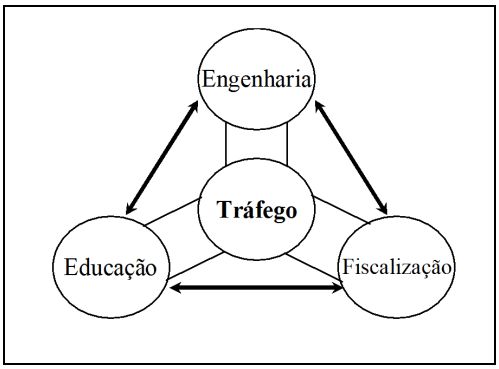
2.1 SISTEMA DE CONTROLE DE TRÂNSITO

Trânsito “é a movimentação e imobilização de veículos, pessoas e animais nas vias terrestres” (DENATRAN, 2004).

O controle de tráfego garante que veículos e pessoas movimentem-se com eficiência e segurança. Uma via é considerada eficiente quando acomoda uma certa quantia de veículos, possibilitando uma movimentação fluente e ainda pode ser considerada segura quando há condições para reduzir ou eliminar os acidentes (MONTEIRO, 2006).

Para Monteiro (2006, grifo do autor) (Figura 1), “os sistemas de tráfego são baseados em três pilares. Estes três pilares, conhecidos como 3E´s [...], compreendem: engenharia (***e****ngineering*), educação (***e****ducation*) e fiscalização (***e****nforcement*).” Eles podem ser definidos como:

a) relacionamento entre a via transitável e o veículo com o objetivo de transformar as vias e veículos cada vez mais seguros (engenharia);
b) uma população instruída sobre a correta técnica de circulação, causas e conseqüências de acidentes, atitudes corretas no trânsito, certamente trará maior segurança e tranqüilidade para o trânsito (educação);
c) fiscalizar e aplicar as penalidades previstas em lei são atividades complementares necessárias para garantir a integridade do sistema (fiscalização).



Fonte: Monteiro (2006).

Figura 1 – Pilares do sistema de tráfego

Segundo BRASIL (1997) no Código de Trânsito Brasileiro (CTB) “considera-se trânsito a utilização das vias por pessoas, veículos e animais, isolados ou em grupos, conduzidos ou não, para fins de circulação, parada, estacionamento e operações de carga ou descarga.”; e complementa que “são vias terrestres urbanas e rurais as ruas, as avenidas, os logradouros, os caminhos, as passagens, as estradas e as rodovias.”

O parágrafo único do artigo 2º do CTB em BRASIL (1997) ainda define que “são consideradas vias terrestres as praias abertas à circulação pública e as vias internas pertencentes aos condomínios constituídos por unidades autônomas.”

Para Monteiro (2006) “são usuários dos sistemas de tráfego os ocupantes dos veículos, guiando-os ou não, e os pedestres, principalmente.”

Em um sistema de tráfego um veículo pode ser um automóvel, ônibus, caminhões, motocicletas, triciclos, bicicletas, carroças, bonde, etc, entretanto, para uma definição padrão e menos abrangente é suficiente adotar uma unidade veicular padrão. Assim um veículo pode ser definido como um automóvel de passeio básico (MONTEIRO, 2006).

A via ou rua é o espaço destinado à circulação de pessoas e veículos. O sistema viário urbano é formado de dois grupos distintos de componentes básicos: os nós e as ligações. As ligações são vias que interligam os nós que por sua vez caracterizam-se por interseções, cruzamentos, passagem de nível, etc (MONTEIRO, 2006).

As vias são definidas quanto às suas características físicas como também ao tipo de uso. Monteiro (2006) sugere a seguinte classificação:

a) vias expressas: vias de ligação entre cidades, geralmente com duplo sentido, alto volume de tráfego e velocidade em torno de 80km/h;
b) vias locais: via predominantemente local que promovem acesso a residências ou locais de trabalho com velocidade média de 20km/h;
c) vias arteriais: ligação entre a via expressa e a cidade ou uma região, com volume de tráfego e velocidade média de 60km/h;

d) vias colaterais: ligação entre vias arteriais e localidades residenciais ou de trabalho, com velocidade média de 40km/h.

Um cruzamento é definido quando há duas vias, ou mais, em sentido de trafegabilidade oposto e, em níveis iguais que se cruzam ou convirjam para um mesmo ponto (MONTEIRO, 2006). Para Monteiro (2006), uma solução para evitar que haja conflitos em cruzamento de mesmo nível é a criação de viadutos, onde as vias são projetadas em níveis diferentes, permitindo sua sobreposição.

Segundo Monteiro (2006) “o controle de tráfego é, fundamentalmente, a supervisão do movimento dos veículos, pessoas e bens, com o objetivo de garantir a eficiência e a segurança”. Muitas vezes os dois objetivos, eficiência e segurança, são conflitantes. Um exemplo é a instalação de semáforos em cruzamentos, que apesar de garantir a segurança, acaba reduzindo a eficiência e a mobilidade das vias.

As técnicas de controle de tráfego estão ligadas ao tipo de via e localização da via. Em ambientes que há escolas, hospitais ou intenso movimento de veículos (saída de restaurantes, *shopping center*) o ideal é que haja um gerenciamento de velocidade com auxílio de aparelhos que medem a velocidade dos veículos e aplicam multas caso houver excesso de velocidade. Em cruzamentos de mesmo nível é imprescindível a semaforização, para garantir a segurança dos motorista e pedestres e permitir maior fluidez do trânsito (MONTEIRO, 2006). Os sinais verticais e horizontais são usados para alertar os motoristas e pedestres que há fiscalização de controle de tráfego na via (MONTEIRO, 2006).

2.2 PROTÓTIPO DE SIMULAÇÃO DO CONTROLE DE TRÁFEGO DE AUTOMÓVEIS EM UMA MALHA RODOVIÁRIA URBANA

O objetivo principal do protótipo de software desenvolvido por Freire (2004) é simular o trânsito de veículos em uma malha rodoviária. O mapa rodoviário utilizado pelo protótipo inicialmente era configurado manualmente num arquivo texto. Atualmente pode ser gerado num editor gráfico proposto e implementado por Bertoldi (2005), o qual foi estendido por Froeschlin (2006).

> Para a construção de um simulador de trânsito de veículos é necessário modelar atividades que trabalhem de forma simultânea, independentes, mas que em determinados momentos comunicam-se com outras para obter ou liberar um recurso ou ainda evitar conflitos que possam ocorrer. (FREIRE, 2004, p. 13).

O protótipo foi desenvolvido no ambiente Delphi 7 com a biblioteca gráfica OpenGl. A visualização da malha rodoviária foi implementada usando a biblioteca gráfica OpenGl. O desenho da malha e dos veículos transitando são definidos em duas dimensões (2D). Conseqüentemente, a visualização também é em duas dimensões.

Algumas funcionalidades definidas e implementadas no protótipo são:

a) permite a configuração da quantidade e velocidade dos carros que circulam na malha rodoviária;

b) permite a mudança de *zoom* sobre a malha rodoviária;

c) permite o cálculo da distância entre dois pontos sobre a representação gráfica.

Toda a movimentação dos carros é de forma automatizada, assim a rota escolhida não é previsível e pode ocorrer que caminhos alternativos nunca sejam utilizados em uma simulação.

A Figura 2 mostra a tela inicial do simulador desenvolvido por Freire (2004). Após ter carregado o arquivo com a especificação da malha rodoviária, a mesma é exibida na tela do simulador (Figura 3).

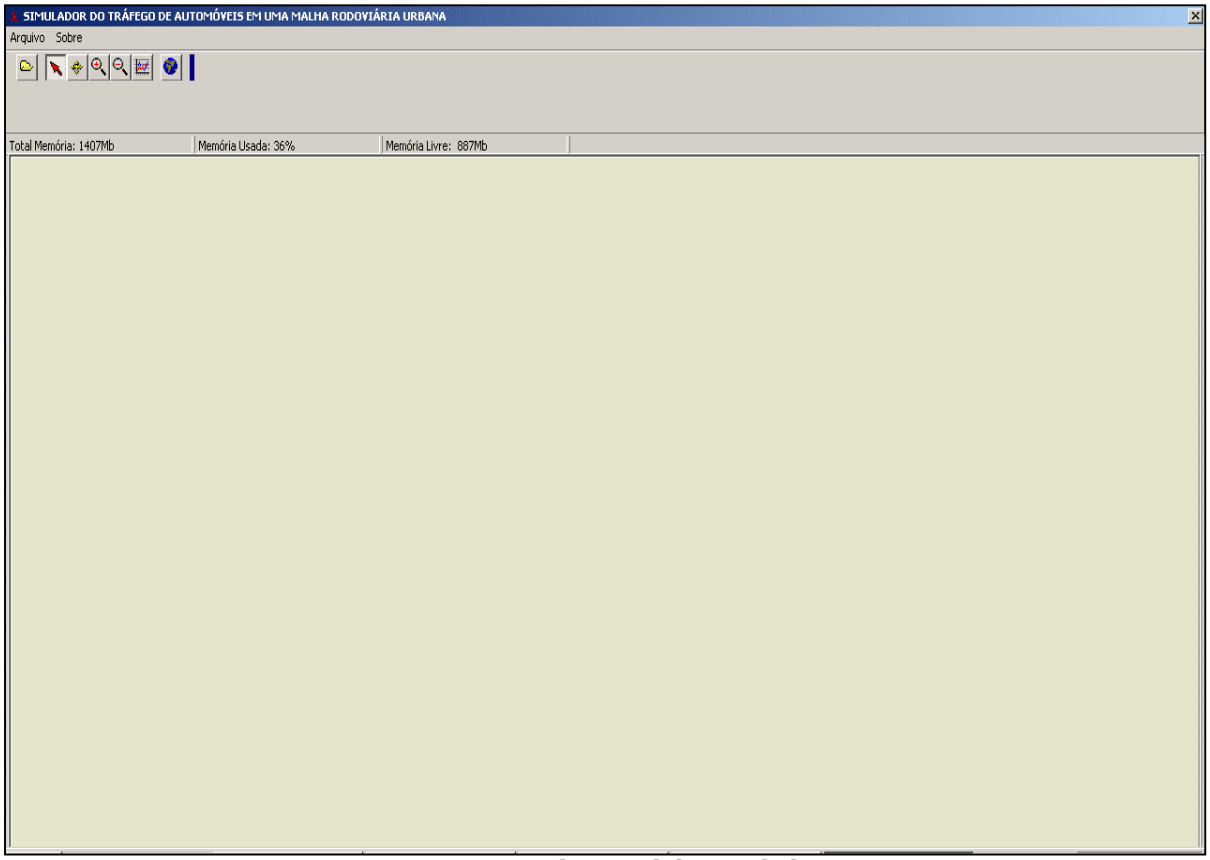


Figura 2 – Tela inicial do simulador

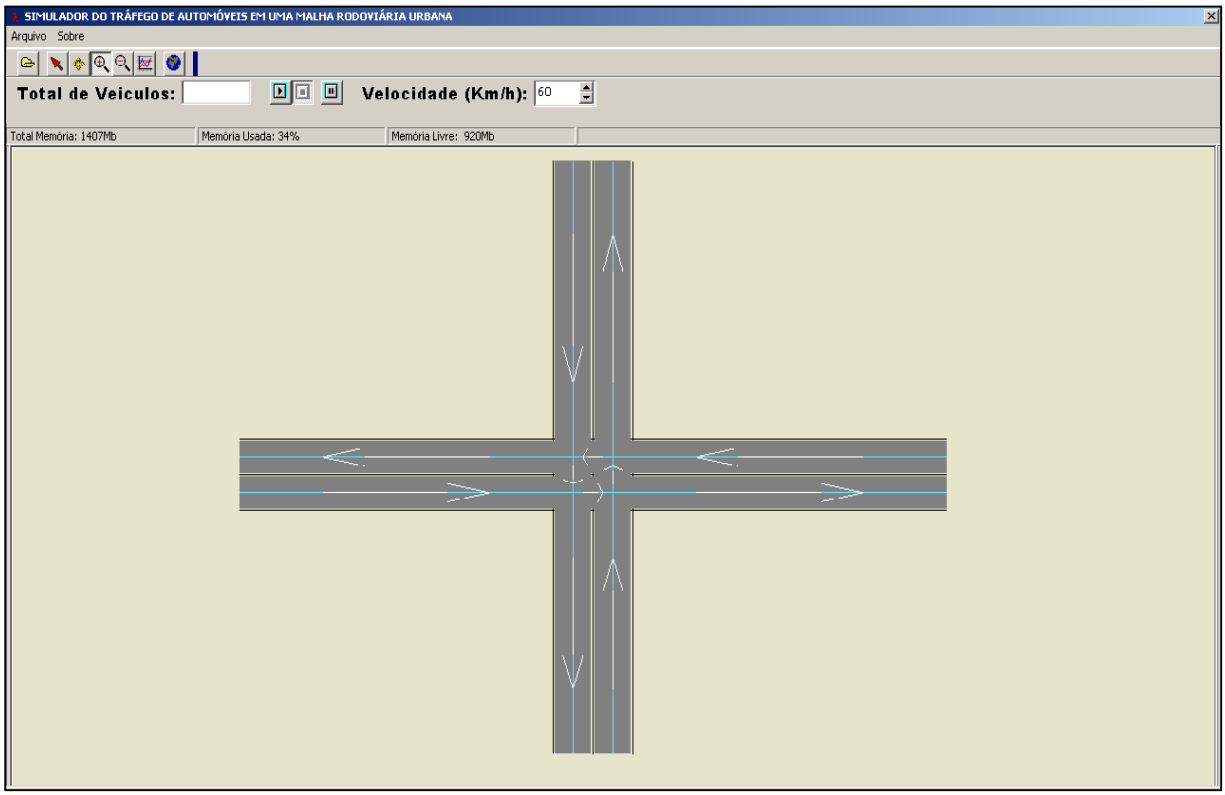


Figura 3 – Malha carregada

Anteriormente ao início da simulação é preciso informar a quantidade de veículos e opcionalmente ajustar a velocidade. A velocidade *default* (padrão indicado pelo simulador) é 60 km/h. O início da simulação dá-se após clicar no botão `Executar Simulação` (Figura 4).

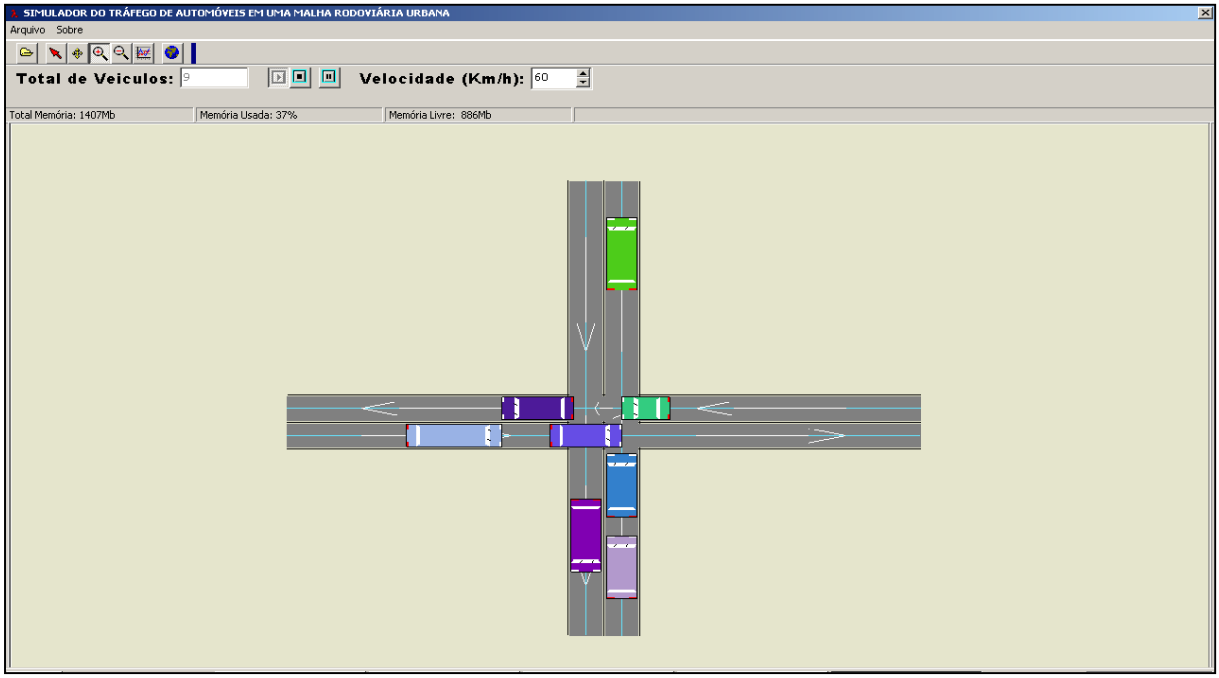


Figura 4 – Simulação de veículos trafegando

Para calcular a distância entre o inicio e o fim de um trecho, o simulador possui a funcionalidade `Calcular comprimento da Rua`, que exibe o comprimento de uma reta

traçada pelo usuário sobre a representação gráfica do simulador (Figura 5).

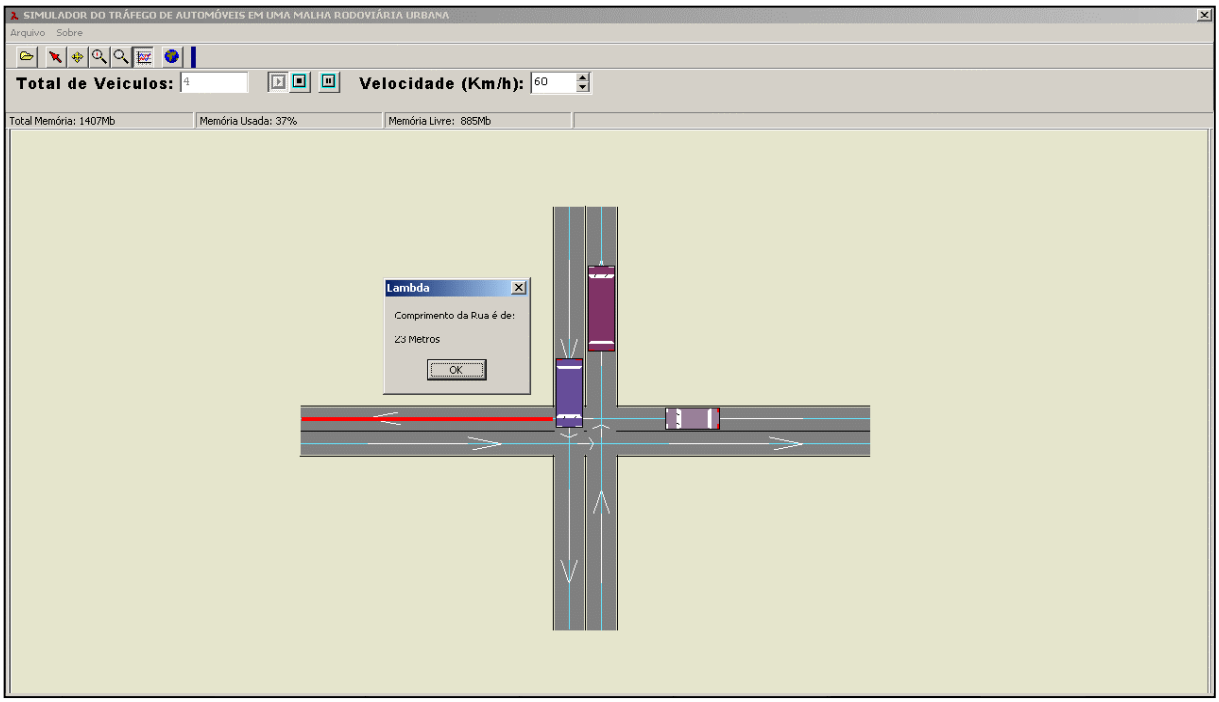


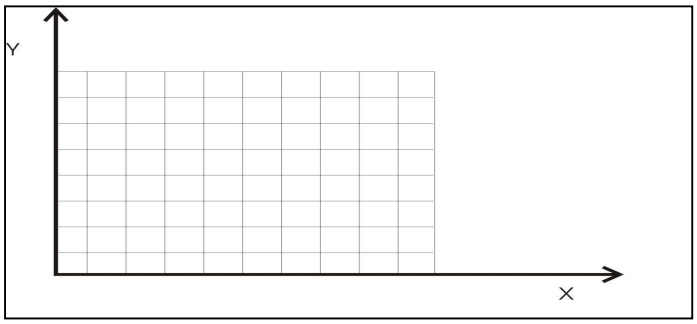
Figura 5 – Comprimento de uma rua

2.3 ROTINAS MATEMÁTICAS USADAS EM SISTEMAS DE REPRESENTAÇÃO DE MALHAS VIÁRIAS

Segundo Persiano e Oliveira (1989, p. 8), “Toda imagem criada através de recursos computacionais deve, em última instância, ser representada em algum meio físico que permita sua visualização.” Este meio físico pode ser, por exemplo, um dispositivo de vídeo ou uma impressora.

Nestes dispositivos a imagem é feita por meio de um conjunto de *pixels* que podem individualmente assumir posições sobre uma grade regular na qual pode-se definir atributos visuais. Para estabelecer um endereço a cada *pixel*, de forma que possam ser alterados seus atributos, existe uma organização que sugere um sistema cartesiano de coordenadas (figura 6) (PERSIANO; OLIVEIRA, 1989, p. 24,26).

Com a representação de um plano cartesiano é possível desenhar qualquer objeto no dispositivo gráfico, somente preenchendo as coordenadas *x* e *y*. Para manipular os pares ordenados no plano, o estudo da matemática nas áreas de geometria e trigonometria é essencial.



Fonte: Persiano e Oliveira (1989, p. 27).
Figura 6 – Sistema de coordenadas do espaço de exibição

A seguir são descritas as rotinas matemáticas utilizadas na representação gráfica do simulador implementado por Freire (2004), acrescentando-se a terceira dimensão.

2.3.1 Geometria Analítica

A geometria analítica, também chamada de geometria de coordenadas e que antigamente recebia o nome de geometria cartesiana, é o estudo da geometria através da álgebra. Em geral, é usado o sistema de coordenadas cartesianas para manipular equações para planos, retas, curvas e círculos, geralmente em duas dimensões, mas por vezes também em três ou mais dimensões (GONÇALVES, 1969, p. 5).

Com a geometria pode-se obter a distância entre duas coordenadas em um plano cartesiano aplicando a fórmula $dab = \sqrt{(Xb - Xa)^2 + (Yb - Ya)^2 + (Zb - Za)^2}$; onde *a* e *b* são duplas de coordenadas do ponto cartesiano. Para obter-se o ponto médio de uma reta usa-se a fórmula $X_m = (Xa + Xb)/2$ para as coordenadas *X*, $Y_m = (Ya + Yb)/2$ para as coordenadas *Y* e $Z_m = (Za + Zb)/2$ para as coordenadas *Z*, nos pontos extremos de uma reta (VENTURE, 2005, p. 37-39).

2.3.2 Trigonometria

Trigonometria (do grego *trigonon*=três ângulos e *metro*=medida) é o ramo da matemática que estuda as relações em triângulos, ângulos e funções trigonométricas como seno e cosseno (PINTO, 1958, p. 1-2).

A trigonometria surgiu como uma ferramenta matemática prática para determinar distâncias que não podiam ser medidas. Serviu de apoio à navegação, à agrimensura e à astronomia. Com a evolução da representação tridimensional a trigonometria auxiliou estudos de física, química e engenharia (COSTA, 1999).

Aplicando os estudos de distância entre pontos e ponto médio da geometria analítica juntamente com relações entre ângulos de uma ou mais retas da disciplina de trigonometria é possível representar retas paralelas em um plano cartesiano. Esta técnica é usada em Freire (2004) para calcular os pontos que formam uma rua na malha rodoviária.

2.4 COMPUTAÇÃO GRÁFICA

"A computação gráfica é a área da ciência da computação que estuda a geração, manipulação e interpretação de imagens por meio de computadores" (PERSIANO; OLIVEIRA, 1989, p. 3).

Pode-se dividir a computação gráfica em três grandes áreas (PERSIANO; OLIVEIRA, 1989, p. 3): (a) síntese de imagens que se ocupa da produção de representações visuais através de especificações geométricas; (b) processamento de imagens e; (c) análise de imagens que busca obter a especificação de uma imagem a partir de sua representação visual.

A síntese de imagens é a área mais ampla da computação gráfica, muitas vezes até confundida com a própria computação gráfica. As transformações geométricas estão presentes em inúmeras aplicações gráficas, desde programas de representação de *layouts* de circuitos até simuladores de movimentos, incluindo formas de animação (PERSIANO; OLIVEIRA, 1989, p. 43).

Uma aplicação gráfica pode ser desenvolvida em qualquer linguagem de programação com o auxílio de bibliotecas gráficas. Estas bibliotecas gráficas auxiliam o desenvolvedor na criação de objetos gráficos, pois possuem pacotes de rotinas específicas para editoração de imagens.

2.4.1 Biblioteca gráfica OpenGl

OpenGl é definida como um programa de interface para *hardware* gráfico. Na

> verdade, OpenGl é uma biblioteca de rotinas gráficas e de modelagem, bi (2D) e tridimensional (3D), extremamente portátil e rápida. Usando OpenGl é possível criar gráficos 3D com uma qualidade visual próxima de um *ray tracer*[2]. Entretanto, a maior vantagem de sua utilização é a rapidez, uma vez que usa algoritmos cuidadosamente desenvolvidos e otimizados pela *Silicon Graphics, Inc.*, líder mundial em Computação Gráfica e Animação. (MANSSOUR, 2003).

A OpenGl não é uma linguagem de programação, mas sim uma *Application Programming Interface* (API) para desenvolvimento de aplicações gráfica 2D e 3D. Um programa que é baseado em OpenGl significa que em suas rotinas há chamadas a uma ou mais bibliotecas OpenGl (MANSSOUR, 2003).

"Além do desenho de primitivas gráficas, tais como linhas e polígonos, OpenGl dá suporte a iluminação, colorização, mapeamento de textura, transparência, animação, entre muitos outros" (MANSSOUR, 2003). Atualmente OpenGl é reconhecida e aceita como um padrão de API para desenvolvimento de aplicações gráficas 3D.

Quando se trabalha com representações de objetos tridimensionais tem-se que mapear as representações 3D para imagens que possam ser exibidas em um dispositivo como um monitor, que somente exibe imagens bidimensionais. Para poder haver esse mapeamento dos objetos 3D para representações 2D é utilizada a operação de projeção (COHEN; MANSSOUR, 2006, p. 79).

Existem dois tipos principais de projeções: "**Projeção Paralela Ortográfica**: as projetantes são paralelas entre si, passam pelos pontos que definem os objetos e interseccionam o plano com um ângulo de 90º" (COHEN; MANSSOUR, 2006, p. 79, grifo do autor) e "**Projeção Perspectiva**: as projetantes emanam de um único ponto que está a uma distância finita do plano de projeção e passam pelos pontos que definem os objetos." (COHEN; MANSSOUR, 2006, p. 80, grifo do autor).

> Na projeção ortográfica [...] não há alteração nas medidas do objeto. Sua construção é bastante simples, pois basicamente consiste em omitir uma das componentes de cada vértice. Entretanto, a projeção perspectiva é mais utilizada, uma vez que representa melhor o que acontece na realidade. Por exemplo, se dois objetos possuem o mesmo tamanho e estão posicionados a diferentes distâncias do plano de projeção, o que está mais longe vai aparecer menor do que o objeto que está próximo. (COHEN; MANSSOUR, 2006, p. 80).

Outro fator importante na visualização de cenas 3D é a especificação do observador virtual. Segundo Cohen e Manssour (2006, p. 78) "este observador define de que local deseja-se que a cena 3D seja exibida, por exemplo, de cima, do lado direito ou do esquerdo." O observador virtual é necessário, visto que no universo 3D a cena vista de diferentes lugares tem, para quem a observa, uma percepção diferente a cada posição (COHEN; MANSSOUR,

2 É uma técnica de renderização de imagens fotorealísticas, cujos algoritmos utilizam as propriedades ópticas das luzes, objetos e materiais.

2006, p. 78).

Além de configurar o tipo de projeção e a posição da câmera virtual para exibição de cenas 3D, outro fator importante é descobrir qual objeto vai ser desenhado na *viewport* [3] e qual vai ficar oculto, ou seja, como desenhar objetos que estão um na frente do outro.

A OpenGl implementa o algoritmo denominado *z-buffer*, que segundo Cohen e Manssour (2006, p. 191-192) "é bastante simples: dois *buffers* do tamanho da janela de exibição são utilizados, um para guardar valores de profundidade (chamado de *z-buffer*) e outro para guardar valores de cor (denominado *color buffer*)."

> O z-buffer é inicializado com o maior valor de profundidade possível e o color buffer é inicializado com a cor de fundo da imagem. Depois, cada face projetada é percorrida, pixel por pixel, e o valor de profundidade de cada pixel é comparado com o valor já armazenado no z-buffer: se este pixel estiver mais próximo do observador, coloca-se seu valor de profundidade no z-buffer e seu valor de cor no color buffer. Assim, no final do processamento o color buffer irá conter a imagem final. (COHEN; MANSSOUR, 2006, p. 192).

2.5 PROCESSOS CONCORRENTES

O estudo de processos concorrentes é utilizado para a definição dos veículos que circulam na malha rodoviária. Cada veículo representa um processo concorrente em execução.

Toscani, Oliveira e Carissimi (2003, p. 13) dizem que, "um processo é um programa em execução, o qual é constituído por uma seqüência de instruções, um conjunto de dados e um registro descritor." O processador somente é capaz de executar um processo de cada vez e quando há mais de um processo sendo executado na máquina diz-se que esses processos estão sendo executados de forma concorrente.

> A programação concorrente, além de ser intelectualmente desafiante e ser essencial ao projeto de sistemas operacionais, também tem importantes aplicações práticas. Na verdade, ela tem aplicação natural nos sistemas que precisam atender requisições que ocorrem de forma imprevisível. Exemplos de aplicações desse tipo são sistemas para controle on-line de informações (contas bancárias, estoques de supermercados, etc) e controle automático de processos externos (processos industriais, por exemplo). (TOSCANI; OLIVEIRA; CARISSIMI, 2003, p. 16).

A definição de processos concorrentes, ou programas concorrentes, não implica em criar um programa com trechos de instruções paralelas e mandar para o processador executá-las. Sabe-se que os recursos computacionais são compartilhados entre os processos e também variáveis que estão definidas no programa. Em determinado instante é possível que dois

3 Região limitada onde o desenho é exposto. É a área de visualização da imagem.

processos estejam tentando acessar a mesma variável. O primeiro processo pode ter lido um valor da variável e na próxima instrução ocorre uma interrupção por *time out*[4]. Este processo pára sua execução e imediatamente o outro processo entra em execução acessando a mesma variável. Neste ponto ocorre uma inconsistência nos dados, pois o conteúdo da variável estava sendo alterado pelo processo anterior, o qual não foi efetivamente realizado (TOSCANI; OLIVEIRA; CARISSIMI, 2003, p. 41-44). Nesta situação é imprescindível que exista um mecanismo de comunicação entre os processos.

Uma solução para evitar este problema é a definição de regiões críticas.

> Os trechos dos processos onde os dados compartilhados são acessados são denominados *trechos críticos*, *regiões críticas* ou *seções críticas.* A maneira de eliminar as condições de corrida de um programa concorrente é simples: basta garantir a exclusão mútua na execução dos trechos críticos dos processos. (TOSCANI; OLIVEIRA; CARISSIMI, 2003, p. 43).

Em um algoritmo denomina-se qual a região crítica e com uma variável global representa-se quando há algum processo nessa região. Essa variável pode ser do tipo *boolean*, logo, quando um processo entra na região crítica a variável é setada em estado falso, se outro processo tentar entrar na mesma região não terá permissão para entrar até que o estado da variável global mude para verdadeiro, e isto só ocorrerá quando o primeiro processo estiver saindo da região crítica, onde a variável global vai ser definida como verdadeira (TOSCANI; OLIVEIRA; CARISSIMI, 2003, p. 43-44).

Nos programas concorrentes os recursos do computador são requisitados ininterruptamente, e quando não forem tomados alguns cuidados especiais quanto à alocação, os processos podem entrar em *deadlock*[5].

Os semáforos são mecanismo de comunicação e de sincronização entre processos que permitem, simultaneamente, controlar o acesso a recursos em modo concorrente, com exclusividade, e até mesmo em modo cooperado, onde há mais de um processo que pode acessar o recurso (SEBESTA, 2000, p. 475).

As principais operações sobre semáforos são a inicialização, que recebe o número de processos que podem acessar um determinado recurso. A operação de *wait* ou *P*, decrementa o valor do semáforo. Caso seja zero, o processo espera a operação *V*. Na operação *signal* ou V, se o semáforo estiver em zero, o processo que estiver mais tempo esperando pelo recurso será acordado. Caso contrário o valor do semáforo é incrementado (SEBESTA, 2000, p. 476).

---

4 A faixa de tempo para o processo ser executado chegou ao limite, chegou a vez de passar o processador a outro processo.

5 Caracteriza uma situação em que ocorre um impasse e o Sistema Operacional fica impedido de continuar suas atividades. Esse impasse pode ser a espera de algum recurso que um processo está usando e que por sua vez está

A terminologia *P* e *V* pode-se atribuir a Dijkstra que denominou as operações com as iniciais dos verbos em holandês de *proberan* (testar) e *verhoegen* (incrementar) (SEBESTA, 2000, p. 475).

Para um bom desenvolvimento de sistemas com processos concorrentes deve-se denominar a região crítica o mais pequeno possível e de execução rápida, para minimizar a espera dos demais processos concorrentes (SEBESTA, 2000, p. 472).

2.6 TRABALHOS CORRELATOS

A seguir são apresentados três trabalhos que tratam do controle de trânsito, os quais são: Sistemas Multiagentes para Gerenciamento de Tráfego Urbano (SILVA, 2005); Simulação de Tráfego: uma Experiência com Realidade Virtual (BARROS; KELNER, 2003) e Definição de um Sistema de Software para Simulação e Supervisão de Tráfego Viário Urbano (ZANUZ, 1998).

2.6.1 Sistemas multiagentes para gerenciamento de tráfego urbano

O objetivo deste trabalho (Figura 7) é o desenvolvimento de um sistema inteligente de gerenciamento de trânsito, com três funções fundamentais: (a) gerenciar os recursos necessários ao controle do tráfego; (b) ajudar no treinamento dos funcionários das companhias de trânsitos; e (c) acompanhar as mudanças do tráfego urbano para fins de tomada de decisão.

O sistema proposto por Silva (2005) tem como característica a utilização de interação entre agentes, provendo desta forma um ambiente dinâmico de funcionamento. O agente é uma entidade capaz de interagir com o usuário para receber tarefas e ao mesmo tempo tem autonomia de funcionamento sem intervenção direta. Possui inteligência para interpretar os eventos monitorados e tomar decisões apropriadas para a operação de autonomia.

A proposta permite que sejam feitas simulações, onde atividades de treinamento e

---

esperando algum recurso que outro processo possuí, formando um círculo onde um processo está esperando pelo outro.

melhorias possam ser analisadas virtualmente.

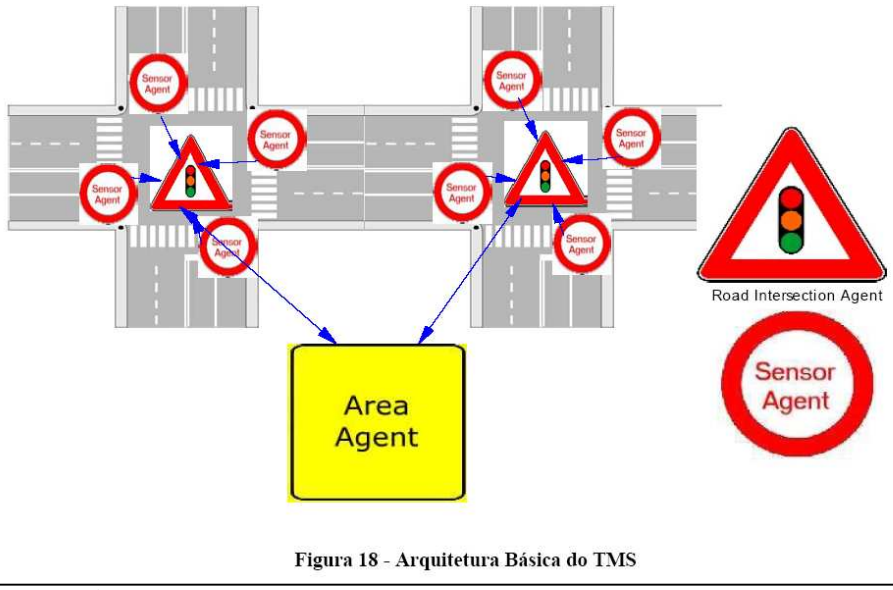


Figura 18 - Arquitetura Básica do TMS

Fonte: Silva (2005, p. 71).

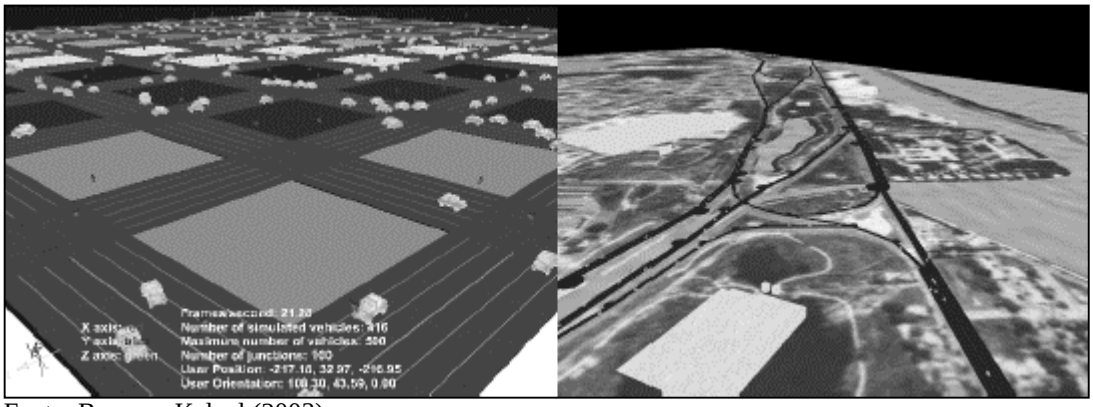
Figura 7 – Sistema multiagente para gerenciamento de tráfego urbano

2.6.2 Simulação de tráfego: uma experiência com realidade virtual

O protótipo desenvolvido por Barros e Kernel (2003) (Figura 8) tem por objetivo testar a viabilidade de implementar um simulador utilizando dispositivos de realidade virtual imersiva (envolvimento e interação do usuário com a realidade virtual), construindo um modelo para simulação de alagamento em rodovias e seu efeito sobre o fluxo de veículos.

A realidade virtual tem sido muito empregada em diversos setores industriais, permitindo a manipulação de objetos de dados complexos através de modelos tridimensionais. No mesmo sentido, uma aplicação envolvendo simulação de tráfego rodoviário pode ser mais produtiva se for expressa de forma tridimensional, tornando o planejamento urbano mais abrangente e com um grau de certeza de aplicabilidade maior.

Para a definição e adaptação das vias de tráfego das regiões no simulador foi utilizada a ferramenta de modelagem gráfica 3D *Studio Max*. As pistas foram extraídas de uma ortofotocarta manualmente, utilizando funcionalidades da ferramenta de edição de imagens *CorelPhotopaint*. Após, foi ajustado o relevo da pista através de operações entre as pistas e a superfície do terreno.



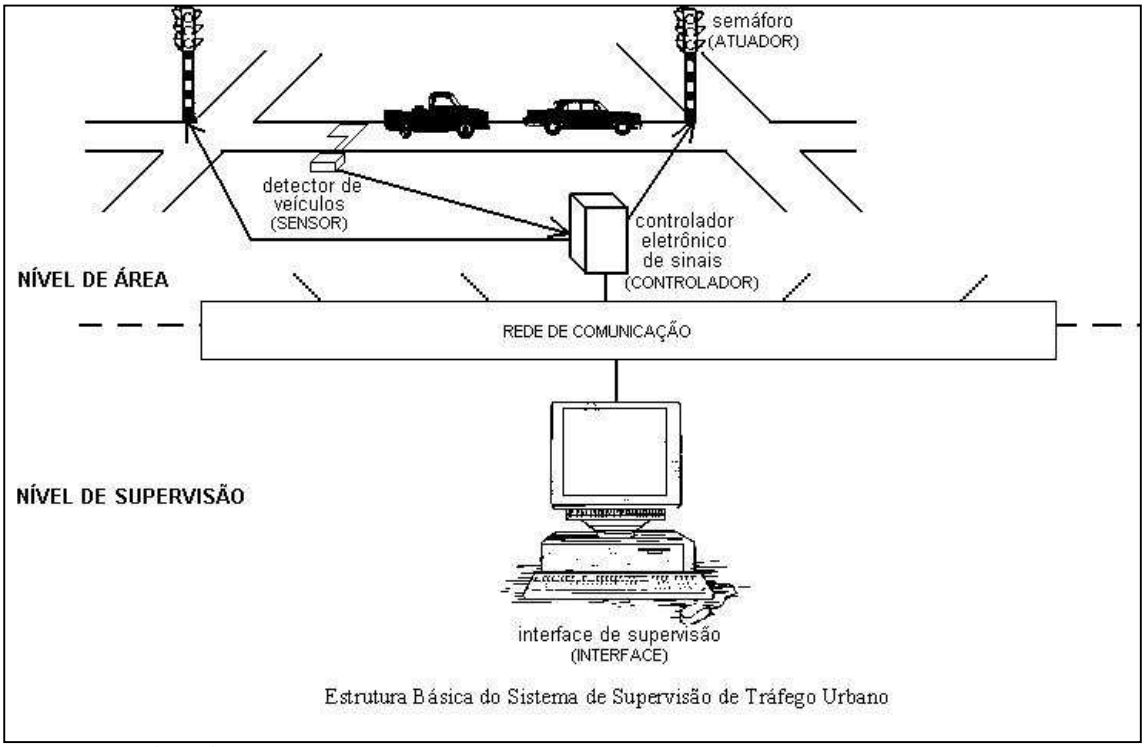
Fonte: Barros e Kelnel (2003).

Figura 8 – Simulação de tráfego: uma experiência com realidade virtual

2.6.3 Definição de um sistema de software para simulação e supervisão de tráfego viário urbano

Zanuz (1998) faz uma apresentação de uma estrutura de sistema de supervisão aplicável ao tráfego veicular urbano. O sistema proposto por Zanuz (1998) compreende um sistema distribuído hierárquico, usando elementos de controle comerciais disponíveis como semáforos luminosos e sensores de posicionamento e de velocidade.

Este sistema distribuído agrega controladores eletrônicos de sinais de tráfego, redes de comunicação, detectores de veículos e atuadores. Os controladores eletrônicos de sinais são aparelhos com microprocessadores que realizam o controle e sincronismo entre os semáforos. Os detectores de veículos são laços indutivos instalados abaixo da superfície das vias, que são usados juntamente com os controladores eletrônicos de sinais, evitando que ruas sem veículos estejam livres em cruzamentos, enquanto que ruas com veículos estejam fechadas no mesmo cruzamento (Figura 9).



Fonte: Zanuz (1998).

Figura 9 – Definição de um sistema de software para simulação e supervisão de tráfego viário urbano

3 DESENVOLVIMENTO DO TRABALHO

A seguir são feitas considerações sobre a versão 1.0 do Simulador de Tráfego de Automóveis em uma Malha Rodoviária (Freire, 2004). Logo após, são descritos os requisitos do sistema assim como também toda a sua especificação. Na seção posterior é explanada a implementação executada para atingir os objetivos deste trabalho. Em seguida é apresentada a operacionalidade do software desenvolvido e também uma discussão sobre procedimentos adotados no desenvolvimento.

3.1 ANÁLISE DA VERSÃO 1.0 DO SIMULADOR DE TRÁFEGO DE AUTOMÓVEIS EM UMA MALHA RODOVIÁRIA

Feita uma análise no Simulador de Tráfego de Automóveis em uma Malha Rodoviária (Freire, 2004) observou-se que não é possível definir uma rota específica para os automóveis, ou ao menos para um automóvel, na tentativa de fazer rotas com caminhos preestabelecidos para a verificação de regiões específicas da malha rodoviária.

Observa-se que em determinadas malhas rodoviárias há viadutos, objetivando dar uma maior eficiência ao trânsito dos veículos. Situações como estas não são possíveis de serem simuladas na versão 1.0 do Simulador de Tráfego de Automóveis em uma Malha Rodoviária.

Na interseção dos trechos, por exemplo em um cruzamento, o tamanho do veículo é diminuído. Isto ocorre devido a um erro de especificação/implementação no software (Figura 10).

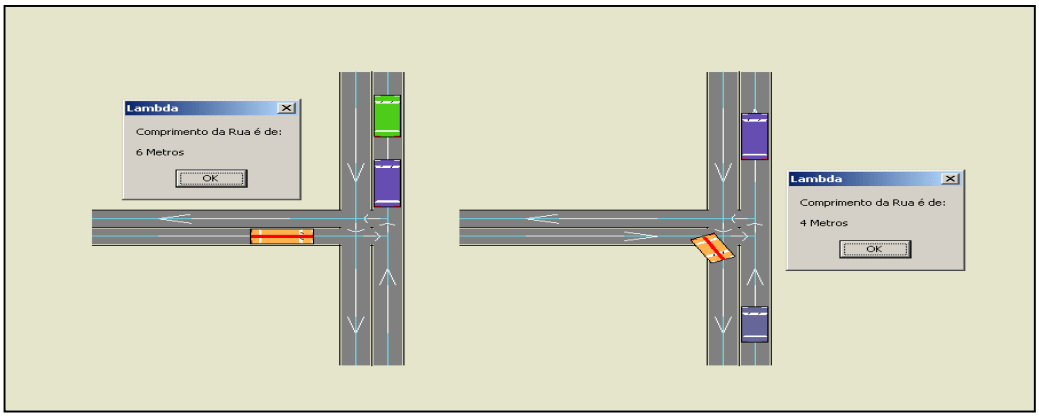


Figura 10 – Tamanho do veículo em um cruzamento

Analisando o código, verificou-se uma baixa legibilidade na função que determina o

próximo trecho que o automóvel deverá percorrer (nos casos em que há mais de uma opção, a determinação ocorre de forma aleatória). A determinação da baixa legibilidade foi definida após análise (leitura) da função por mais de uma vez e dúvidas sobre a mesma ainda persistiam.

3.2 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos do trabalho proposto são:

a) o sistema deverá permitir que um usuário controle a direção de um carro em tempo de execução (Requisito Funcional - RF);
b) o sistema deverá permitir a visualização do mapa rodoviário em três dimensões (RF);
c) o sistema deverá ser implementado no ambiente de programação Delphi 7 e usar a biblioteca gráfica OpenGl (RNF).

3.3 ESPECIFICAÇÃO

Neste trabalho a especificação e a implementação são desenvolvidos em paralelo, uma vez que é preciso analisar o código desenvolvido por Freire (2004) e implementar novas funcionalidades. O código procedural antigo é reescrito em código voltado para orientação a objetos. Os diagramas de casos de uso, atividades e modelo de classes foram produzidos com o auxílio da ferramenta Enterprise Architect, descrevendo-os através da *Unified Modeling Language* (UML).

No diagrama de casos de uso (Figura 11) são mostradas as ações de carregar mapa, aplicar efeito de *zoom in* e *zoom out*, calcular comprimento de trecho, setar tamanho *default* do mapa, parar, pausa e executar a simulação. Estas ações já estavam presentes na versão 1.0 do simulador de tráfego.

Nesta nova versão (2.0) é incluída a ação de movimenta carro guiado (em destaque na

Figura 11).

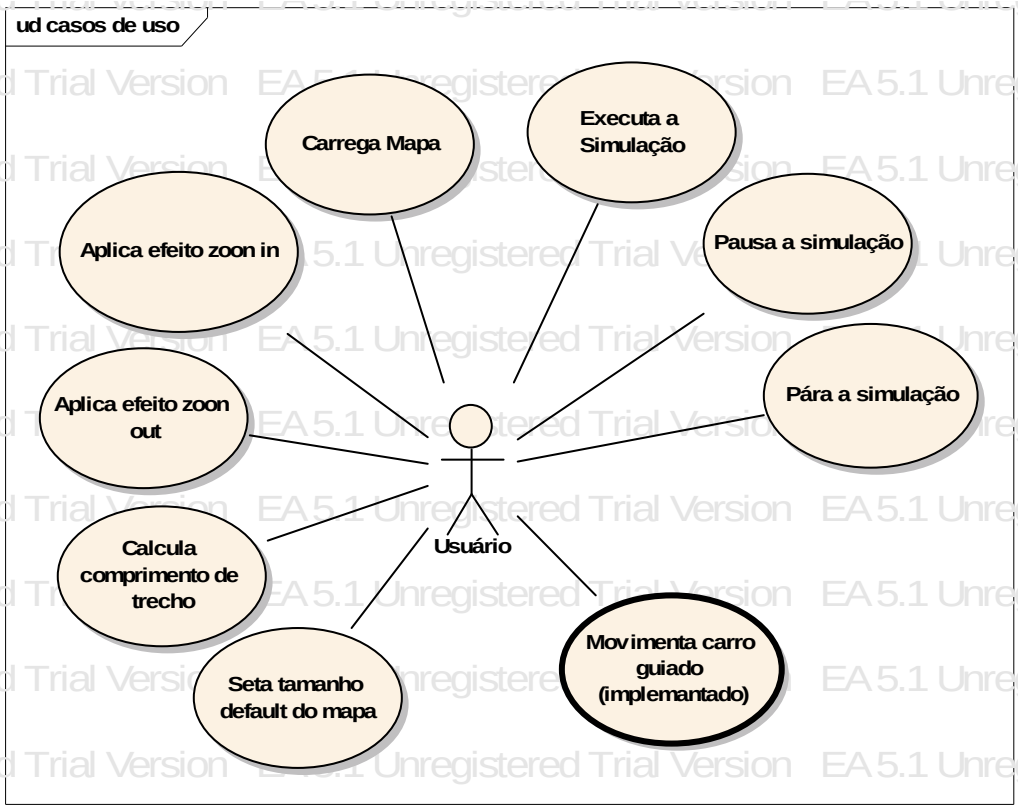


Figura 11 – Casos de usos

No Quadro 1 é descrito o cenário para o caso de uso `Movimenta carro guiado`.

Pré Condição: a função Carro Guiado deverá estar habilitada. Pós Condição: o veículo segue a rota escolhida pelo usuário.
movimenta carro cenário - Principal 1. O simulador exibe carro guiado parado em um cruzamento. 2. O usuário define uma nova rota para o carro guiado. 3. O usuário pressiona a tecla "up" confirmando a escolha da nova rota para o carro guiado. 4. O simulador exibe o carro guiado movimentando-se na rota escolhida pelo usuário.
seleção de trechos a direita - cenário Alternativo No passo 2 do cenário principal. 2.1. O usuário pressiona a tecla "rigth" para selecionar um trecho a direita. 2.2. O simulador exibe um retângulo sobre o trecho escolhido. 2.3. Retorna para o passo 3 do cenário principal.
seleção de trechos a esquerda - cenário Alternativo No passo 2 do cenário principal. 2.1. O usuário pressiona a tecla "left" para selecionar um trecho a esquerda. 2.2. O simulador exibe um retângulo sobre o trecho escolhido. 2.3. Retorna para o passo 3 do cenário principal.
seleção de trechos dentro de um cruzamento - cenário de Exceção No passo 3 do cenário principal se o último trecho escolhido pelo usuário ainda estiver dentro de um cruzamento. 3.1. O usuário escolhe um trecho de continuação para o trecho definido anteriormente. 3.2. O simulador exibe um retângulo sobre o trecho escolhido. 3.3. Retorna para o passo 3 do cenário principal.

Quadro 1 – Descrição do cenário do caso de uso `Movimenta carro guiado`

O diagrama de atividades (Figura 12) demonstra a interação do aplicativo com o usuário. Seguindo o diagrama de atividades, o usuário quando inicia a aplicação deve carregar a malha. Com a malha carregada pode-se calcular o comprimento de trechos ou definir a quantidades de carros que irão circular na malha rodoviária. Em seguida o usuário pode iniciar, pausar, continuar ou parar a simulação. Quando iniciada a simulação, o usuário pode habilitar ou desabilitar a função de carro guiado através de um *checklist* e através das setas direcionais do teclado movimentar o veículo para um trecho determinado.

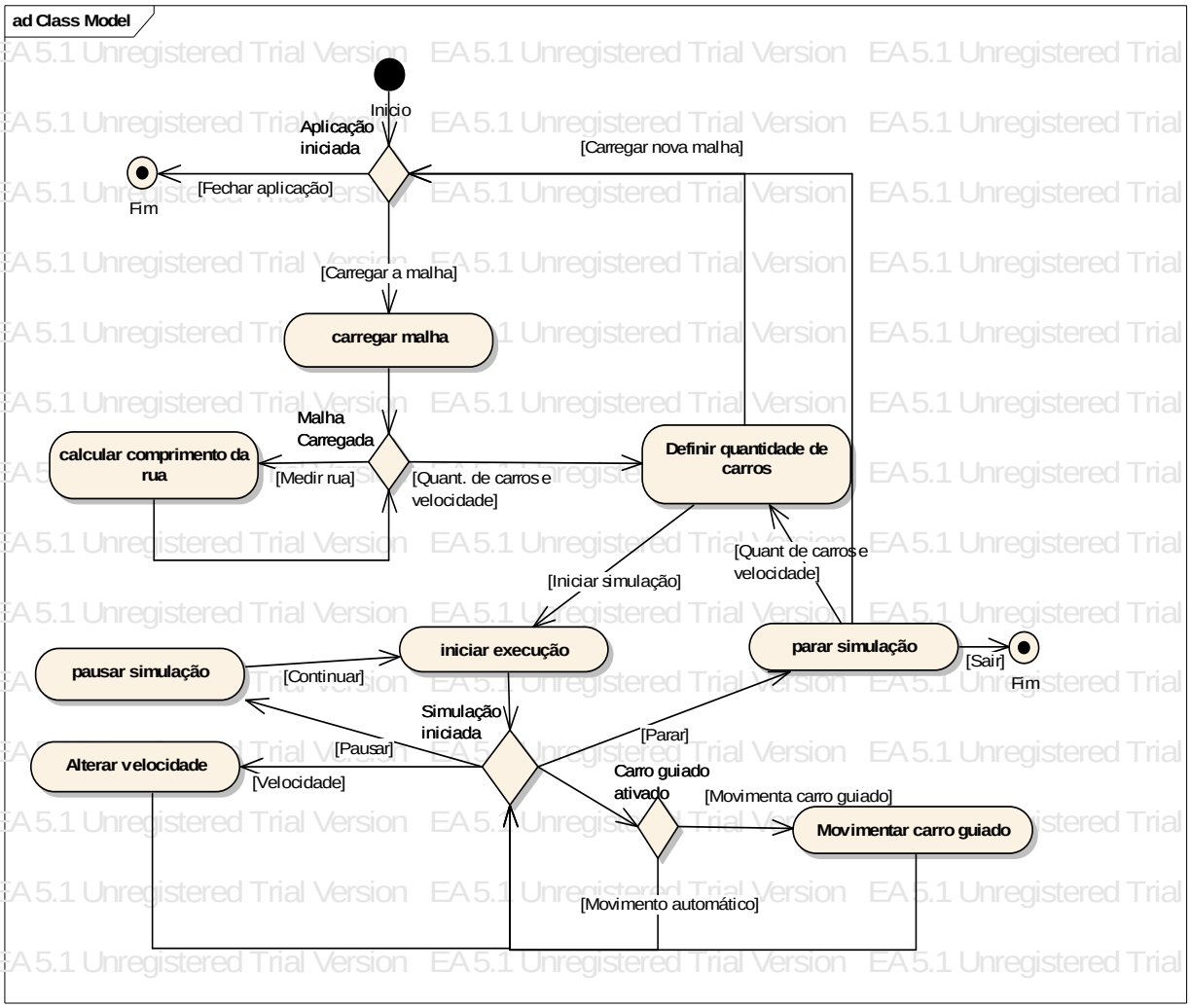


Figura 12 – Diagrama de atividades

No diagrama de classes (Figura 13) estão representadas as classes envolvidas no desenvolvimento do simulador de tráfego de automóveis em uma malha rodoviária, versão 2.0. A superclasse `Carro` implementa atributos e métodos que são herdados pelas classes `CarroGuiado` e `CarroRobo.` A classe `Malha` agrega os veículos que circulam na malha e também os trechos e ruas que formam a malha rodoviária. As classes `Trecho` e `Rua` são usadas para definir as características e operações sobre os trechos que compõem a

malha rodoviária.

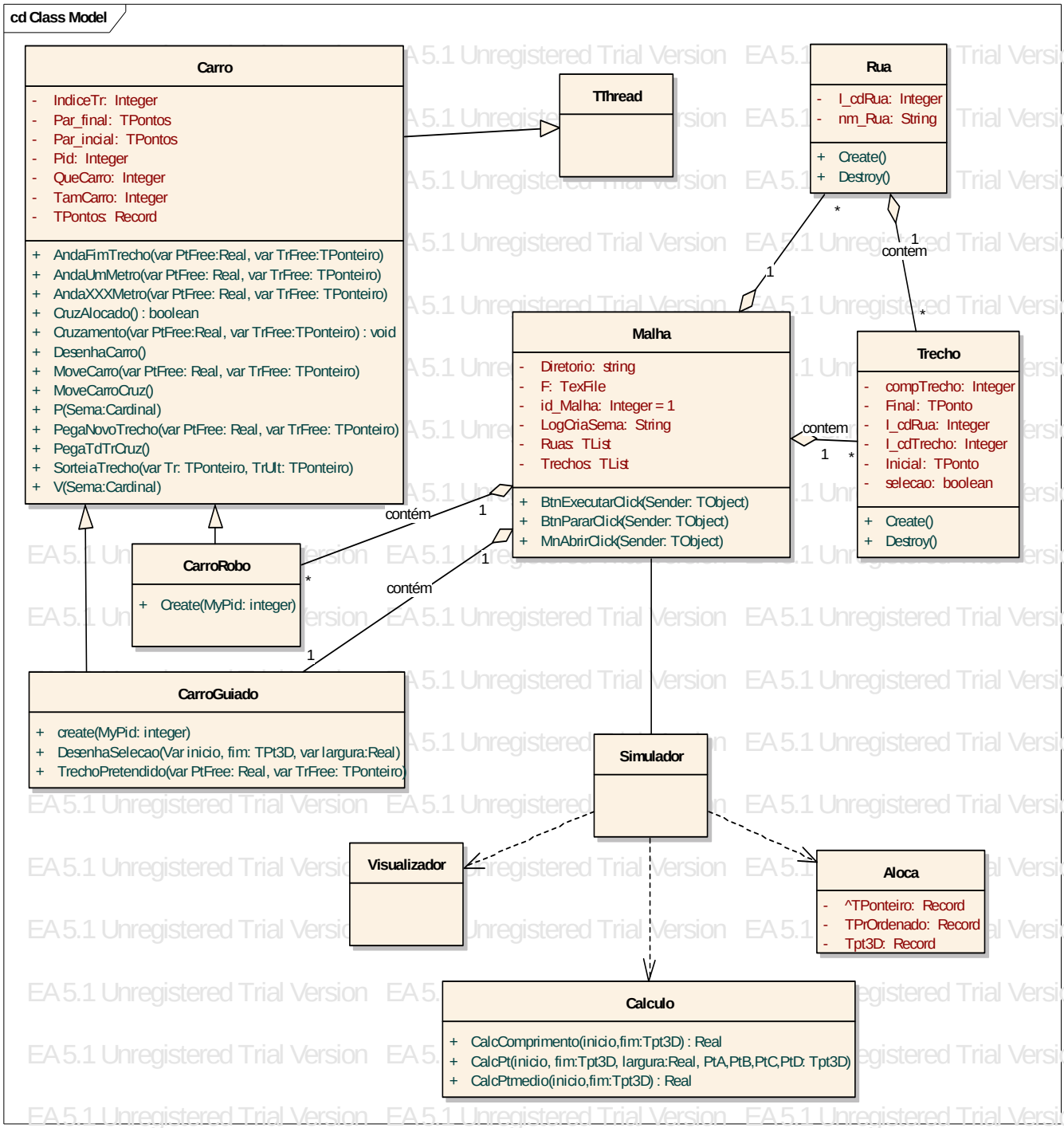


Figura 13 – Diagrama de classes

No Apêndice A os métodos das classes `Carro`, `CarroGuiado`, `CarroRobo` e `Malha` são descritos.

A movimentação dos veículos na malha rodoviária é controlado pelo método `MoveCarro()`, da superclasse `Carro` e quando os veículos encontram o final de um trecho e o mesmo possui uma continuação em um cruzamento, o método `Cruzamento()` é ativado.

O método `Cruzamento()` pode ser dividido em três etapas: a) primeiro é chamado o

método `PegaTdTrCruz()` que possui as rotinas `SorteiaTrecho()` e `TrechoPretendido()` utilizadas para a escolha de um caminho disponível de continuação; b) o método `CruzAlocado()` que verifica se existe algum carro trafegando dentro do cruzamento e se existir fica aguardando até que seja liberado o cruzamento; e c) o método `MoveCarroCruz()` que faz o movimento do veículo dentro do cruzamento.

Na Figura 14 é mostrado o diagrama de seqüência para a movimentação do veículo robô.

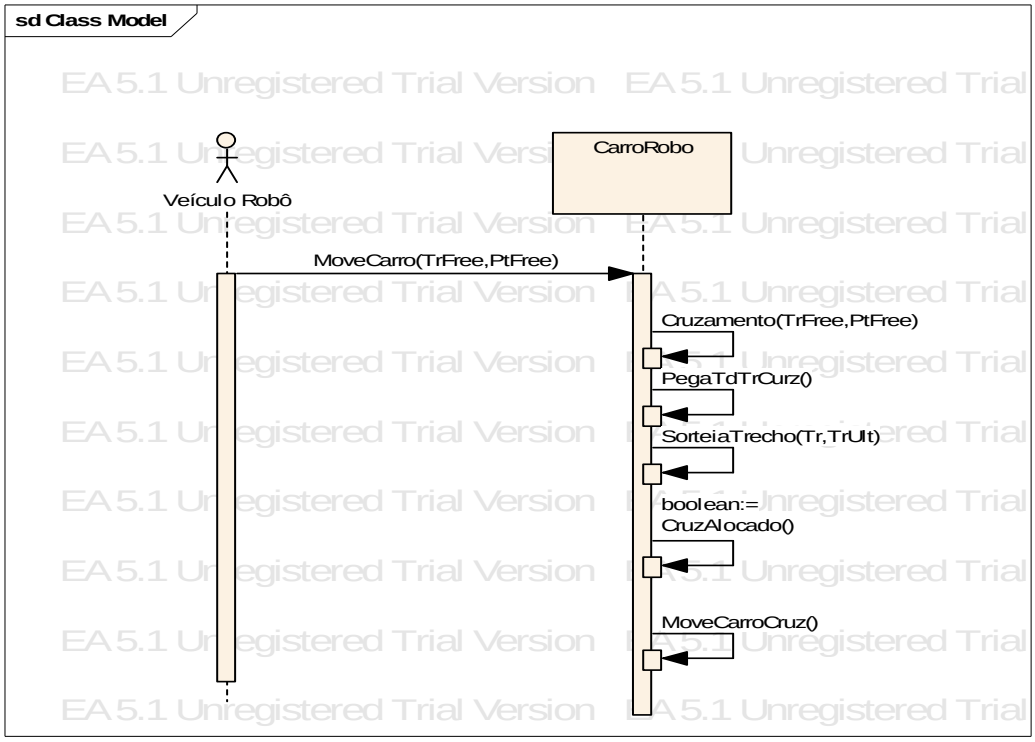


Figura 14 – Diagrama de seqüência para o veículo robô

Durante a simulação é possível ativar ou desativar o veículo guiado. Caso o botão de ativação do veículo guiado esteja desmarcado, todos os veículos trafegam aleatoriamente na malha rodoviária. Enquanto o usuário mantiver pressionada a ativação do carro guiado, o método `PegaTdTrCruz()` chama o método específico da classe `CarroGuiado`.

A classe `CarroGuiado` é responsável pela especificação do veículo que o usuário poderá guiar durante a execução da simulação. O método desta classe, `TrechoPretendido()`, implementa a rotina que permite que o usuário escolha uma direção para o veículo. Nesta rotina há um laço que fica aguardando que o usuário pressione a tecla *up* do teclado, que inicia o movimento do carro, e com as teclas *left* e *rigth* são selecionadas as direções. Enquanto o usuário escolhe a direção do veículo, outro método, `DesenhaSelecao()`, também da classe `CarroGuiado`, identifica para o usuário qual trecho está sendo selecionado através de um retângulo que fica destacado sobre o trecho escolhido.

Na Figura 15 é mostrado o diagrama de seqüência para a movimentação do veículo guiado pelo usuário.

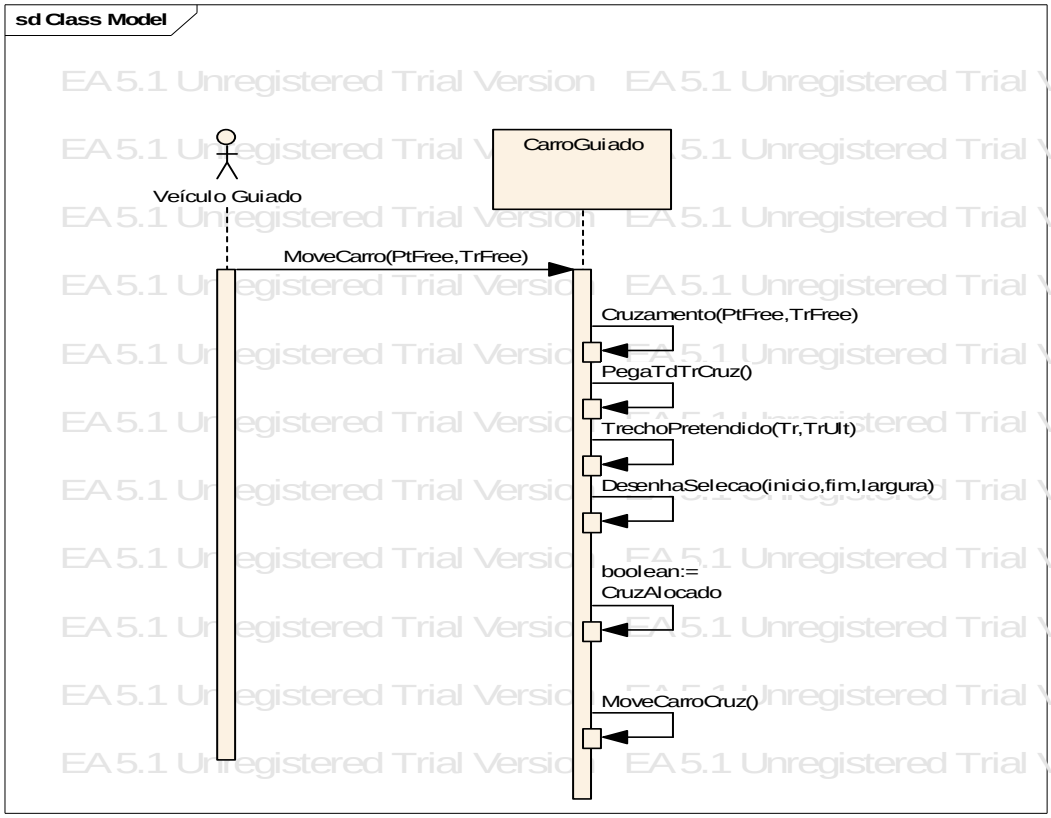


Figura 15 – Diagrama de seqüência para o veículo guiado

Para os demais veículos, e também para o carro guiado quando não está ativada a opção de `Carro Guiado`, que circulam na malha rodoviária, o método `SorteiaTrecho()`, da superclasse `Carro`, define aleatoriamente os trechos disponíveis de continuação.

No sorteio do trecho de continuação, em Freire (2004), são especificados critérios para restringir o acesso dos veículos em cruzamento com rotatória, para evitar que o veículo faça o retorno proibido ou fique andando em círculos.

O arquivo texto que é usado para carregar a malha, descrito em Freire (2004), possui informações que permitem controlar o acesso dos veículos nos trechos (Quadro 2). Os atributos especificados no Quadro 2 são:

a) Xi: ponto inicial em `x` do trecho;
b) Yi: ponto inicial em y do trecho;
c) Zi: ponto inicial em z do trecho;
d) Xf: ponto final em x do trecho;
e) Yf: ponto final em y do trecho;
f) Zf: ponto final em z do trecho;
g) La: define a largura da via;

h) Me: define via na faixa de rolamento da esquerda;

i) Md: define via na faixa de rolamento da direita;

j) Ni: define o nível da via (não utilizado pelo simulador);

k) Qe: define quebra de carro;

l) Vq: define número de vezes que o veículo pode entrar nesta via;

m) Cd: define código de rua.

Xi	Yi	Zi	Xf	Yf	Zf	La	Me	Md	Ni	Qe	Vq	Cd
000	–03	0	025	-03	0	03	00	00	00	00	01	01
025	-03	0	028	-03	0	03	00	00	00	01	01	02
025	-03	0	025	-25	0	03	00	00	00	00	01	03
025	000	0	025	-03	0	03	00	00	00	01	00	04
028	-25	0	028	-03	0	03	00	00	00	00	01	05
028	-03	0	028	000	0	03	00	00	00	01	01	06
028	000	0	028	025	0	03	00	00	00	00	01	07
025	000	0	000	000	0	03	00	00	00	00	01	08

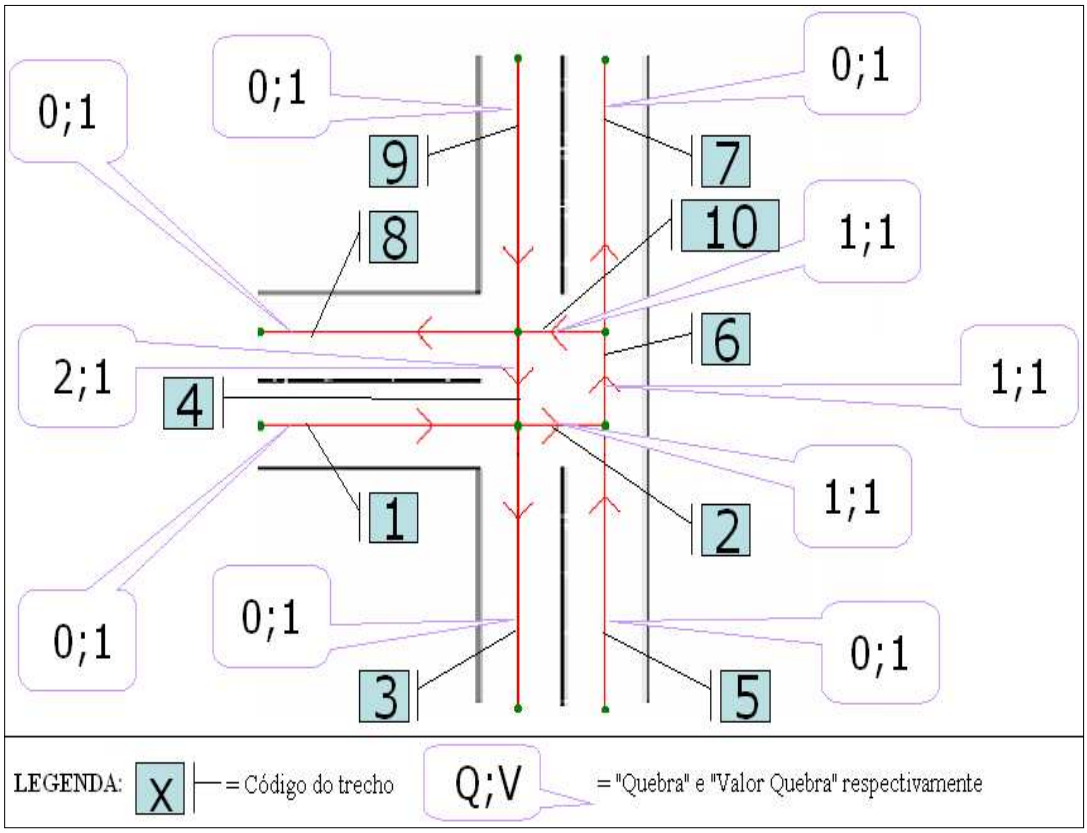
Quadro 2 – Arquivo texto com configuração da malha

> O atributo “quebra” do trecho e do carro e “valor de quebra” do trecho são usados para definir a quantidade de vezes que um carro pode circular em um cruzamento ou restringir a tomada de direção não permitida. Exemplo de tomada de direção não permitida cita-se o retorno de veículos em cruzamentos. No protótipo, onde a direção é aleatória, os carros poderiam ficar infinitamente andando em círculo [...]. Através da configuração dos atributos “quebra” e “valor de quebra” evita-se tal situação. (FREIRE, 2004, p. 28).

Os trechos possuem um valor de quebra que é especificado pelo criador da malha rodoviária e durante a inicialização do veículo. O valor de quebra de carro é atualizado para o valor de quebra do trecho de entrada na malha. Quando o veículo encontrar um cruzamento e um trecho foi escolhido, é verificado se o valor quebra deste novo trecho é menor que o valor quebra de carro. Se o valor quebra de trecho for menor, o veículo pode seguir neste novo trecho e o valor de quebra do novo trecho é somado ao valor de quebra do veículo. Se for escolhido outro trecho e este novo trecho possuir o valor quebra mais alto que o valor de quebra do veículo, significa que o veículo não pode trafegar neste trecho e terá que escolher novamente outro trecho até que satisfaça a condição. Quando o trecho escolhido tiver um valor quebra igual à zero, significando que o veículo está saindo do cruzamento, o valor de quebra de carro é atualizado com o valor de quebra do trecho de saída do cruzamento.

Conforme a Figura 16, deseja-se limitar o tráfego de veículos nos trechos cujo código de trecho são dois, quatro, seis e dez. O veículo que trafega pelo trecho de código cinco possui o valor de quebra de carro igual ao valor de quebra do trecho cinco (quebra de carro=quebra do trecho=0) e ao encontrar o trecho de código seis verifica se o seu valor de quebra de carro é menor que o valor quebra do trecho. Em caso positivo, o valor de quebra do trecho é somado com o valor de quebra do carro (quebra do carro=0 + quebra do trecho=1 =

novo valor quebra de carro=1). Para esta situação o veículo foi autorizado a trafegar pelo trecho de código seis e seu valor de quebra foi somado com o valor de quebra do trecho cinco. Logo a frente há outro cruzamento e o mesmo teste deverá ser feito. O veículo tem duas opções de rota, o trecho de código sete que possui o valor quebra igual a zero e o trecho de código dez que possui o valor quebra igual a um. Caso o veículo escolha o trecho de código dez é verificado se o valor de quebra do carro (igual a um) é menor ou igual ao valor de quebra do trecho (com valor um). O pedido de acesso ao trecho é autorizado é o valor de quebra do carro é somado com o valor de quebra do trecho de código dez (quebra do carro=1 + quebra do trecho=1 = novo valor quebra de carro=2). O veículo segue pelo trecho de código dez e ao final deste encontra mais duas opções de rota, o trecho de código oito e o trecho de código quatro. Se na escolha aleatória de trechos, o trecho de código quatro seja selecionado, o teste de quebra de carro irá verificar se o valor de quebra do carro (que é igual a 2) é menor ou igual ao valor quebra do trecho (que é igual a 2). O teste acusará que o veículo não pode trafegar neste trecho e outro trecho deve ser escolhido. Quando é escolhido um trecho com valor de quebra igual a zero, neste caso o trecho de código oito, o valor de quebra de trecho é atribuído ao valor quebra de carro (não a soma de valores) e o veículo pode trafegar até o final deste novo trecho até sair da malha ou encontrar um novo cruzamento.



Fonte: Freire (2004, p. 29).

Figura 16 – Cruzamento com quebra e valor de quebra respectivamente

Na especificação da visualização gráfica dos veículos é desenvolvido o método `DesenhaCarro()` da superclasse `Carro`. Neste método são calculadas as coordenadas dos pontos iniciais e finais do veículo que será desenhado.

A classe `Visualizador` especifica o desenvolvimento da visualização gráfica do simulador de tráfego de automóveis em uma malha rodoviária urbana. A área de visualização, *viewport*, pode ser representada através de modelo matemático com pontos em um plano cartesiano. No plano cartesiano bidimensional as coordenadas são representadas pela tupla (*x*,*y*). No simulador desenvolvido por Freire (2004) é criado uma estrutura do tipo `TpReal` que contém dois tipos reais: *x* e *y*. Para o desenvolvimento do simulador em três dimensões deve-se atribuir valores de profundidade aos objetos desenhados. Esses valores de profundidade são atribuídos à coordenada *z* do plano cartesiano. Para isso é necessário que seja definida uma nova estrutura, `TPt3D`, semelhante a que Freire (2004) desenvolveu, com um elemento a mais, a coordenada *z*, formando a tripla (*x*,*y*,*z*).

Para a representação gráfica do veículo é criada uma estrutura do tipo `TPrOrdenado`, formado pelas variáveis `Inicio` e `Fim` da estrutura `TPt3D` que contém as informações sobre os pontos iniciais e finais do veículo e as variáveis `corA`, `corB` e `corC` do tipo `real` que armazenam os valores para a definição da cor do veículo. Neste simulador a definição de cor é obtido com a combinação das cores vermelho, verde e azul (*Red*, *Green* e *Blue* - RGB).

Os objetos e cenas criados usando OpenGl consistem em um conjunto de primitivas gráficas simples que são combinadas para formar estruturas mais complexas. Para desenhar o veículo e ou os trechos foram usados vários desenhos de polígonos e retas, combinados de tal forma que representam um veículo ou um trecho. Esses elementos básicos que formam a imagem podem ser linhas, pontos ou círculos. Para determinar qual objeto será desenhado deve-se passar o tipo do objeto como parâmetro, por exemplo `GL_LINES` que desenha uma linha, para a função `glBegin()` e dentro do escopo, até o `glEnd`, deve-se definir os vértices do objeto especificado. Os vértices são as coordenadas que formam o objeto, sendo definido passando os valores para a função `glVertex`. Para sistemas bidimensionais é utilizado o `glVertex2`, onde 2 no final do nome da função especifica a quantidade de argumentos passados para a função. Em sistemas com representação tridimensional é utilizado `glVertex3`, passando como argumentos os valores de *x*, *y* e *z*.

Ainda para a implementação em três dimensões, é necessário ativar uma função nativa do OpenGl que faz o gerenciamento e conseqüentemente a exibição das imagens projetadas na *viewport*. Este gerenciamento, como já foi explicado na seção 2.4.1, é ativado pela função

`glEnable(),` passando como parâmetro a constante `GL_DEPT_TEST`. Esta constante é uma variável de estado utilizada para ativar ou desativar o *buffer* de profundidade, essencial para a visualização de cenas em terceira dimensão.

Na configuração da câmera virtual são calculados os valores mínimos de *x* e *y* e os valores máximos de *x* e *y* dos trechos que compõem a malha rodoviária, desta forma a malha sempre é exibida no centro da tela. Em Freire (2004) esta rotina não está presente.

Para este simulador o tipo de projeção utilizada é a projeção paralela ortográfica, onde independentemente ao valor de *z*, os trechos e os veículos sempre terão o mesmo tamanho, diferente da projeção perspectiva, que para valores diferentes do *z* os objetos possuirão tamanhos diferentes, para longe ou para perto.

3.4 IMPLEMENTAÇÃO

A seguir são mostradas as ferramentas utilizadas na implementação, o código fonte das rotinas principais e a operacionalidade da implementação.

3.4.1 Ferramentas utilizadas

O simulador foi desenvolvido usando o ambiente de programação Delphi 7 e para as rotinas de desenho e de toda a parte gráfica foi utilizado a biblioteca gráfica Opengl.

3.4.2 Rotinas do simulador (implementação)

Nas subseções a seguir são apresentadas as rotinas de definição e criação dos veículos, de teste de continuação de trecho, `TrechoPretendido()` (onde o usuário escolhe o trecho de continuação e reinicia o movimento do veículo guiado), `DesenhaSelecao()` (que desenha um retângulo em volta do trecho escolhido pelo usuário), `SorteiaTrecho()` (reformulada a partir da descrita em Freire (2004)), que calcula os ponto inicial e final do veículo e controla a velocidade de tráfego dos veículos, que desenha os veículos, que especifica o tipo de projeção

utilizado no simulador, que configura a câmera virtual, que especifica os objetos 3D, que ajusta o tamanho do veículo em cruzamentos, dos cálculos matemáticos utilizados e definição de processos concorrentes.

3.4.2.1 Rotina de definição e criação dos veículos

A quantidade de veículos que circulam na malha é definida pelo usuário no campo quantidade de veículos. Durante a criação dos veículos é determinado que o último veículo a entrar na malha possuirá características únicas, como por exemplo a cor da carroceria em preto para diferenciar dos demais veículos. No Quadro 3 pode-se observar a atribuição aleatória de cor aos veículos da classe `CarroRobo` através do método `Create()`.

```
constructor TCarroRobo.create(MyPid:Integer);
begin
  inherited create(mypid);
  Pid:= MyPid;
    Randomize;   // embaralha sorteio
    PrOrdenado[Pid].corA:=((Random(6)+ Random(6))/10);// sorteia cor Red
    Randomize;   // embaralha sorteio
    PrOrdenado[Pid].corB:=((Random(8)+Random(4))/10);//Sorteia cor Green
    Randomize;   // embaralha sorteio
    PrOrdenado[Pid].corC:=((Random(10)+Random(2))/10);//Sorteia Cor Blue

end;
```

Quadro 3 – Sorteio de cores para os veículos

No Quadro 4 tem-se a criação do veículo guiado.

```
constructor TCarroGuiado.create(MyPid:Integer);
begin
  inherited create(mypid);
  Pid:= MyPid;
  PrOrdenado[pid].corA:= 0;//define o valor zero para todas as
  PrOrdenado[pid].corB:= 0;//cores, formando a cor preta
  PrOrdenado[pid].corC:= 0;

end;
```

Quadro 4 – Criação do veículo guiado

3.4.2.2 Rotina de teste de continuação de trecho

No simulador desenvolvido por Freire (2004), o veículo ao chegar ao final de um trecho, o qual possuía mais de uma opção de continuação, o método `SorteiaTrecho` da superclasse `Carro` era ativado para escolher o trecho de continuação. Para o carro controlado

da classe `CarroGuiado` o método `TrechoPretendido` é implementado. O Quadro 5 exibe o teste usado para identificar se a continuação do trecho será aleatória (para o carro robô) ou se será estabelecida pelo usuário (carro guiado).

```
Procedure TCarro.PegaTdTrCruz(Var TrFree: TPonteiro; Var PtFree: Real);
...

if FrmLambda.CBveiculoguiado.Checked then
  //se estiver ativada a funcao de carro guiado entao escolhe o trecho

  begin
    if pid = StrToInt(FrmLambda.EdTotal.Text)-1 then
    begin
    //pega trecho escolhido pelo usuario
    TCarroGuiado(Pid).TrechoPretendido(Trecho[CarroPts[High(carroPts)].
        Indicetr].Tr, TrAtu);
    end
    else
    begin
    //sorteia um trecho qualquer
    SorteiaTrecho(Trecho[CarroPts[High(CarroPts)].Indicetr].Tr, TrAtu);
    end;
  end
  else
  begin
  //senao estiver ativada a funcao carro guiado o trecho é sorteado
  //sorteia um trecho qualquer
    SorteiaTrecho(Trecho[CarroPts[High(CarroPts)].Indicetr].Tr, TrAtu);
  end;
```

Quadro 5 – Teste de continuação de trecho

3.4.2.3 Rotina `TrechoPretendido()`

O método `TrechoPretendido` (Quadro 6) da classe `CarroGuiado` implementa um laço que fica aguardando que o usuário pressione a tecla *up* (para reiniciar o movimento do veículo) e com as teclas *left* e *rigth* o usuário seleciona as direções, que ficam em destaque na malha.

```
Procedure TCarroGuiado.TrechoPretendido(TrUlt: TPonteiro; Var Tr: TPonteiro);
...
if TrUlt^.I_QtdFCont = 1 then //quando houver somente um trecho de
  Tr:= TrUlt^.FCont[qtd]      //continuacao nao precisa escolher.
else begin
  qtd:=TrUlt^.I_QtdFCont; //qtd recebe a quantidade de trecho de conti.
  Tr:=TrUlt^.FCont[0];//Tr recebe o primeiro trecho, caso o usuario
  while not SetaUp do//pressione a tecla up sem escolher o trecho.
  begin//loop para aguardar o pressionar da tecla up.

    if SetaLeft then begin//se for pressionada a tecla left, então
     while continua do//selecionada o trecho.
      begin
      Dec(qtd);//deqrementa o numero de trecho
      if qtd < 0 then qtd:=TrUlt^.I_QtdFCont-1;//se for menor q zero pega
      TrAux2:=TrUlt^.FCont[qtd];//ultimo trecho da lista.
      if not ((TrUlt^.Fim.X = TrAux2^.Fim.X) and//faz teste se o trecho
            (TrUlt^.Fim.Y = TrAux2^.Fim.Y))then//é continuação.
       continua:=false;
     end;//fim do while para teste de trecho
    continua:=true;
    Tr:=TrAux2;//Tr recebe o trecho escolhido
    selecao:=true;//habilita o desenho da selecao
    Inicio:=TrAux2[I].Inicio;Fim:=TrAux2[I].Fim;//passa o valor
    Largura:= TrAux2[I].F_Largura;//necessário para desenhar o
    DesenhaSelecao(Inicio, Fim, Largura, 2);//retangulo
    SetaLeft:=false;//desabilita a seta left
    end;

    if SetaRight then begin//testa a tecla rigth
     while continua do
     begin
      Inc(qtd);//incrementa o numero de trechos
      if qtd > TrUlt^.I_QtdFCont-1 then// se for maior que o numero
       qtd:= 0;//de trechos reinicia a contagem
      TrAux2:=TrUlt^.FCont[qtd];
      if not ((TrUlt^.Fim.X = TrAux2^.Fim.X) and//teste para saber se o
            (TrUlt^.Fim.Y = TrAux2^.Fim.Y)) then// trecho é continuacao
       continua:=false;
     end;//fim while para encontrar um trecho compativel
    continua:=true;
    Tr:=TrAux2;//Tr recebe o trecho escolhido
    selecao:=true;
    Inicio:=TrAux2[I].Inicio;//passa os valores para desenhar a selecao
    Fim:= TrAux2[I].Fim; Largura:= TrAux2[I].F_Largura;
    DesenhaSelecao(Inicio, Fim, Largura, 2);
    SetaRight:=false;//desabilita o teste
    end;
  end;
end;//fim do while da seta up
SetaUp:=false;//quando for pressionada a sete up seta seu valor para
selecao:=false;//desabilita o desenho da selecao de trecho
```

Quadro 6 – Laço de seleção de trechos

3.4.2.4 Rotina `DesenhaSelecao()`

O método `DesenhaSelecao()` (Quadro 7) da classe `CarroGuiado` passa como parâmetro os valores de inicio, fim e largura do trecho selecionado para a rotina `DesenhaPontoSel()`da classe `Visualizador`, que calcula os pontos necessários para desenhar um retângulo sobre o trecho selecionado, para que o usuário possa visualizar o trecho escolhido.

No Quadro 8 é mostrado a rotina `DesenhaPontoSel()`.

```
Procedure TCarroGuiado.DesenhaSelecao(Var Inicio,Fim: TPt3D; Var Largura:
Real);
begin
    FrmGrafico.DesenhaPontoSel(Inicio, Fim, Largura);
end;
```

Quadro 7 – Rotina `DesenhaSelecao()`

```
Procedure TFrmGrafico.DesenhaPontoSel(Inicio,Fim: TPt3D; Largura: Real);
...
CalcPtSelecao(Inicio,Fim,Largura,PtA,PtB,PtC,PtD); //busca os pontos extremos

  glLineWidth(2);                 // define a espessura da linha
  glColor3f(0, 0,255);             // definir a cor azul p/ a linha

  glBegin(GL_LINES);              // desenha linha simples
    glVertex3f(PtA.X, PtA.Y, 10);
    glVertex3f(PtB.X, PtB.Y, 10);//desenha os segmentos de reta para
    glVertex3f(PtB.X, PtB.Y, 10);// destacar o trecho selecionado
    glVertex3f(PtC.X, PtC.Y, 10);
    glVertex3f(PtC.X, PtC.Y, 10);
    glVertex3f(PtD.X, PtD.Y, 10);
    glVertex3f(PtD.X, PtD.Y, 10);
    glVertex3f(PtA.X, PtA.Y, 10);
  glEnd;
  glLineWidth(1);//volta a espessura da linha original
```

Quadro 8 – Rotina `DesenhaPontoSel()`

3.4.2.5 Rotina `SorteiaTrecho()`

A rotina `SorteiaTrecho()` desenvolvida por Freire (2004), utilizada para atribuir uma opção de caminho para o veículo, foi reformulada para adequar-se aos critérios de seleção de trecho. Esses critérios são utilizados para restringir o acesso dos veículos em cruzamentos com rotatória, para evitar que o veículo fique andando em círculos. O Quadro 9 mostra a rotina desenvolvida por Freire (2004).

Devido à dificuldade em avaliar os critérios de escolha de um novo trecho através da rotina desenvolvida por Freire (2004) é proposto uma reformulação para rotina `SorteiaTrecho()` presente na superclasse `Carro` (Quadro 10).

```
Procedure TCarro.SorteiaTrecho(TrUlt: TPonteiro; Var Tr: TPonteiro);
...
while not achou do
  begin
  dead:= true;
  for I:= 0 to High(Qtd) do
  begin
   if Qtd[I] <> -1 then
   dead:= false;
   end;
  if dead then
   begin
   achou:= true;
   FrmLambda.LbMensagem.Caption:='ErroDeadlock!   Trecho'+Inttostr(TrUlt^.I_CdTrecho)+Quebra   Carro'+   inttostr(queCarro)+'
(VERIFICAR A CONFIGURAÇÃO DA MALHA!)  ';
   sleep(infinite);
   end   else
   begin
   //sorteia um do trechos de continuacao
   Randomize;
   // pega o indice do novo trecho
   Ent:= Random(TrUlt^.I_QtdFCont);
   qtd[Ent]:= -1;//define quantas vezes entrou para procura um novo trechobe novo trecho sorteado
   Tr:= TrUlt^.Fcont[Ent];
   if (Tr^.Inicio.X = TrUlt^.Fim.X) and (Tr^.Inicio.Y = TrUlt^.Fim.Y) then
       begin
       if QueCarro < 0 then
       begin
       //se a quebra for maior q o valor da quebra posso pegar qqr trecho
       if QueCarro >= Tr^.I_VlQuebra then
       begin
       if Tr^.I_Quebra = 0 then
       QueCarro:= 0
       else
       QueCarro:= QueCarro + Tr^.I_Quebra;
       //farca a saida da busca
       achou:= true;
       end;
       end       else
       begin
       //se a quebra for maior q o valor da quebra posso pegar qqr trecho
       if QueCarro <= Tr^.I_VlQuebra then
         begin
         if Tr^.I_Quebra = 0 then
           QueCarro:= 0
          else
            QueCarro:= QueCarro + Tr^.I_Quebra;
            //farca a saida da busca
            achou:= true;
          end;
         end;
      end      else
      begin
      //se nao for, entao acha o primeiro trecho onde o final é igual ao inicio
      Ent:= 0;
      while Ent < TrUlt^.I_QtdFCont do
    begin
    qtd[Ent]:= -1;//definequantas vezes entrou para procura um novo trecho recebe novo trecho sorteado
    Tr:= TrUlt^.Fcont[Ent];
    if (Tr^.Inicio.X = TrUlt^.Fim.X) and  (Tr^.Inicio.Y = TrUlt^.Fim.Y) then
       begin
       if QueCarro < 0 then
         begin
        //se a quebra for maior q o valor da quebra posso pegar qqr trecho
         if QueCarro >= Tr^.I_VlQuebra then
         begin
          if Tr^.I_Quebra = 0 then
          QueCarro:= 0
          else
          QueCarro:= QueCarro + Tr^.I_Quebra;
          //forca a saida do loop
         Ent:= TrUlt^.I_QtdFCont;
          //farca a saida da busca
          achou:= true;
          end;
          end       else
          begin
        //se a quebra for maior q o valor da quebra posso pegar qqr trecho
      if QueCarro <= Tr^.I_VlQuebra then
      begin
      if Tr^.I_Quebra = 0 then
      QueCarro:= 0
      else
      QueCarro:= QueCarro + Tr^.I_Quebra;
      //forca a saida do loop
      Ent:= TrUlt^.I_QtdFCont;
      //farca a saida da busca
      achou:= true;
     end;
    end;
   end;
  Inc(Ent);
 end;
end;
end;
end;
```

Quadro 9 – Rotina `SorteiaTrecho()` Freire (2004)

```
Procedure TCarro.SorteiaTrecho(TrUlt: TPonteiro; Var Tr: TPonteiro);
...
SetLength(Qtd, TrUlt^.I_QtdFCont);
while not achou do
begin//while achou
 Randomize;
 Ent:= Random (TrUlt^.I_QtdFCont);//sorteio do trecho
 Tr:= TrUlt^.FCont[Ent];//pega o trecho sorteado
 while not achou do
 Begin     //o trecho é continuação?
  if  (Tr^.Fim.X = TrUlt^.Fim.X)and(Tr^.Fim.Y = TrUlt^.Fim.Y) then
  begin
   Inc(Ent);//então pega proximo trecho disponivel
   if Ent >= TrUlt^.I_QtdFCont then//se já nao existir mais
    Ent:= 0;                    //trechos pega o de indice 0
   Tr:= TrUlt^.FCont[Ent];   //que com certeza servi
  end  else
  begin
   if Tr^.I_Quebra <> 0 then //verificação do criterio de
   begin                     //de quebra de carro
    if QueCarro <= Tr^.I_VlQuebra then  //se a quebra for
    Begin                               //menor que quebra
     QueCarro:= QueCarro + Tr^.I_Quebra;//de carro o carro
     achou:= true;                      //pode seguir viajem
    end    else
    begin               //senao for
     while not achou do
     begin
      dead:=true;         //verifica se o carro já testou
      for I:= 0 to High(Qtd) do//todas as opçoes
      begin
       if Qtd[I] <> -1 then
        dead:= false;
      end;
      if dead then    //se ja testou todas as opções exibe
      begin           //mensagem de erro
       achou:= true;
       FrmLambda.LbMensagem.Caption:= 'Erro DeadLock!
       sleep(infinite);     //e fica esperando
      end;
     Qtd[Ent]:=-1;
     Inc(Ent);
     if Ent >= TrUlt^.I_QtdFCont then
      Ent:= 0;
     Tr:= TrUlt^.FCont[Ent];//se já testou todas as rotas
     achou:= true;// e nenhuma passou no test epode seguir
     end;         //pela rota que sai do cruzamento
     achou:=false;
     end;
    end    else
    begin
     QueCarro := Tr^.I_Quebra;//se a rota passou no teste
     achou:= true;    //atribui o valor de quebra no carro
    end;
  end;
 end;
end;
```

Quadro 10 – Rotina `SorteiaTrecho()` reformulada

3.4.2.6 Rotina para calcular o ponto inicial e final do veículo e controle de velocidade de tráfego

O método `DesenhaCarro()` da superclasse `Carro` prepara as informações que serão usadas na visualização gráfica dos veículos (Quadro 11). Os valores calculados são inseridos no *array* `PrOrdenado` do tipo `TPrOrdenado` na posição indicada pelo valor `Pid` de cada

veículo. Para estipular a velocidade é usado um valor de pausa na rotina, onde o processo não executa nada em um determinado tempo, através da função do Delphi 7 `Sleep`, utilizando como parâmetro o valor fornecido para a velocidade dos veículos.

```
Procedure TCarro.DesenhaCarro;
...
//calcula ponto inicial e final do carro
CalculaPtTrecho(Trecho[CarroPts[0].Indicetr].Tr,
                       CarroPts[0].PtCarro,PtICar);
CalculaPtTrecho(Trecho[CarroPts[High(CarroPts)-1].Indicetr].Tr,
                        CarroPts[High(CarroPts)-1].PtCarro,PtFCar);

 //pega um tempo para desenhar o carro
 Sleep(3600 div strtoint(FrmLambda.EdVelocidade.Text));

 if PtFCar.Z < PtICar.Z then
 begin
  PtFCar.Z:=PtICar.Z;
 end
 else
 begin
  PtICar.Z:=PtFCar.Z;
 end;
 P(SemaTela);                          //pega p para tela
 PrOrdenado[Pid].Inicio:= PtICar;   // desenha carro na lista
 PrOrdenado[Pid].Fim   := PtFCar;   // de posicoes de carros
 V(SemaTela);                          //libera V para tela
```

Quadro 11 – Cálculo do ponto inicial e final do veículo e velocidade de tráfego

3.4.2.7 Rotina para desenhar os veículos

Através das informações armazenadas no *array* `PrOrdenado` a rotina `DesenhaCarro` da classe `Visualizador` desenha a estrutura do veículo, como lanternas, faróis, limpador de pára-brisa e carcaça do veículo (Quadro 12).

3.4.2.8 Especificação do tipo de projeção do OpenGl

Para permitir a visualização da malha rodoviária e a simulação dos veículos é necessário especificar para a OpenGl o tipo de projeção e, para ambientes em terceira dimensão habilitar o teste de profundidade (Quadro 13).

```
Procedure TFrmGrafico.DesenhaCarro(Pares: TPrOrdenado);
...
if (Pares.Inicio.X <> Pares.Fim.X) or     //teste para se valor de inicio
     (Pares.Inicio.Y <> Pares.Fim.Y) then //e fim sao validos
  begin
    //calcula os pontos paralelos a reta
    CalcPt(Pares.Inicio,Pares.Fim, 2.5/2, A, B, C, D);
    //**************lanterna traseira esquerda************************
    PtMedio.X:= (Pares.Inicio.X + A.X) / 2;
    PtMedio.Y:= (Pares.Inicio.Y + A.Y) / 2;
    PtMedio.Z:= (Pares.Inicio.Z + a.Z) / 2;
    //calcula os pontos paralelos a reta
    CalcPt(PtMedio, A, 0.2, auxA, auxB, auxC, auxD);
    //cor lanterna traseira vermelha
    glColor3f(1.0, 0.0, 0.0);
    //desenha Lanternas traseiras
    DesenhaPoligno(A, PtMedio, AuxB, AuxC);
    //****************lanterna traseira direita************************
    PtMedio.X:= (Pares.Inicio.X + B.X) / 2;
    PtMedio.Y:= (Pares.Inicio.Y + B.Y) / 2;
    PtMedio.Z:= (Pares.Inicio.Z + B.Z) / 2;
    //calcula os pontos paralelos a reta
    CalcPt(PtMedio, B, 0.2, auxA, auxB, auxC, auxD);
    //cor lanterna traseira vermelha
    glColor3f(1.0, 0.0, 0.0);
    //desenha Lanternas traseiras
    DesenhaPoligno(B, PtMedio, AuxA, AuxD);
    //***************farol dianteiro esquerdo************************
    PtMedio.X:= (Pares.Fim.X + D.X) / 2;
    PtMedio.Y:= (Pares.Fim.Y + D.Y) / 2;
    PtMedio.Z:= (Pares.Fim.Z + D.Z) / 2;
    //calcula os pontos paralelos a reta
    CalcPt(PtMedio, D, 0.2, auxA, auxB, auxC, auxD);
    //cor farol dianteiro branco
    glColor3f(1.0, 1.0, 1.0);
    //desenha Lanternas traseiras
    DesenhaPoligno(D, PtMedio, AuxA, AuxD);
    //******************farol dianteiro direito************************
    PtMedio.X:= (Pares.Fim.X + C.X) / 2;
    PtMedio.Y:= (Pares.Fim.Y + C.Y) / 2;
    PtMedio.Z:= (Pares.Fim.Z + C.Z) / 2;
    //calcula os pontos paralelos a reta
    CalcPt(PtMedio, C, 0.2, auxA, auxB, auxC, auxD);
    //cor farol dianteiro branco
    glColor3f(1.0, 1.0, 1.0);
    //desenha Lanternas traseiras
    DesenhaPoligno(C, PtMedio, AuxB, AuxC);
    //****************desenha parabrisa dianteiro*********************
    //calcula os pontos paralelos a reta
    CalcPt(D, C, 1, auxA, auxB, auxC, auxD);
    Aux1:= AuxB;
    Aux2:= AuxC;
    //calcula os pontos paralelos a reta
    CalcPt(Pares.Fim, C, 1.4, auxA, auxB, auxC, auxD);
    //calcula os pontos paralelos a reta
    CalcPt(Pares.Fim, AuxB, 1, auxA, auxB, auxC, auxD);
    //cor parabrisa branco
    glColor3f(1.0, 1.0, 1.0);
    //desenha parabrisa
    DesenhaPoligno(Aux1, Aux2, AuxD, AuxC);
    //cor bordas do parabrisa
    glColor3f(0.0, 0.0, 0.0);
    //calcula os pontos paralelos a reta
    CalcPt(Pares.Fim, C, 1, auxA, auxB, auxC, auxD);
    //calcula os pontos paralelos a reta
    CalcPt(Pares.Fim, AuxB, 0.5, auxA, auxB, auxC, auxD);
    Aux1:= AuxC;
    Aux2:= AuxD;
    //calcula os pontos paralelos a reta
    CalcPt(Pares.Fim, C, 1.3, auxA, auxB, auxC, auxD);
    //calcula os pontos paralelos a reta
    CalcPt(Pares.Fim, AuxB, 0.7, auxA, auxB, auxC, auxD);
    //cor bordas do parabrisa
    glColor3f(0.0, 0.0, 0.0);
    //****************desenha limpador parabrisa*********************
    DesenhaLinha(Aux1, AuxC);
    //calcula os pontos paralelos a reta
    CalcPt(Pares.Fim, C, 1.3, auxA, auxB, auxC, auxD);
    //calcula os pontos paralelos a reta
    CalcPt(Pares.Fim, AuxB, 0.3, auxA, auxB, auxC, auxD);
    //cor bordas do parabrisa
    glColor3f(0.0, 0.0, 0.0);
    //desenha limpador parabrisa
    DesenhaLinha(Aux2, AuxD);
    //****************desenha parabrisa traseiro*********************
    //calcula os pontos paralelos a reta
    CalcPt(A, B, 0.8, auxA, auxB, auxC, auxD);
    Aux1:= AuxA;
```

```
    Aux2:= AuxD;
    //calcula os pontos paralelos a reta
    CalcPt(Pares.Inicio, B, 1.1, auxA, auxB, auxC, auxD);
    //calcula os pontos paralelos a reta
    CalcPt(Pares.Inicio, AuxA, 1, auxA, auxB, auxC, auxD);
    //cor parabrisa branco
    glColor3f(1.0, 1.0, 1.0);
    //desenha parabrisa
    DesenhaPoligno(Aux1, Aux2, AuxC, AuxD);
    //****************cor bordas do carro preto************************
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_LINE_LOOP);
      glVertex3f(A.X, A.Y, A.Z);
      glVertex3f(B.X, B.Y, B.Z);
      glVertex3f(C.X, C.Y, C.Z);
      glVertex3f(D.X, D.Y, D.Z);
    glEnd;
   //**************desenha carcaça do carro*****************************
   //define cor do carro
    glColor3f(pares.corA, pares.corB, pares.corC);
    DesenhaPoligno(A, B, C, D);
  end;
```

Quadro 12 – Desenha o veículo

3.4.2.9 Configuração da câmera virtual

A função `gluLookAt()` é utilizada para definir a posição da câmera virtual. Os parâmetros são obtidos da configuração da malha rodoviária (Quadro 13).

```
procedure TFrmGrafico.Redesenha;
...
//evita a divisão por zero
 if Height = 0 then
Height:=1;
//define tamanho da viewport
glViewport(0,0,ClientWidth, ClientHeight);
Aspecto:= Width/Height;
//diz para o Opengl que as transformações serão
//no modelo da projeção
glMatrixMode(GL_PROJECTION);
glLoadIdentity;
//defini o tipo de projeção e a posição da câmera virtual
glOrtho(Esquerda,Direita,Abaixo,Acima,-400.0,400.0);
//posiciona a camera virtual em cima da cena
gluLookAt((FrmLambda.XDist)/2+FrmLambda.XMenor,(FrmLambda.YDist)/2+
FrmLambda.YMenor,FrmLambda.DistMaior/2,(FrmLambda.XDist)/2+FrmLambda.
XMenor,(FrmLambda.YDist)/2+FrmLambda.YMenor,0.0,0,1.0,0);
//habilita o teste de profundidade
glEnable(GL_DEPTH_TEST);
//limpa os buffers de cor e de profundidade para
//começar a desenhar novamente
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
```

Quadro 13 – Tipo de projeção e teste de profundidade

3.4.2.10 Especificação dos objetos em terceira dimensão

No Quadro 14 é mostrado o uso da função `glVertex3f()`, para desenhar objetos em três dimensões. Nesta rotina é desenhado um polígono com quatro vértices, número correspondente a chamada da função `glVertex3f()` dentro do escopo do `glBegin`. O argumento passado para a função `glBegin`, `GL_POLYGON`, define um polígono sólido, utilizado para desenhar a parte preenchida em cinza do trecho e também para desenhar a carcaça do veículo.

Para especificar somente o contorno de um polígono (não preenchido) é utilizado o `glBegin` com o argumento `GL_LINE_LOOP`. Essa primitiva gráfica é utilizada para definir o contorno dos trechos e veículos como também desenhar o retângulo em azul nos trechos selecionados pelo usuário (Quadro 15).

```
 glBegin(GL_POLYGON);
     glVertex3f(A.X, A.Y, A.Z);
     glVertex3f(B.X, B.Y, B.Z);
     glVertex3f(C.X, C.Y, C.Z);
     glVertex3f(D.X, D.Y, D.Z);
   glEnd;
```

Quadro 14 – Desenhando um polígono sólido com Opengl

```
    glColor3f(0.0, 0.0, 0.0);
    //desenha bordas
    glBegin(GL_LINE_LOOP);
      glVertex3f(A.X, A.Y, A.Z);
      glVertex3f(B.X, B.Y, B.Z);
      glVertex3f(C.X, C.Y, C.Z);
      glVertex3f(D.X, D.Y, D.Z);
    glEnd;
```

Quadro 15 – Desenhando o contorno de um polígono com OpenGl

3.4.2.11 Ajuste do tamanho do veículo em cruzamento

Para ajustar o tamanho do veículo quando o mesmo está passando por um cruzamento é preciso recalcular os valores para o ponto inicial e final do veículo. No Quadro 16 é descrito e comentado parte do código `DesenhaCarro()` da superclasse `Carro`.

```
Procedure TCarro.DesenhaCarro;
...
//Calcula par ordenado inicial do carro
  CalculaPtTrecho(Trecho[CarroPts[0].Indicetr].Tr,
                  CarroPts[0].PtCarro,PtICar);
  //Calcula par ordenado final do carro
  CalculaPtTrecho(Trecho[CarroPts[High(CarroPts)-1].Indicetr].Tr,
                  CarroPts[High(CarroPts)-1].PtCarro,PtFCar);
  //verifica se o tamanho atual do carro é menor que seu tamanho real
  if CalcComprimento(PtICar,PtFCar) < CarroPts[pid].tamanhoCar-1 then
  begin
    //verifica x inicial é menor que x final
    if (PtICar.X < PtFCar.X) then
    begin
      //diminui o valor de x
      //calcula o valor para diminuir em x
      compri:=(CarroPts[pid].tamanhoCar-1) - (CalcComprimento(PtICar,PtFCar));
      PtICar.X:= PtIcar.X-compri;
    end else
    begin
      //senao aumenta o valor em x
      //calcula o valor para aumentar
      compri:=(CarroPts[pid].tamanhoCar-1) - (CalcComprimento(PtICar,PtFCar));
      PtICar.X:= PtIcar.X+compri;
    end;
    end;
```

Quadro 16 – Ajuste do tamanho do veículo em cruzamento

3.4.2.12 Rotinas matemáticas

O Quadro 17 exibe a rotina utilizada para calcular o comprimento de uma rua utilizando a fórmula da geometria analítica para obter a distância entre dois pontos.

```
Function CalcComprimento(Inicio, Fim: Tpt3D): Real;
var
  QuadradoX, QuadradoY, QuadradoZ: Real;
begin
  //formula Dab = Raiz((xi - xf)^2 + (Yi - Yf)^2)
  QuadradoX:= (Inicio.X - Fim.X) * (Inicio.X - Fim.X);
  QuadradoY:= (Inicio.Y - Fim.Y) * (Inicio.Y - Fim.Y);
  QuadradoZ:= (Inicio.Z - Fim.Z) * (Inicio.Z - Fim.Z);
  result:=strtofloat(format('%5.0f',[sqrt(QuadradoX + QuadradoY)]));
end;
```

Quadro 17 – Rotina para calcular a distância entre dois pontos

No Quadro 18 é mostrada a rotina para calcular o ponto médio de um segmento utilizado para desenhar a linha central da rua.

```
Function CalcPtmedio(Inicio, Fim: Tpt3D): Tpt3D;
var
  Pt: Tpt3D;
begin
  Pt.X:= (Inicio.X + fim.X)/ 2;
  Pt.Y:= (Inicio.Y + fim.Y)/ 2;
  PT.Z:= (Inicio.Z + fim.Z)/ 2;
  Result:= Pt;
end;
```

Quadro 18 – Rotina para calcular o ponto médio

O Quadro 19 exibe o cálculo realizado para os pontos iniciais e finais dos objetos

desenhados utilizando a trigonometria para obter a inclinação em relação ao eixo x.

```
      Procedure CalcPt(Inicio, Fim: Tpt3D; Largura: Real; Var PtA, PtB,
PtC, PtD: TPt3D);
      var
        CAng, Grau: Real;
        PtDelta: TPt3D;
      begin
      …
      //Calcula o coeficiente angula
          CAng:= (Fim.Y - Inicio.Y)/(Fim.X- Inicio.X);

          //transforma o coeficiente angular para radianos
          //para saber inclinacao da reta
          Grau:= ArcTan(CAng);


          //acha a variacao em x e y pelo sen e cos
          PtDelta.X:= Largura * sin(Grau);
          PtDelta.Y:= Largura * cos(Grau);
      ...
      end;
```

Quadro 19 – Rotina para calcular os pontos iniciais e finais dos objetos

3.4.2.13 Definição de processos concorrentes

Para os processos concorrentes a classe Carro é definida como derivada da classe TThread, própria do ambiente Delphi 7 (Quadro 20).

```
type
  TCarro = class(TThread)    // define uma classe derivada de TThread
  Public
    Pid: integer;         //Define o identificador do carro
    Trecho:  Array  of  TTrechos;//Define  os  trechos  ocupados  e  seus
respectivos tamanhos
    CarroPts: Array of TPontos;//Define os pontos ocupados pelo carro e
                                     // seus respectivos trechos
    QueCarro  : Integer;  //Define o indice da quebre
  //Public
...
end;
```

Quadro 20 – Definição de processos concorrentes

Para a comunicação entre processos concorrentes são utilizados as operações de P e V sobre semáforos (Quadro 21).

```
      Procedure TCarro.P(Sema: Cardinal);
      begin
        WaitForSingleObject(Sema, INFINITE);
      end;
      Procedure TCarro.V(Sema: Cardinal);
      begin
        ReleaseSemaphore(Sema, 1, nil);
      end;
```

Quadro 21 – Operações sobre semáforos

3.4.3 Operacionalidade da implementação

A interface do simulador é mostrada na Figura 17, juntamente com a visualização de uma simulação.

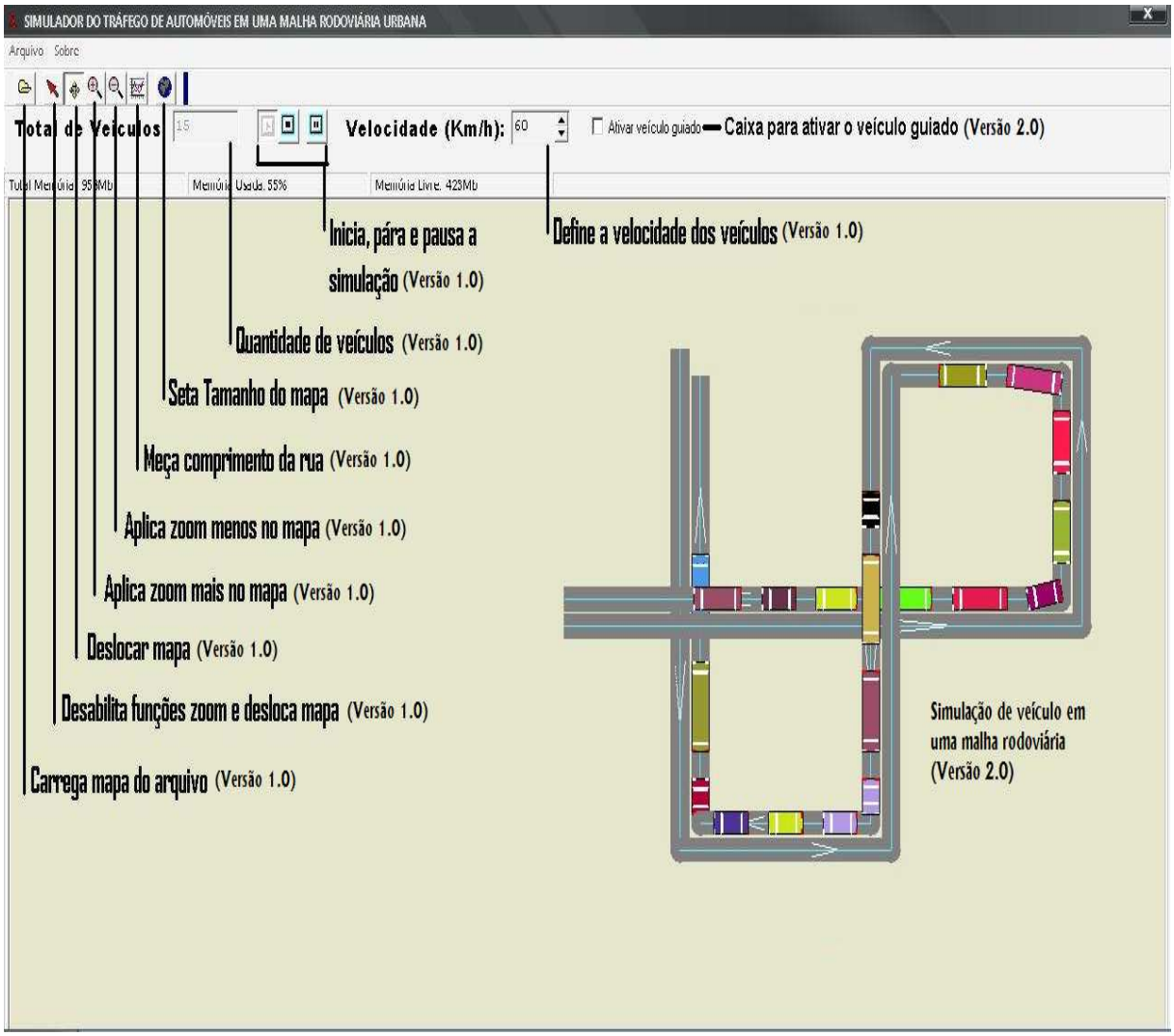


Figura 17 – Tela principal do simulador

No menu `Arquivo` (Figura 17), ou também através do ícone `Abrir Arquivo Mapa` na `ToolBar`, é possível selecionar o caminho onde se encontra o arquivo texto com a configuração da malha rodoviária. Na Figura 18 é apresentada a tela para abrir o arquivo texto.

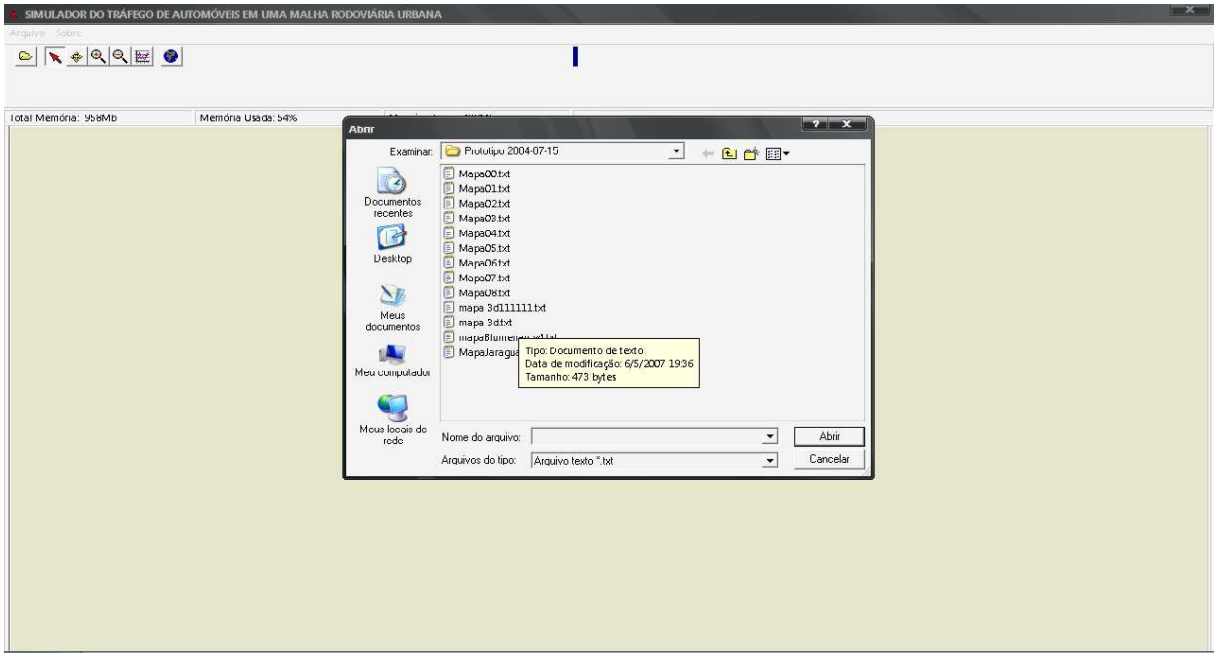


Figura 18 – Carregar o mapa

Depois de informar o nome do arquivo, o simulador desenha na tela a representação gráfica das informações contidas neste arquivo texto, ou seja, desenha a malha rodoviária. Neste momento o usuário pode, através do botão de `zoom` que se encontra na `ToolBar`, efetuar sobre o desenho operações de `zoom-in`, a qual aproxima a tela de desenho e conseqüentemente aumenta o mesmo ou aplicar a operação de `zoom-out` a qual afasta o desenho, deixando-o menor. Pressionando o botão `Seta Tamanho Padrão do Mapa` a visualização da malha rodoviária volta ao tamanho definido como padrão.

O botão `Mover` permite que o usuário, ao fixar o *click* em cima do mapa, mova a malha rodoviária para uma outra posição da tela, acompanhando o movimento do *mouse*.

Para inicializar a simulação é preciso que seja informado o número de veículos que irão circular na malha rodoviária, sendo que a velocidade também pode ser informada. Caso não seja informado um valor padrão, é assumido sessenta (60) (Figura 19).

Ao selecionar `Executar Simulação`, os veículos começam a entrar na malha rodoviária, circulando na mesma. O veículo com a cor da carcaça em preto representa o veículo que o usuário poderá guiar. Desta forma, se a opção de `Ativar veículo guiado` estiver ativada, o veículo guiado ao se aproximar de um cruzamento pára em uma área segura, permitindo que outros veículos trafeguem, e aguarda a orientação do usuário que precisará definir qual direção tomar. A Figura 20 mostra o veículo controlado pelo usuário aguardando uma ação e, outros veículos que estão atrás do mesmo na mesma faixa de rolamento ficam também parados até que o veículo controlado seja movimentado.

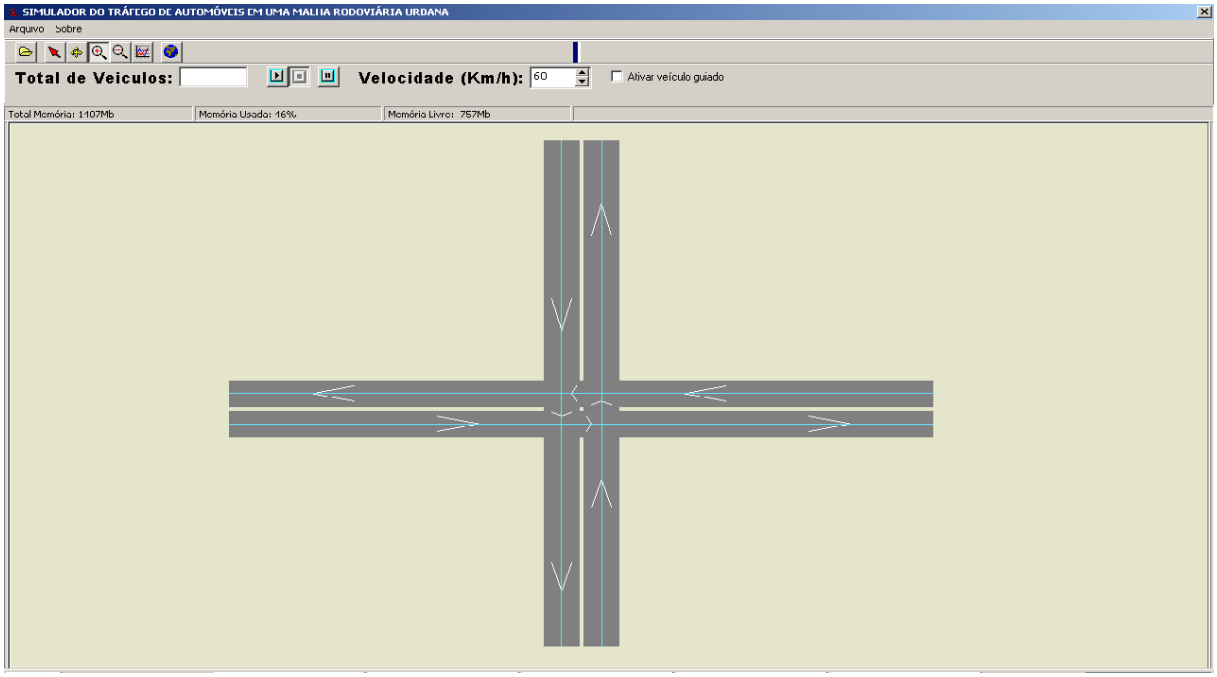


Figura 19 – Início da execução do simulador

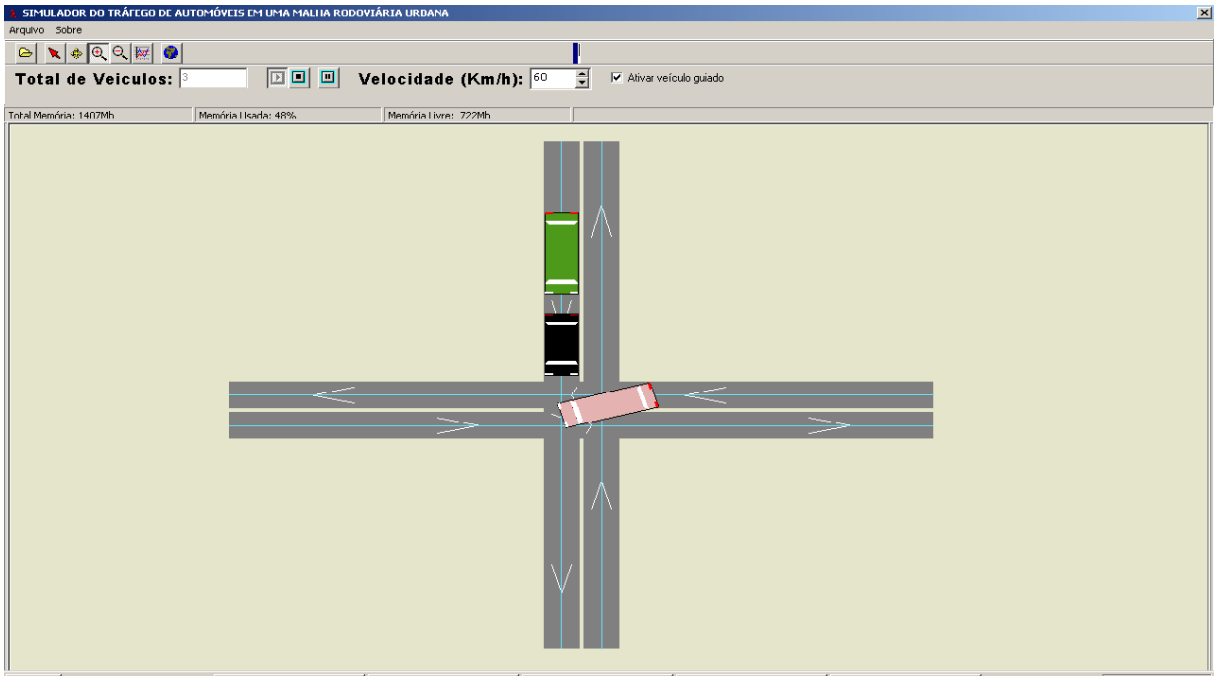


Figura 20 – Veículo aguardando a ação do usuário

Na Figura 21 pode-se observar a escolha, pelo usuário, de um trecho de continuação fora do cruzamento.

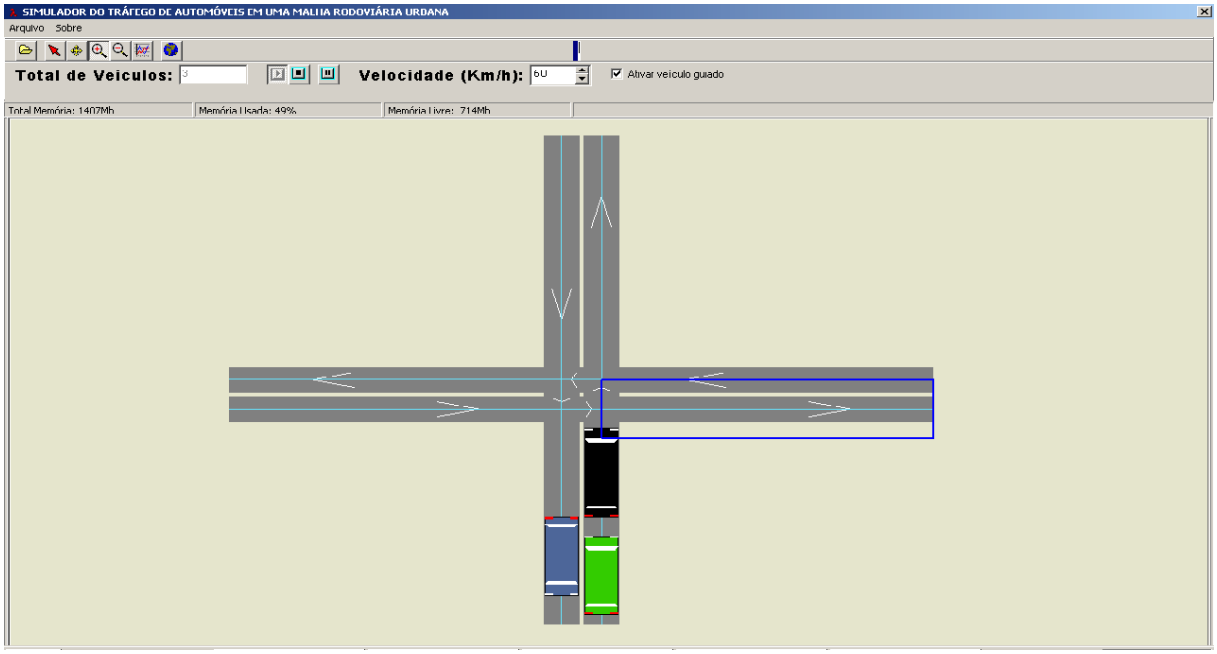


Figura 21 – Selecionando trecho fora de cruzamento

Pode-se observar na Figura 22 o usuário selecionando um trecho dentro do cruzamento, e nesta situação, o veículo somente irá movimentar-se após a escolha de todos os trechos necessários para o veículo sair do cruzamento.

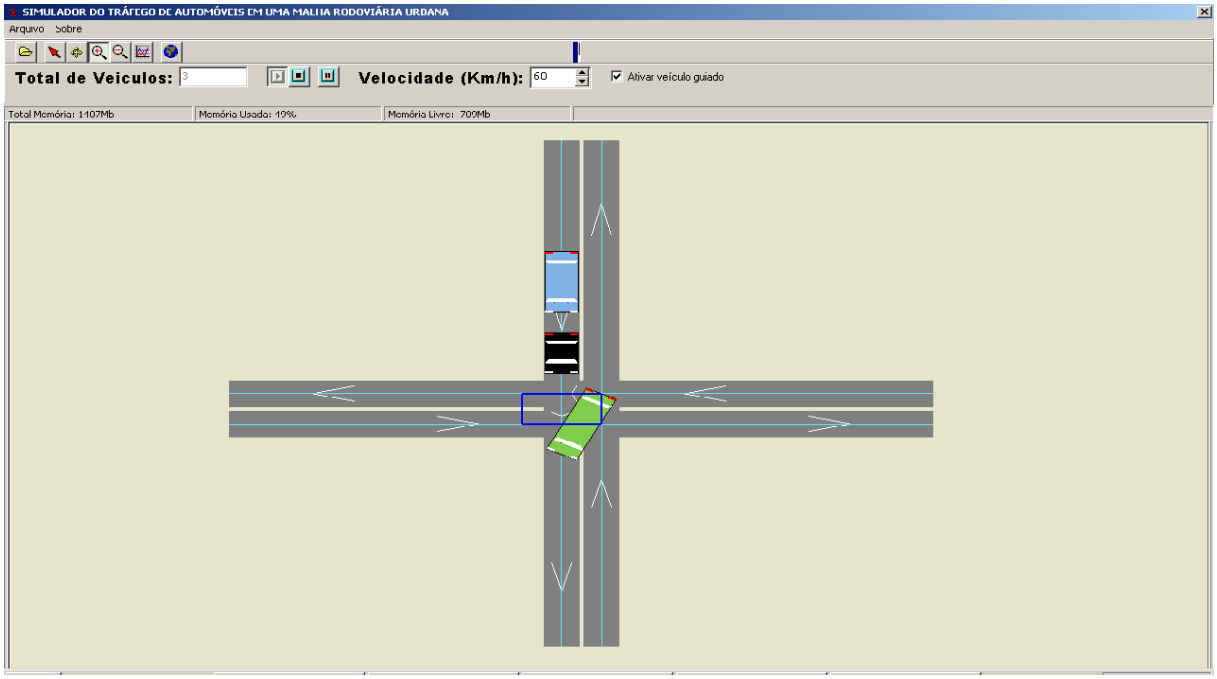


Figura 22 – Seleção de trecho em cruzamento

Na Figura 23 observa-se o usuário selecionando um trecho de continuação no cruzamento. Neste momento as opções de trechos disponíveis serão somente aquelas que promovam a continuação do trecho anteriormente selecionado pelo usuário.

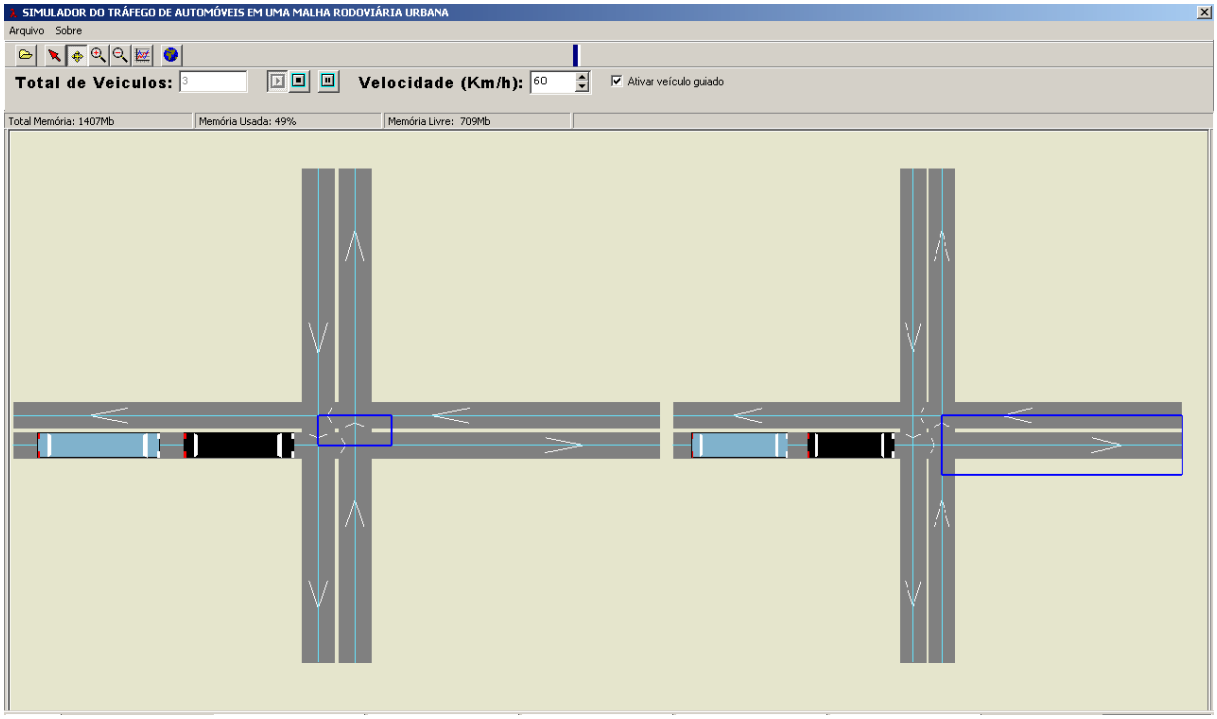


Figura 23 – Seleção de trechos no cruzamento

Ainda na `ToolBar`, a opção `Pausar Simulação` congela o movimentos dos veículos e a opção `Para Simulação` encerra a simulação, permitindo que seja novamente editado o campo com o número de veículos para uma nova execução.

Durante a execução o usuário pode alterar a velocidade que os veículos estão transitando para valores maiores ou menores.

Para ambientes em terceira dimensão, especificados no arquivo texto usado para carregar a malha rodoviária no simulador, a operacionalidade é a mesma para cenas em duas dimensões. Quando o veículo encontra-se em um trecho que está em um nível inferior e há uma passagem sob um viaduto, o veículo é visualizado embaixo do mesmo (Figura 24).

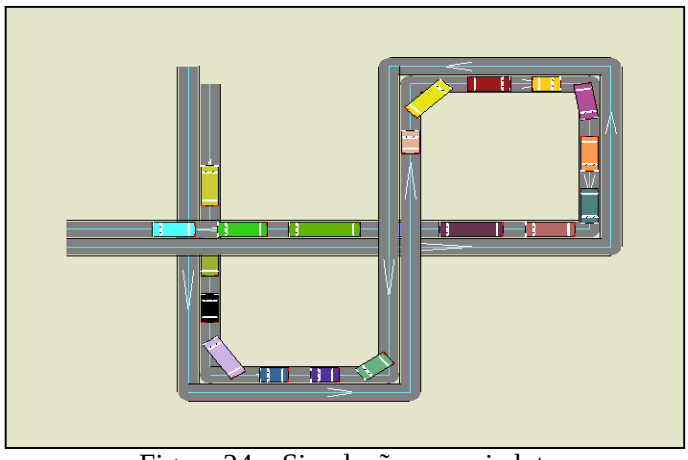


Figura 24 – Simulação com viadutos

3.5 RESULTADOS E DISCUSSÃO

Na especificação da projeção gráfica do simulador é utilizada a projeção paralela ortográfica. Neste sentido os valores atribuídos para a coordenada z são utilizados somente para definir qual objeto será exibido e qual ficará oculto. Desta forma a percepção de tamanho do veículo não é alterada quando ele passa em um nível inferior, onde o valor de *z* é menor ou quando ele passa em um nível superior, onde o valor de *z* é maior.

Na representação dos trechos não é possível visualizar uma subida ou descida de uma rua, o valor da coordenada *z* no ponto inicial e final do trecho são iguais, ou seja, não há inclinação nos trechos. Na junção de trechos com níveis diferentes não há suavização dos valores para a coordenada z, o trecho termina com um valor para a coordenada *z* alto e o próximo trecho já inicia com um valor para a coordenada *z* mais baixo, formando um degrau entre os trechos.

A Figura 25 exibe como ficaria a representação gráfica do simulador com projeção perspectiva.

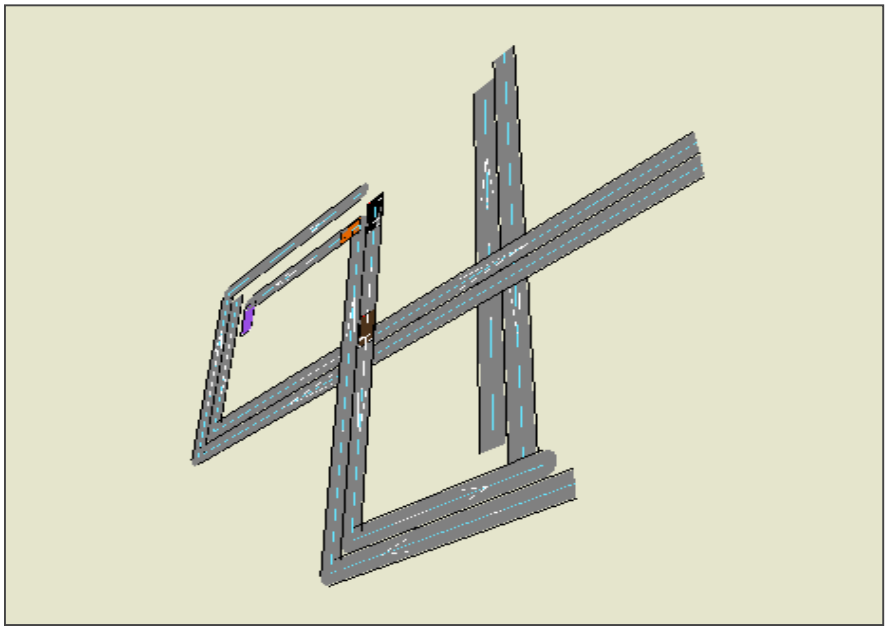


Figura 25 – Simulador com projeção perspectiva

No Quadro 22 é mostrada uma relação entre o trabalho realizado e os trabalhos correlatos.

	Trabalho proposto		Trabalhos correlatos		
	Protótipo de Freire (2004) versão 1	**Protótipo versão 2**	**Sist. multiagentes**	**Simulação de tráfego**	**Simulação e supervisão de tráfego**
Simulação de tráfego	**X**	**X**	**X**	**X**	**X**
Multiagente			**X**		
Realidade imersiva				**X**	
Sistema distribuído					**X**
Utilização de semáforos luminosos					**X**
Opção de carro guiado		**X**			
Visualização em 3D		**X**	**X**	**X**	
Definir velocidade para os veículos	**X**	**X**			
Definir quantidade de veículos	**X**	**X**			
Iniciar simulação	**X**	**X**			
Parar simulação	**X**	**X**			
Pausar simulação	**X**	**X**			
Calcular comprimento da rua	**X**	**X**			
Representação de viadutos		**X**			
Representação de cruzamentos	**X**	**X**	**X**	**X**	**X**

Quadro 22 – Relação entre o trabalho proposto e os trabalhos correlatos

4 CONCLUSÕES

O presente trabalho é uma extensão do simulador de tráfego de automóveis em uma malha rodoviária desenvolvido por Freire (2004).

As principais extensões desenvolvidas são a visualização do simulador em três dimensões e a criação de um veículo guiado pelo usuário.

Os objetivos de disponibilizar um veículo guiado, visualização do simulador em três dimensões e reformulação da rotina de disponibilidades de trechos foram alcançados.

Para representação da malha viária e dos veículos circulando na mesma, são utilizadas rotinas matemáticas da geometria analítica e trigonometria, sendo que para prover a visualização da simulação são buscados subsídios na área de computação gráfica, utilizando a biblioteca gráfica OpenGl.

Ainda, para cada veículo especificado no simulador é criado um processo concorrente (*thread*), visto que os mesmos podem trabalhar de forma simultânea. Existem momentos onde um veículo vai depender do outro. Essa dependência ocorre quando existiam áreas já ocupadas, sendo que para isso, mecanismos de comunicação foram usados para evitar conflitos.

A velocidade que os veículos circulam na malha rodoviária é estabelecida pelo usuário, não sendo possível especificar uma velocidade especifica para um veículo. O sentido de tráfego dos veículos não pode ser alterado, desta forma o veículo fica impossibilitado de trafegar em sentido contrário (contra-mão). Veículos especiais (carros de polícia, ambulâncias) que possuem privilégios de circulação em uma malha rodoviária não podem ser representados neste simulador.

O simulador faz o controle de acesso nos cruzamentos pela ordem de chegada. O veículo que chegar primeiro terá o direito de passagem. Não existe semáforos nos cruzamentos.

4.1 EXTENSÕES

Sugere-se as seguintes extensões:

a) velocidade variável para os veículos, onde um veículo pode estar trafegando na malha em uma velocidade mais baixa que outro veículo;
b) inserção de semáforos para cruzamentos e controle dos mesmos. Permitir que seja representado semáforos em cruzamentos;
c) inserção de controladores de velocidade (lombadas eletrônicas).
d) rotina para fazer estatísticas sobre o tráfego na malha;
e) determinação de atributos para o trecho, como velocidade mínima e máxima, permitindo que seja configurado vias expressas com velocidade máxima superior a vias urbanas;
f) algoritmos para determinação de rotas com estimativa de melhor percurso e distância percorrida;
g) especificar um controle para amenizar as paradas dos veículos, permitindo que em situação onde há um cruzamento à frente ou veículos parados na pista, o carro vá diminuindo a velocidade gradativamente e não de forma instantânea;
h) criar carros especiais para situações de emergências (para uso da policia, bombeiros, entre outros), com definições de alguns privilégios diferenciados para circulação. Cita-se um dos privilégios para veículos especiais de emergência a circulação contra mão de direção na malha viária (FREIRE, 2004);
i) utilizar o modo de projeção perspectiva para obter maior realismo sobre a simulação.

REFERÊNCIAS BIBLIOGRÁFICAS

BARROS, Paulo Gonçalves; KELNER, Judith. Simulação de tráfego: uma experiência com realidade virtual. In: SYMPOSIUM ON VIRTUAL REALITY, 6., 2003, Ribeirão Preto. **Anais eletrônicos**, Ribeirão Preto: UFPE, 2003. Não paginado. Disponível em: <http://users.wpi.edu/ ~pgb/>. Acesso em: 10 jun. 2007.

BERTOLDI, Geferson. **Editor gráfico de ruas para o sistema de controle de tráfego em uma malha rodoviária urbana**. 2005. 56 f. Trabalho de Conclusão de Curso (Bacharel em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

BRASIL. **Código de trânsito brasileiro**. Brasília, DF: Casa Civil, Subchefia para Assuntos Jurídicos, 1997. Disponível em: < http://www.planalto.gov.br/ccivil/leis/L9503.htm>. Acesso em: 10 jun. 2007.

COHEN, Marcelo; MANSSOUR, Isabel Harb. **OpenGl uma abordagem prática e objetiva.** São Paulo: Novatec Editora Ltda, 2006.

COSTA, Fernando César Garcia da. **Trigonometria:** introdução. São Carlos, [1999]. Disponível em: <http://educar.sc.usp.br/licenciatura/1999/TRIGO.HTML>. Acesso em: 10 jun. 2007.

DENATRAN - Departamento nacional de trânsito. Brasília, [2004]. Disponível em: <http://denatran.gov.br/Legislacao.htm/>. Acesso em: 10 jun. 2007.

FREIRE, Jocemar José. **Simulador de tráfego de automóveis em uma malha rodviária**. 2004. 48 f. Trabalho de Conclusão de Curso (Bacharel em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

FROESCHLIN, Gustavo Eduardo Grahl. **Editor gráfico de ruas para o sistema de controle de tráfego de automóveis um uma malha rodovária urbana:** versão 2.0. 2006. 72 f. Trabalho de Conclusão de Curso (Bacharel em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

GONÇALVES, Zózimo Menna. **Curso de geometria analítica com tratamento vetorial.** Rio de Janeiro: Editora Científica, 1969.

HOFFMANN, Evandro Harrison; AEBI, Kelvin William Zimmermann; BORTOLETO, Silvio. Sistema de Roteirização Urbana. In CONGRESSO DE TECNOLOGIAS PARA GESTÃO DE DADOS E METADADOS DO CONE SUL, 4., 2006, Ponta Grossa. **Anais eletrônicos...** Ponta Grossa: Universidade Estadual de Ponta Grossa, 2006. Não paginado. Disponível em: < http://conged.deinfo.uepg.br/artigo1.pdf>. Acesso em: 10 jun. 2007.

MANSSOUR, Isabel Harb. **Introdução à OpenGl.** Porto Alegre, 2003. Disponível em <http://www.inf.pucrs.br/~manssour/opengl>. Acesso em: 10 jun. 2007.

MONTEIRO, Paulo. **Material para aulas.** Belo Horizonte, 2006. Disponível em: <http://etg.ufmg.br/~paulo/>. Acesso em: 10 jun. 2007.

PERSIANO, Ronaldo César Marinho; OLIVEIRA, Antônio Alberto Fernandes de. **Introdução à computação gráfica**. Rio de Janeiro: Livros Técnicos e Científicos Editora Ltda, 1989.

PINTO, Hebert F. **Trigonometria.** Rio de Janeiro: Editora Científica, 1958.

SEBESTA, Robert W. **Conceitos de linguagem de programação.** Tradução José Carlos Barbosa dos Santos. Porto Alegre: Bookman, 2000.

SILVA, Marcos Barros e. **Sistema multiagentes para gerenciamento de tráfego urbano**. 2005. 81 f. Dissertação (Mestrado em Ciências da Computação) – Centro de Ciências Exatas e Tecnológicas, Universidade Federal do Maranhão, São Luís. Disponível em: <http://www.dominiopublico.gov.br/pesquisa/PesquisaObraForm.do?select_action=&co_autor=5774>. Acesso em: 10 jun. 2007.

TOSCANI, Simão Sirineo; OLIVEIRA, Rômulo Silva de; CARISSIMI, Alexandre da Silva. **Sistemas operacionais e programação concorrente**. Porto Alegre: Sagra Luzzarro S/A, 2003.

VENTURE, Jacir J. **Álgebra vetorial e geometria analítica**. Curitiba: Unificado, [2005?]. Disponível em: <http://www.geometriaanalitica.com.br/>. Acesso em: 10 jun. 2007.

ZANUZ, Adriano. Definição de um sistema de software para simulação e supervisão de tráfego viário urbano. In: Semana Acadêmica do CPGCC, 3., 1998, Porto Alegre. **Anais eletrônicos**... Porto Alegre: Universidade Federal de Rio Grande do Sul, 1998. Não paginado. Disponível em: < http://www.inf.ufrgs.br/pos/SemanaAcademica/Semana98/zanuz.html>. Acesso em: 10 jun. 2007.

Apêndice A – Descrição dos métodos das classes `Carro`, `CarroGuiado`, `CarroRobo` e `Malha` descritas na especificação do sistema

A descrição dos métodos das classes `Carro`, `CarroGuiado`, `CarroRobo` e `Malha` é apresentada respectivamente no Quadro 23, Quadro 24, Quadro 25 e Quadro 26.

Classe Carro	
Método	**Descrição**
`AndaFimTrecho(var PtFree, var TrFree)`	Usado quando o veículo está no fim de um trecho e esse trecho possui uma continuação em um cruzamento
`AndaUmMetro(var PtFree, var TrFree)`	Usado quando o veículo pode andar mais um metro no mesmo trecho
`AndaXXXMetro(var PtFree, var TrFree)`	Usado quando o veículo está no final de um trecho para testar se o trecho é o fim da malha ou se existe um trecho de continuação
`CruzAlocado():boolean`	Testa se o cruzamento está livre para o veículo trafegar sem bater colidir.
`DesenhaCarro()`	Calcula os pontos necessário para desenhar o veículo na malha rodoviária
`MoveCarro(var PtFree, var TrFree)`	Faz o movimento do veículo dentro da malha. Aqui é verificado se o veículo está no final de um trecho ou se existe um cruzamento no final do trecho
`MoveCarroCruz()`	Faz o movimento do veículo dentro do cruzamento
`PegaNovoTrecho(var PtFree, var TrFree)`	Usado para definir um novo trecho quando o veículo chegou ao final de um trecho e não há cruzamentos
`PegaTdTrCruz(var PtFree, var TrFree)`	Usado para definir um novo trecho quando o veículo chegou ao final de um trecho e existe um cruzamento
`SorteiaTrecho(var Tr, TrUlt)`	Utilizado para sortear um novo trecho de continuação para o veículo robô ou o veículo guiado quando a opção Carro Guiado estiver desabilitada

Quadro 23 – Descrição dos métodos da classe `Carro`

Classe `CarroGuiado`	
Método	**Descrição**
`Create(MyPid)`	Cria um Carro Guiado
`DesenhaSelecao(var Inicio,fim,var largura)`	Desenha um retângulo azul no trecho selecionado pelo usuário para o veículo guiado
`TrechoPretendido(var PtFree, var TrFree)`	Utilizado para a escolha do trecho pelo usuário para movimentar o veículo guiado

Quadro 24 – Descrição dos métodos da classe `CarroGuiado`

Classe `CarroRobo`	
Método	**Descrição**
`Create(MyPid)`	Cria um Carro Robô

Quadro 25 – Descrição dos métodos da classe `CarroRobo`

Classe `Malha`	
Método	**Descrição**
`BtnExecutarClick(Sender)`	Evento `onClick` do botão Executar o qual inicia a simulação
`BtnParaClick(Sender)`	Evento `onClick` do botão Parar o qual pára a simulação
`MnAbrirClick(Sender)`	Evento `onClick` do botão Carregar Malha o qual abre uma caixa de texto onde o usuário selecionado o arquivo texto contendo as informações sobre a malha rodoviária

Quadro 26 – Descrição dos métodos da classe `Malha`