UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

AUTOMATIZAÇÃO DE LAUDOS DE TRÂNSITO

FABRÍCIO NICOLETTI

BLUMENAU
2006

2006/2-02

FABRÍCIO NICOLETTI

AUTOMATIZAÇÃO DE LAUDOS DE TRÂNSITO

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 Sistemas de Informação - Bacharelado.

Prof. Paulo César Rodacki Gomes, Dr. - Orientador

BLUMENAU
2006

2006/2-02

AUTOMATIZAÇÃO DE LAUDOS DE TRÂNSITO

Por

FABRÍCIO NICOLETTI

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

Presidente:	Prof. Paulo César Rodacki Gomes, Dr. – Orientador, FURB
Membro:	Prof. Francisco Adell Péricas, Me. – FURB
Membro:	Prof. Mauro Marcelo Mattos, Dr. – FURB

Blumenau, 12 de dezembro de 2006.

Em memória de Paulo Nicoletti

AGRADECIMENTOS

À Deus, pelo seu imenso amor e graça.

À minha namorada, pelo incentivo e paciência.

À minha família, por te me apoiado durante todo meu estudo.

Aos meus amigos, por estarem sempre por perto.

Ao professor Alexander Roberto Valdameri, por sua colaboração no desenvolvimento do Modelo de Entidades e Relacionamentos.

Ao meu orientador, Paulo César Rodacki Gomes, por tudo que me ensinou durante o desenvolvimento deste trabalho.

Um bom desenho é melhor do que um longo discurso.

Napoleão

RESUMO

Este trabalho apresenta o desenvolvimento de uma ferramenta para emissão de laudos de trânsito, que permite a criação do desenho do local da ocorrência, o cadastro das informações do acidente, dos veículos e pessoas envolvidas. Utilizando os conceitos de computação gráfica e orientação a objetos foi desenvolvido uma estrutura de dados para a criação dos diversos tipos de objetos gráficos disponíveis no sistema. O trabalho foi desenvolvido no ambiente de programação Delphi utilizando a biblioteca gráfica OpenGL para desenhar elementos gráficos no dispositivo de saída. Durante a etapa de validação do aplicativo junto com o especialista supervisor, foi constatado que com a utilização do sistema proposto é possível eliminar o retrabalho e o processo manual. Consequentemente agilizando o processo e diminuindo a possibilidade de erros.

Palavras-chave: CAD. OpenGL. Emissão de laudos de trânsito.

ABSTRACT

This work presents the development of a software tool to create and store transit accident information. The tool allows the creation of schematic drawings of the place where an accident occured, along with recordo f general information about the accident, the vehicles and people involved. In order to acomplish sush task, the software tool uses computer graphics and object oriented software design concepts, along with an specialized data structure to maintain the graphics elements needed to represent the accident's drawings. The software was inplemented with Delphi and OpenGL. During the tests performed, we concluded that the software can eliminate most part of repetitive tasks, reducing the time to generate all the legal documentation and reducing chances of errors.

Key-words: CAD. OpenGL. Emission of transit findings..

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API – *Application Programming Interface*

CAD – Desenho Assistido por Computador

CPU – Unidade de Processamento Central

DETRAN – Departamento de Trânsito

GLU – *OpenGL Utility Library*

ISO – *International Organization for Standardization*

MER – Modelo de Entidades e Relacionamentos

OO – Orientação a objetos

OpenGL – *Open Graphics Library*

SETERB – Serviço Autônomo Municipal de Terminais Rodoviários de Blumenau

SQL - Structured Query Language

UML – Linguagem de Modelagem Unificada

SUMÁRIO

1 INTRODUÇÃO

A guarda de trânsito de Blumenau é o órgão responsável por atender todas as ocorrências de acidentes que ocorrem dentro da cidade de Blumenau. Para cada acidente assistido pelos agentes é elaborado um laudo contendo as informações dos envolvidos no acidente e um desenho reconstituindo o local da ocorrência.

No local do acidente os guardas fazem um rascunho manual do acidente e preenchem formulários com as informações coletadas e, ao chegarem no departamento refazem o desenho e transferem as informações do acidente para o sistema utilizado atualmente. No atual sistema de atendimento de acidentes em Blumenau, os agentes exercem muito trabalho manual e redigitação de informações. Ambos geram grandes problemas como desperdício de tempo e maior probabilidade de erros.

Para solucionar estes problemas este trabalho apresenta o desenvolvimento de um software capaz de fornecer as ferramentas necessárias para que o guarda, ao atender uma ocorrência, possa desenhar o croqui ilustrativo do acidente e inserir todas as informações diretamente no sistema. Segundo Azevedo e Conci (2003), imagens que exigiriam do artista o uso de uma técnica apurada de desenho podem ser geradas mais facilmente com o auxílio de um software. Com o auxílio da computação gráfica, espera-se que os desenhos se tornem mais precisos, fornecendo um maior número de informações, além maior qualidade gráfica. Espera-se também que o sistema não exija uma habilidade muito grande de desenho por parte dos guardas de trânsito.

Para que o sistema possa ser levado até o local da ocorrência é necessário a utilização de um computador móvel, que possa ser levado junto com guardas dentro de sua viatura. Segundo Dalfovo (2004), aliando a tecnologia de computação móvel e a de sistemas de informação gera-se um diferencial competitivo de grande valor, trazendo para as organizações muitos benefícios. Com a utilização de computador móvel, o processo manual é eliminado, pois o desenho e as informações coletadas do acidente são implantados diretamente no sistema e no local da ocorrência.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é criar um sistema com interface gráfica vetorial para automatizar a emissão dos laudos de acidentes para a Guarda Municipal de Trânsito de Blumenau.

Os objetivos específicos do trabalho são:

a) o sistema deve possuir uma interface gráfica que possibilite aos guardas reconstituírem a cena do acidente em forma de desenho;
b) o sistema deve receber e armazenar todas as informações necessárias a respeito do acidente e dos envolvidos;
c) o sistema deve imprimir o laudo contendo todas as informações do acidente e o desenho do local.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está organizado em quatro capítulos. No primeiro capítulo tem-se a introdução, a justificativa do trabalho, o objetivo geral e objetivos específicos. No segundo capítulo é apresentada a fundamentação teórica utilizada para o desenvolvimento do trabalho. Apresenta-se a descrição do problema a ser solucionado, os principais conceitos utilizando para o desenvolvimento do trabalho e os trabalhos correlatos. No capítulo três, estão descritas as etapas do desenvolvimento, a descrição do sistema atual e do sistema proposto, os requisitos do sistema, a especificação, detalhes da implementação e os resultado obtidos com a aplicação. No quarto e último capítulo estão descritas as conclusões do trabalho e sugestões de possíveis extensões que podem ser realizadas no trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

A guarda de trânsito de Blumenau tem sua origem em meados dos anos 40 mas foi criada oficialmente em 1955, com isso ela é o serviço municipal de trânsito mais antigo do país (SETERB, 2006). Na ocorrência de um acidente com vítimas a guarda de trânsito e o corpo de bombeiros são acionados imediatamente, e ao chegarem ao local atendem e encaminham as vítimas afim de preservar suas vidas e tomam outras medidas necessárias, de acordo com os procedimentos legais. Neste intervalo de tempo, com base nos documentos das pessoas e dos veículos envolvidos, os agentes elaboram um laudo do acidente contendo também um desenho em escala da cena após o acidente.

Apesar de estar a mais de 50 anos atendendo os cidadãos de Blumenau a guarda não possui um sistema de informação que viabiliza o atendimento da ocorrência. Atualmente todo o processo de atendimento no local do acidente se da manualmente através do preenchimento de formulários. O desenho do local também é feito manualmente tornando mais difícil o trabalho dos agentes, pois por seu um desenho técnico exige que os mesmo utilizem várias instrumentos de desenho para que se possa relatar fielmente a cena. Para que os envolvidos no acidente possam ser liberados sem demora do local do acidente e para descongestionar o trânsito decorrente do mesmo os agentes preenchem rapidamente os formulários e fazem apenas um esboço da cena com as medidas principais do local. E ao chegarem no departamento passam as informações dos formulários para o software utilizado atualmente e refazem o desenho do acidente.

A principal dificuldade no sistema atual é elaboração do desenho contendo a cena do acidente, por ser um desenho técnico que necessita de dimensões precisas ele acaba por necessitar de muito tempo e talento dos agentes. Este é o principal objetivo deste trabalho criar uma ferramenta gráfica que facilite a elaboração dos desenhos.

Outra problemática deste sistema é o retrabalho que ele gera para os agentes. Como todas as informações são coletadas manualmente a chegarem no departamento eles são obrigados a repassarem elas para o sistema. Esse processo além de desperdiçar o tempo dos agentes pode gerar erros de digitação e/ou de entendimento das informações escritas no formulário. Segundo Dalfovo (2004), automatizar os processos de produção e eliminar o fluxo de papéis é o caminho mais lógico. Este é outro objetivo deste trabalho fazer com que o retrabalho seja eliminado do atendimento, para isso os formulário de papel darão lugar a formulários eletrônicos inseridos na ferramenta e diretamente associados ao desenho.

Podendo assim armazenar todas as informações do acidente diretamente no sistema incluindo o desenho da cena. Porém este trabalho não tem como objetivo interligar a ferramenta desenvolvida, com o software utilizado atualmente pela guarda.

Com plantão de 24 horas, em todo o município, a guarda de trânsito de Blumenau atende atualmente cerca de 25 acidentes diários de segunda a quinta feira aumentando esse número para 35 ocorrências nos finais de semana (SETERB, 2006). Para que seja eliminado o retrabalho no atendimento destas ocorrências, é necessário utilizar o software no local do acidente, desta maneira coletano os dados e inserindo-os diretamente no sistema sem a utilização de papéis. Para que isto se torne possível é necessário a utilização de um computador móvel que possa ser levando pelo agente. Com o avanço da computação móvel nos últimos anos os computadores tem ficado cada vez menores e mais potentes, um exemplo desta evolução é o *PalmTop*. Porém a criação do desenho em um dispositivo com uma área de tela muito pequena se torna inviável, por isso um *TabletPC* ou um *Notebook* são soluções mais viáveis.

2.1 COMPUTAÇÃO MÓVEL

A computação móvel vem surgindo com uma nova proposta de paradigma computacional. O usuário, portando um dispositivo móvel, como *TabletPC* ou *Notebook*, tem acesso a um infra-estrutura compartilhada independente da sua localização física. Isto fornece uma comunicação flexível entre os usuários.

> Computação móvel representa um novo paradigma computacional. Surge como uma quarta revolução na computação, antecedida pelos grandes centros de processamento de dados da década de sessenta, o surgimento dos terminais nos anos setenta, e as redes de computadores na década de oitenta. (MATEUS E LOUREIRO, 1998)

A computação móvel permite que os usuários tenham acesso a serviços independentemente de onde estão localizados, e o mais importante, de mudanças de localização, ou seja, sua principal característica é a mobilidade. Esta por sua vez se vê necessária para atender a necessidade de implantar um sistema de informação para pessoas que não possuem um ambiente de trabalho fixo. Isso possibilita que o usuário possa utilizar seu computador em qualquer lugar, conectado ou não a uma rede de dados.

Os avanços na tecnologia de transmissão de dados sem fio (*Wireless*) associadas aos dispositivos móveis, eliminaram a necessidade de um usuário manter-se conectado a uma

infra-estrutura fixa (WIKIPEDIA, 2006). Isso permite ao usuário uma maior liberdade, já que desde que esteja dentro da área coberta por sua rede sem fio, ele pode locomover-se livremente e mesmo assim permanecer conectado. De outra forma, ao entrar dentro da área de cobertura de uma outra rede é possível conectar-se a ela também desde que possua a autorização necessária. Ou seja, um indivíduo em posse de um dispositivo móvel equipado com um equipamento de rede sem fio pode se conectar a inúmeras redes de computadores e desta forma trabalhar em diversos lugares.

Atualmente o mercado oferece um enormidade de dispositivos móveis. O mais importante deles e o mais vendido é o *Notebook*, que consiste em um CPU com um monitor de cristal líquido, teclado e mouse. Este dispositivo possui as mesmas características de um computador normal, seus diferenciais são o tamanho reduzido, podendo chegar ao tamanho de um caderno universitário, e seu peso mínimo que pode chegar a cerca de um quilo. Outro equipamento interessante é o *TabletPC* que é muito semelhante a um *Notebook.* O conceito deste dispositivo foi originado das antigas pranchetas de anotações, ou seja, trata-se de um tela de cristal líquido sem teclado e sem mouse e para acessá-lo utiliza-se um caneta de toque especial sobre a tela. O *TabletPC* não pode ser considerado um *PalmTop*, pois seu tamanho e capacidade de processamento são maiores, excluindo-os desta classificação.

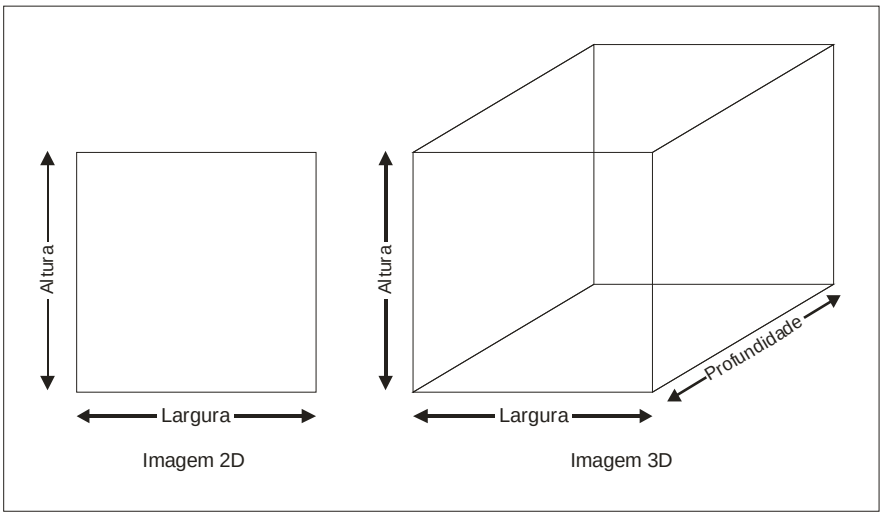
2.2 COMPUTAÇÃO GRÁFICA

A computação gráfica é uma área da ciência da computação que se dedica ao estudo e ao desenvolvimento de técnicas e algoritmos para geração, manipulação e análise de imagens pelo computador (COHEN E MANSSOUR, 2006). Segundo Azevedo e Conci (2003), a ISO definiu que a computação gráfica é um conjunto de ferramentas e técnicas para converter dados para ou de um dispositivo gráfico através do computador. Ou seja, ela estuda os métodos necessários para transformar as informações contidas na memória do computador em imagens que serão exibidas no monitor.

Conforme Gomes e Velho (2003), desde sua origem a computação gráfica estuda os métodos que permitem a visualização de informações armazenadas na memória do computador. Para transformar as informações da memória em imagens a computação gráfica utiliza a matemática. Conceitos de trigonometria e geometria são fundamentais para a criação de um software gráfico. Todo o objeto gerado na tela e constituído por *pixels*, que são pontos

que fazem com que a imagem seja sintetizada visualmente (WIKIPEDIA, 2006). Utilizando fórmulas matemáticas a computação gráfica transforma um conjunto dados da memória em um conjunto de *pixels* devidamente ordenados para que formem a imagem na tela. Segundo Azevedo e Conci (2003), as transformações geométricas podem ser representadas na forma de equações, o problema é que manipulações de objetos gráficos normalmente envolvem muitas operações de aritmética simples. Com o intuito de agilizar e facilitar o processo de criação de softwares gráficos, nos anos 90 começaram a surgir as bibliotecas gráficas. Estas bibliotecas se ocupam com a parte matemática necessária para criar os objetos e exibi-los na tela, assim como suas transformações posteriores. Desta forma não há necessidade do programador implementar as equações necessárias para criação e exibição dos objetos.

A computação gráfica permite a criação de desenhos em duas dimensões (2D) e em três dimensões (3D). A principal diferença entre elas é maneira de como observamos os objetos na tela. Em um desenho 2D podemos visualizar a largura e a altura do objeto, enquanto que em um desenho 3D podemos visualizar também a sua profundidade. A figura 1 mostra um exemplo de desenho 2D e de um desenho 3D. Um desenho bidimensional leva em consideração somente um lado do objeto e é possível visualizá-lo somente do ângulo em que foi criado. Já em um desenho tridimensional todos os lados do objeto são considerados, ou seja, um desenho 3D é a transcrição de um objeto real para a tela do computador e é possível visualizá-lo de qualquer ângulo. Como a maioria dos dispositivos de saída são bidimensionais (monitores, impressoras, etc.) os desenhos tridimensionais devem ser convertidos para bidimensionais. Segundo Azevedo e Conci (2003), para que a percepção de profundidade seja transmitida a computação gráfica utiliza técnicas de desenho como luzes, sobras, texturas, perspectiva entre outras que forma criadas por artistas a vários séculos. Desta maneira é possível transmitir a percepção de espaço para o usuário.



Fonte: adaptado de Woo (1999).

Figura 1 – Exemplo de imagem 2D e 3D.

2.3 DESENHO ASSISTIDO POR COMPUTADOR – CAD

Segundo Denis e André (1992), o CAD, é uma prancheta de desenho eletrônica criada pela informática. CADs são sistemas computacionais utilizados para facilitar o desenvolvimento de desenhos técnicos. Desenho técnico é um meio de expressão que permite a transmissão de informações técnicas com a ajuda de indicações diversas, ligadas a uma representação gráfica dos objetos pretendidos.

Antes da criação de sistemas CAD o processo de desenho técnico era realizado manualmente utilizando papel, lápis e ferramentas que auxiliavam no desenvolvimento com réguas e gabaritos. Esse processo manual exigia muito tempo de trabalho para ser realizado e quando havia necessidade de alteração, geralmente, todo o desenho tinha de ser refeito. Os gastos com matéria-prima também começavam a preocupar as empresas. Segundo Denis e André (1992), um avião projetado pela *AirBus* utilizava cerca de um milhão de plantas manuais. Isso significa algumas toneladas de papel que devem ser armazenadas em um local seguro para que não sejam extraviadas e nem estragadas. Todo este processo exigia um elevado grau de investimento e muita mão de obra. Esses foram os principais motivos para a criação dos sistemas CAD, substituir as antigos desenhos em papel por arquivos eletrônicos.

Um sistema CAD como o próprio nome diz foi criado com o intuito de auxiliar os desenhista na criação dos desenhos. Por isso as ferramentas utilizadas na confecção dos desenhos manuais como as réguas e os gabaritos tiverem que ser reproduzidas na aplicação.

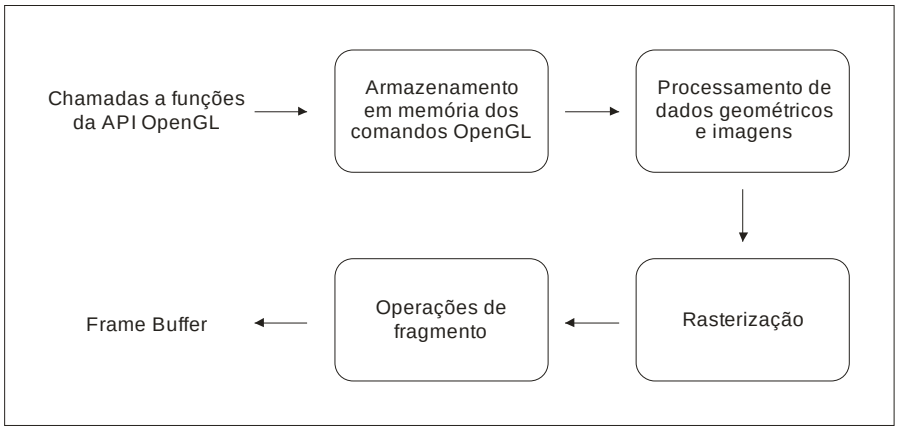
Atualmente considera-se que um sistema CAD deve fornecer uma série de ferramentas para construção entidades planas como linhas, curvas e polígonos e também ferramentas capazes de manipular e relacionar estes objetos. Apesar de todas as suas facilidades, os softwares CAD costumam ser utilizados por um nicho pequeno de usuários, devido a sua intensa especialização e seu alto custo e, costumam também exigir um elevado investimento em hardware.

2.4 OPENGL

Cohen e Manssour (2006) definem *OpenGL* como uma poderosa e sofisticada API para a criação de programas gráficos 2D e 3D para diversas plataformas, de potentes estações de trabalho a simples computadores pessoais. O *OpenGL* consiste em torno de 250 funções. Destas, 200 são da própria biblioteca que oferecem um conjunto de primitivas gráficas com pontos, linhas e polígonos. As outras 50 funções fazem parte da biblioteca GLU que oferece funções de modelagem tais como curvas e superfícies.

Antes da criação do *OpenGL* a indústria da computação gráfica se restringia somente a programadores experientes com alto grau de conhecimento em linguagens de programação. Isso se deve ao fato de que a computação gráfica exigir uma enorme quantidade de cálculos matemáticos para a criação da cena na memória e exibição da mesma na tela. Atualmente todas as rotinas matemáticas necessárias são realizadas pelo *OpenGL*. Desta forma, em vez de descrever detalhadamente um cena 2D ou 3D, basta especificar o conjunto de passos que devem ser seguidos para se obter o aspecto ou efeito desejado (COHEN E MANSSOUR, 2006).

O processo de geração de imagens do *OpenGL* é chamado de renderização *Pipeline.* A figura 2 mostra um versão simplificada do processo.



Fonte: adaptado de Cohen e Manssour (2006).

Figura 2 – Versão simplificada da renderização *pipeline* do *OpenGl.*

Nesse processo todos os comandos feitos pela aplicação são armazenados em uma memória específica depois os dados geométricos (por exemplo, vértices) e os dados de imagem (por exemplo, os *pixels*) são processados de forma diferente, mas ambos passam pelo estágio de rasterização, que consiste na transformação dos dados em fragmentos. Esses fragmentos são posições na tela que possuem informações como cor, profundidade e coordenadas de textura. Cada fragmento contribui para atualização do *pixels* do *frame buffer*, que corresponde à memória do dispositivo gráfico. No final deste processo a imagem é exibida na tela do monitor.

2.5 TRABALHOS CORRELATOS

Durante o processo de pesquisa bibliográfica para a realização deste trabalho foram encontrado quatro trabalhos correlatos que se destacaram por abranger assuntos relacionado a este trabalho. O sistema de simulação do controle de tráfego de automóveis em uma malha rodoviária urbana (FREIRE, 2004) tem como objetivo principal simular e verificar o comportamento do tráfego de veículos automotores em uma malha rodoviária pré-definida. Um de seus objetivos específicos é disponibilizar ao usuário a visualização do tráfego de veículos sobre uma malha rodoviária. Para que este objetivo fosse alcançado Freire (2004) utilizou a linguagem de programação *Delphi* associada as bibliotecas do *OpenGL* para desenhar as malhas rodoviárias bem como os veículos que transitam sobre ela.

O editor gráfico de ruas para o sistema de controle de tráfego de automóveis em uma malha rodoviária urbana (EMGR) desenvolvido pelo acadêmico Bertholdi (2004), foi

desenvolvido para completar o trabalho de Freire. O trabalho tem como objetivo a construção de um editor gráfico 2D que crie uma malha viária graficamente, e após armazená-la em um arquivo que seja compatível com o sistema de controle de trânsito desenvolvido por Freire (2004). Para desenvolver este trabalho o acadêmico também utilizou as bibliotecas gráficas do *OpenGL* juntamente com o *Delphi* para criar uma ferramenta onde o usuário pudesse criar e alterar malhas rodoviárias que poderiam ser carregadas e analisados no sistema de controle de tráfego. Mais tarde Froeschlin (2006) propôs melhorias ao sistema EMGR. Seu principal objetivo era remodelar o software usando OO, corrigir funções deficientes e agregar novas funcionalidades ao sistema. Em sua nova abordagem ele definiu que cada tipo de objeto teria uma classe com seus métodos e atributos, e que cada elemento gráfico inserido no desenho seria um objeto derivado destas classes. Mas apesar desta mudança de paradigma o trabalho continuou sendo desenvolvido em *Delphi* e *OpenGL*.

Outro trabalho relevante é do acadêmico Witte (2005), que após analisar a empresa Walmor Renovadora de Pneus, criou o sistema de automação de recapadoras de pneus com a coleta de dados via *PalmTop*. Em sua análise ele percebeu que ao atender um cliente o vendedores da empresa preenchiam os pedidos manualmente. Para resolver este problema ele propôs a criação de um sistema de informação de coleta de dados via *PalmTop*, ou seja, utilizando a tecnologia de computação móvel. A correlação com este trabalho se dá no fato do acadêmico utilizar a computação móvel para eliminar o processo manual afim de evitar erros e desperdício de tempo.

3 DESENVOLVIMENTO DO TRABALHO

Este capítulo descreve todos os detalhes relevantes e pertinentes ao processo de desenvolvimento da ferramenta CAD proposta. Para que o trabalho obtesse êxito ao término do prazo de entrega, foram estabelecidas etapas de trabalho na proposta. Estas por sua vez, foram seguidas durante o processo de elaboração e abaixo seguem listadas:

a) levantamento de requisitos: nesta etapa foi realizado entrevistas com os guardas de trânsito e com a diretoria do SETERB. Também foi recolhido o manual de trânsito, desenhos manuais e o formulário de laudo. Todas essas informações foram analisadas para estabelecer os requisitos do sistema;

b) pesquisa sobre computação gráfica e *OpenGL*: essa etapa foi destinada a um estudo aprofundado da utilização de computação gráfica e *OpenGL* através de livros;

c) documentação: nesta etapa foi realizada a documentação do sistema utilizando a UML. Com o auxílio do *Enterprise Architect* foram desenvolvidos os diagramas UML de casos de uso, classes e seqüência. Nesta etapa também foi desenvolvido o diagrama de entidades e relacionamentos com o auxílio do aplicativo *Power Designer;*

d) implementação: nesta etapa o sistema foi implementado em *Delphi* utilizando a orientação a objetos. Para criar a interface gráfica foi utilizada a biblioteca gráfica *OpenGL*. Para armazenar as informações foi utilizado banco de dados *MySQL*;

e) teste: durante a etapa de implementação foram realizados testes de unidade, ou seja, cada função foi testada no término de sua implementação. No final da implementação do sistema foi feito o teste de integração onde todas as funcionalidades do sistema foram checadas;

f) validação: nesta etapa o sistema foi apresentado a supervisora especialista da aplicação, Sra. Odete Brancher Becker, e ao analista de sistemas do SETERB, Sr. José Vilson Bertoldi para avaliação das funcionalidades do sistema.

g) relatório final: nesta etapa foi desenvolvido o texto final, demonstrando detalhadamente as etapas do processo de criação do sistema.

3.1 SISTEMA ATUAL

Todo acidente atendido pela guarda de trânsito resulta em um laudo, contendo os dados dos veículos, condutores, vítimas, testemunhas e um desenho reconstituindo o local do acidente. Segundo Pacheco (2005), o processo de coleta de informações e desenho são feitos manualmente no local da ocorrência através de preenchimento de formulários (anexo A). Após atender a ocorrência o guarda se dirige ao DETRAN, onde passa para o sistema atual as informações dos veículos e condutores envolvidos no acidente. O guarda também desenvolve um desenho em escala de 1/200 mm, reconstituindo o local do acidente. Conforme Souza (2005), por não haver um sistema de desenho, os guardas desenvolveram uma maneira de agilizar o processo. Eles criaram croquis dos principais locais de Blumenau, e referenciaram-nos com números (anexo B), para que possam ser localizados através de uma tabela (anexo C), fornecendo o nome da rua e um ponto de referência. Desta maneira ao ocorrer um acidente nestes locais não há necessidade de desenhar toda a via, pois eles podem fotocopiar o croqui e completar com as informações necessárias.

Após ser finalizado, o desenho, ele é encaminhado junto com os formulários para o setor de laudos, para que o restante das informações coletadas no local do acidente sejam passadas para o sistema. Após o término do preenchimento de todas as informações o encarregado do setor confere os dados do sistema com os do formulário a fim de localizar erros de digitação. Após o término deste processo o guarda recebe uma cópia impressa do laudo para conferir as informações do mesmo, caso ele encontre algum erro, o laudo volta para o setor de laudos para que sejam realizadas as correções necessárias.

Rhenius (2005), afirma que o sistema atual não atende as necessidades da guarda, pois os desenhos devem ser feitos manualmente e as informações coletadas devem ser repassadas para o sistema. Pacheco (2005), completa afirmando que o processo atual está envolvendo de 4 a 5 pessoas quando poderia envolver somente o guarda. A falta de um sistema de desenho gráfico e o retrabalho são os principais defeitos no processo. O desenho do local do acidente deve ser muito detalhado e preciso, todas as medidas devem corresponder com a realidade, dessa maneira não é possível fazê-lo no local da ocorrência, pois exige instrumentos e tempo não disponíveis. Por isso no local o guarda faz somente um rascunho e ao chegar ao departamento ele desenvolve a versão final do desenho. O mesmo acontece com as informações das pessoas envolvidas no acidente, no local os guardas preenchem os formulários manualmente e ao chegarem no departamento estas informações são passadas

para o sistema. O processo atual gera um retrabalho demasiado sobre as informações coletadas, ocasionando outros problemas como erros de digitação, desperdício de tempo e de funcionários.

Segundo Becker (2005), os sistema de informação utilizado atualmente pela guarda de trânsito não foi terminado pois houve um quebra de contrato com a empresa. O sistema utilizado pela guarda pode ser descrito como um protótipo de teste de uso, pois não possui todas as funcionalidades necessárias para a emissão dos laudos. Em alguns casos o sistema não possibilita a entrada de alguns dados necessários no mesmo que devem ser preenchidos manualmente. Becker (2005), afirma que não é possível corrigir o sistema atual pois o mesmo não esta devidamente documentado tornando difícil o entendimento do código fonte.

Segundo Dalfovo (2004), automatizar os processos de produção e eliminar o fluxo de papéis é o caminho mais lógico. Desta maneira com a implantação de um sistema de coleta de dados no local da ocorrência com interface gráfica onde é possível desenhar a cena do acidente, pode-se eliminar o processo manual e o retrabalho, consequentemente os problemas gerados por estes dois fatores.

3.2 SISTEMA PROPOSTO

A computação móvel traz um diferencial para as organizações, pois fornece mobilidade e o acesso as informações de maneira mais rápida (DALFOVO, 2004, p. 275). Aproveitando-se deste novo nicho de mercado, o sistema proposto utiliza a tecnologia de computadores móveis mas especificamente os *Notebooks*.

O sistema continuará utilizando o processo de croquis desenvolvido pela guarda de trânsito conforme descrito anteriormente. Ao atender uma ocorrência, o guarda busca o nome da rua e ponto de referência do local, com estes dados o sistema abre o croqui do local e exige o preenchimento das informações referentes ao tipo de acidente, condições da via e do meio ambiente. Para montar a reconstituição do local, o sistema disponibiliza os elementos gráficos necessários como placas, veículos, pedestres e figuras geométricas. Estes elementos gráficos podem ser deslocados e girados de acordo com o necessário. Para uma melhor identificação dos elementos colocados no croqui, o sistema possui uma descrição para cada item que pode ser preenchida pelo guarda, se necessário. Da mesma forma para cada veículo colocado no croqui o sistema possibilita o preenchimento das informações do veículo, do condutor, das

vítimas e a versão do acidente segundo o motorista do mesmo. Todo o desenho no sistema estará em escala real, pois as medidas devem ser precisas, por isso, o sistema disponibiliza um elemento gráfico que permite ao usuário medir as distâncias entre os pontos do desenho. Para que o guarda chegue a uma maior precisão no desenho o sistema deverá possuir uma ferramenta que aumenta ou diminui o desenho (Zoom) conforme desejado. O sistema também possibilita abrir um laudo para modificações onde é possível alterar todas as informações contidas no mesmo.

A implantação do sistema acarreta na mudança do processo de emissão dos laudos. Com a utilização dos *Notebooks* no atendimento das ocorrências o processo de desenho e preenchimento manual dos formulários é eliminado. Utilizando o equipamento o guarda faz o desenho e preenche as informações referentes ao acidente diretamente no sistema e no local da ocorrência. O sistema também deve permitir a impressão do laudo. Todas as informações coletadas no local do acidente devem ser impressas de acordo os laudos atuais e a impressão do desenho deve estar na escala de 1/200.

Com a utilização do sistema, o retrabalho detectado no processo atual deverá ser eliminado, e conseqüentemente espera-se diminuir a ocorrência de erros de digitação. Como não será mais necessário repassar os dados para o sistema e refazer o desenho espera-se que o tempo necessário para emissão de laudos também diminua. Conforme Pacheco (2005), com o sistema proposto o número de pessoas envolvidas no processo também pode ser reduzido, pois o próprio guarda pode imprimir o laudo eliminando assim a necessidade de um setor de laudos como existe no processo atual.

3.3 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A seguir são apresentados os requisitos funcionais e não funcionais , atendidos pela ferramenta. O quadro 1 apresenta os requisitos funcionais e sua rastreabilidade, ou seja, vinculação com o caso de uso associado.

REQUISITOS FUNCIONAIS	CASO DE USO
RF01: O sistema deverá permitir que o usuário abra os croquis fornecendo o nome da rua e o ponto de referência.	UC01
RF02: O sistema deverá permitir o cadastro das informações gerais do acidente.	UC02
RF03: O sistema deverá permitir o cadastro do tipo de acidente.	UC02
RF04: O sistema deverá permitir o cadastro das condições da via e do meio ambiente no local do acidente.	UC02
RF05: O sistema deverá permitir o cadastro do veículo.	UC03
RF06: O sistema deverá permitir o cadastro do condutor do veículo.	UC03
RF07: O sistema deverá permitir o cadastro de vítimas do acidente.	UC03
RF08: O sistema deverá permitir o cadastro da declaração dos envolvidos no acidente.	UC03
RF09: O sistema deverá permitir o cadastro de testemunhas do acidente.	UC03
RF10: O sistema deverá permitir inserir o elemento gráfico veículo.	UC04
RF11: O sistema deverá permitir inserir o elemento gráfico pedestre.	UC04
RF12: O sistema deverá permitir inserir o elemento gráfico objeto.	UC04
RF13: O sistema deverá permitir inserir o elemento gráfico placa de sinalização.	UC04
RF14: O sistema deverá permitir o desenho de uma reta.	UC05
RF15: O sistema deverá permitir o desenho de um quadrilátero.	UC05
RF16: O sistema deverá permitir o desenho de um círculo.	UC05
RF17: O sistema deverá permitir deslocar os elementos gráficos colocados no desenho.	UC06
RF18: O sistema deverá permitir girar os elementos gráficos inseridos no desenho.	UC06
RF19: O sistema deverá permitir alterar o tamanho dos objetos no desenho	UC06
RF20: O sistema deverá permitir que o usuário meça a distância de um ponto até outro dentro no desenho.	UC07
RF21: O sistema deverá permitir aproximar-se ou afastar-se de um determinado local do desenho (Zoom).	UC08
RF22: O sistema deverá permitir inserir texto no desenho.	UC09
RF23: O sistema deverá permitir salvar o desenho e os dados coletado.	UC10
RF24: O sistema deverá permitir abrir arquivos salvos pelo sistema.	UC10
RF25: O sistema deverá permitir que o desenho e os dados sejam impressos na forma de um laudo.	UC11

Quadro 1: Requisitos funcionais.

No quadro 2 são relacionados os requisitos não funcionais.

REQUISITOS NÃO FUNCIONAIS
RNF01: O sistema deverá ser implementado em *Delphi*.
RNF02: O sistema deverá utilizar *OpenGL* para desenhar os elementos gráficos.
RNF03: O sistema deverá utilizar desenhos vetoriais.
RNF04: O sistema deverá utilizar o banco de dados *MySQL*.

Quadro 2: Requisitos não funcionais.

3.4 ESPECIFICAÇÃO

A modelagem do sistema foi criada utilizando-se a UML. A seguir são apresentados o diagrama de casos de uso, o diagrama de classes e os diagramas de seqüência. que foram criados com o auxílio da ferramenta case *Enterprise Architect*. Para a criação da base de dados foi utilizado o MER apresentado no último subitem, este modelo foi gerado com a utilização da ferramenta *Sybase PowerDesigner*.

3.4.1 Diagrama de casos de uso

Segundo Bezerra (2002), modelagem de casos de uso é uma representação das funcionalidades externamente observáveis do sistema e dos elementos externos ao sistema que interagem com ele. Este modelo é composto de casos de uso, de atores e de relacionamentos entre eles. A integração destes componentes é apresentada na figura 3 que representa o diagrama de casos de uso desenvolvido para o trabalho.

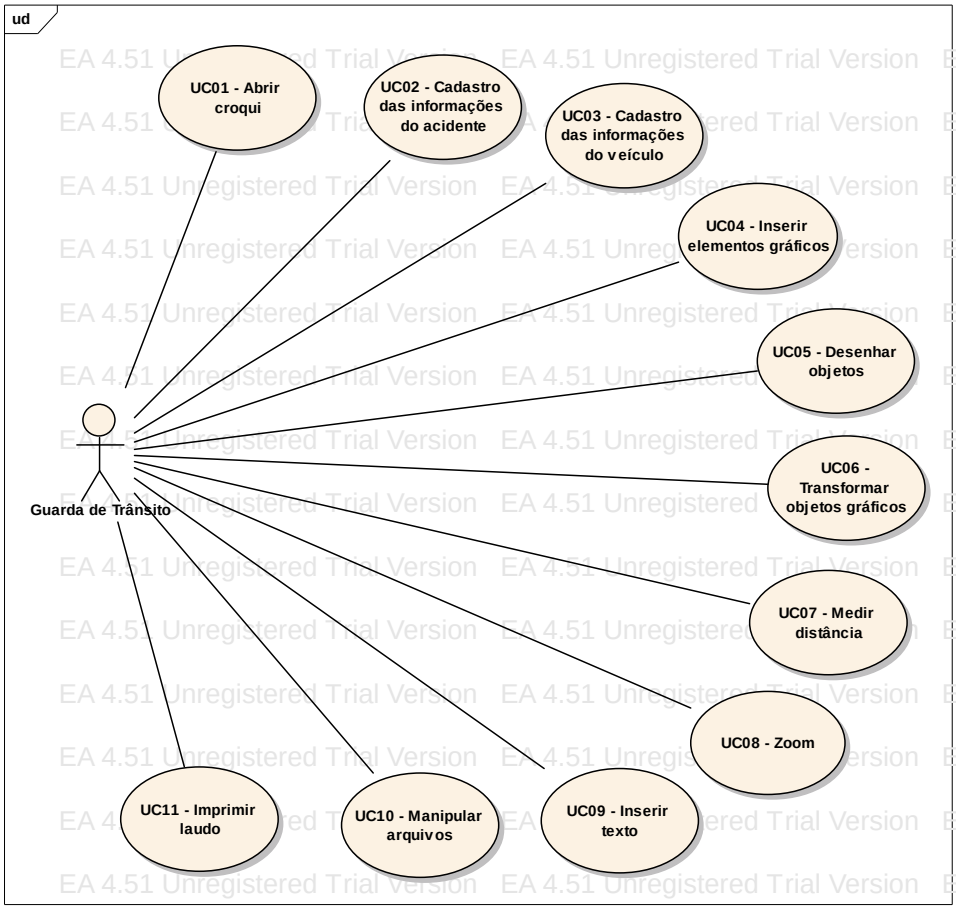


Figura 3: Diagrama de casos de uso.

3.4.2 Diagrama de classes

Na figura 4 é apresentado o diagrama de classes desenvolvido para a aplicação. As classes contidas no modelo apresentam os atributos e métodos mais importantes para o entendimento do mesmo. Apresenta-se ainda os relacionamentos possíveis entre as instâncias de cada classe.

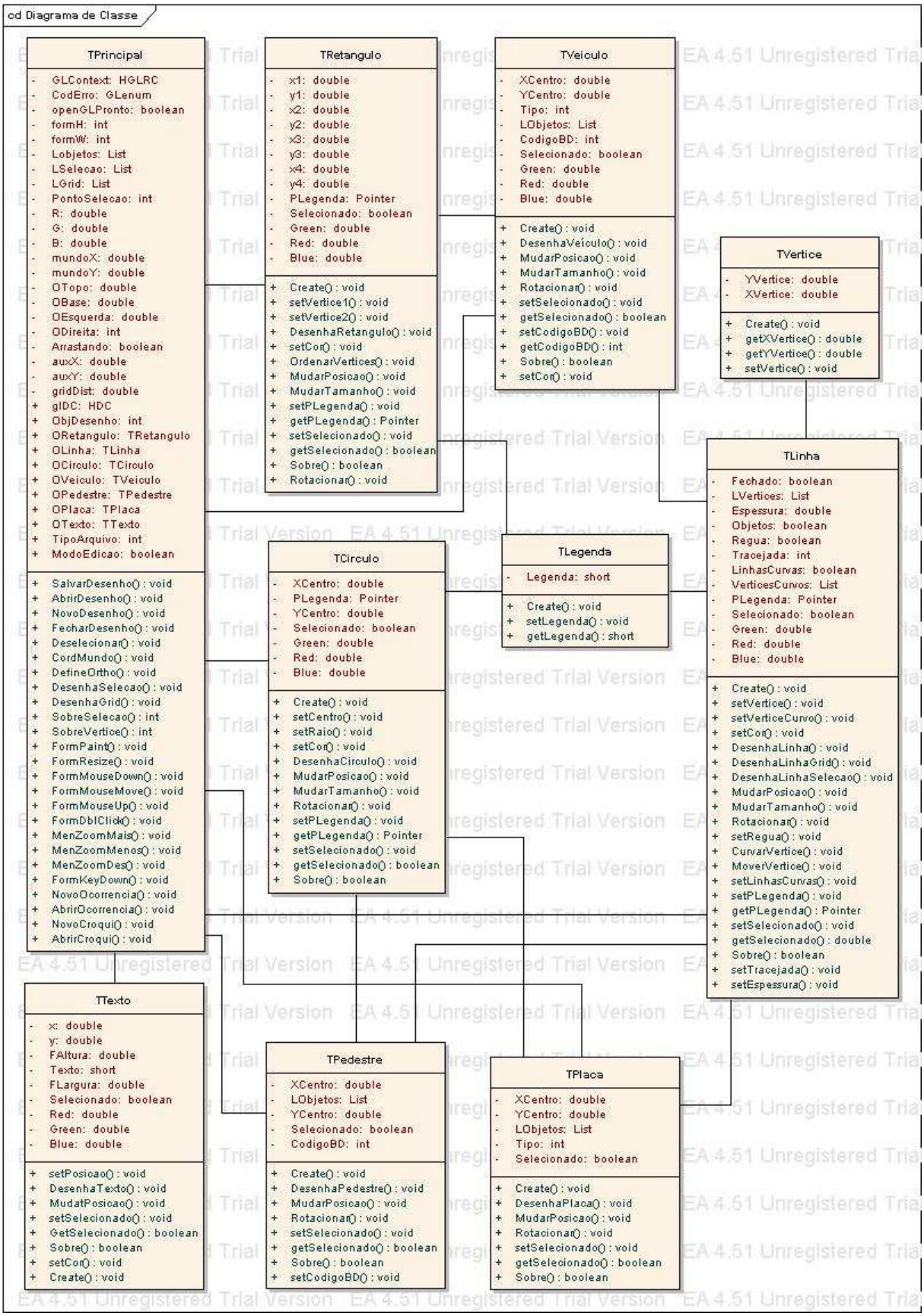


Figura 4: Diagrama de classes.

3.4.3 Diagramas de seqüência

Para melhor compreensão dos relacionamentos existentes entre os objetos foram desenvolvidos os diagramas de seqüência. Segundo Bezerra (2002), estes diagramas ajudam a documentar e a entender os aspectos dinâmicos do sistema, eles descrevem a seqüência de mensagens enviadas e recebidas pelos objetos que participam em um caso de uso. O principal foco do diagrama de seqüência esta em como as mensagens são enviadas no decorrer do tempo.

A figura 5 apresenta a seqüência de mensagens necessárias para que o sistema crie um novo croqui. O primeiro método chamado verifica se existe algum desenho aberto, caso haja o sistema fecha o desenho. Em seguida o sistema cria um novo desenho, configura o *OpenGL*, cria a grade e finalmente mostra o desenho na tela.

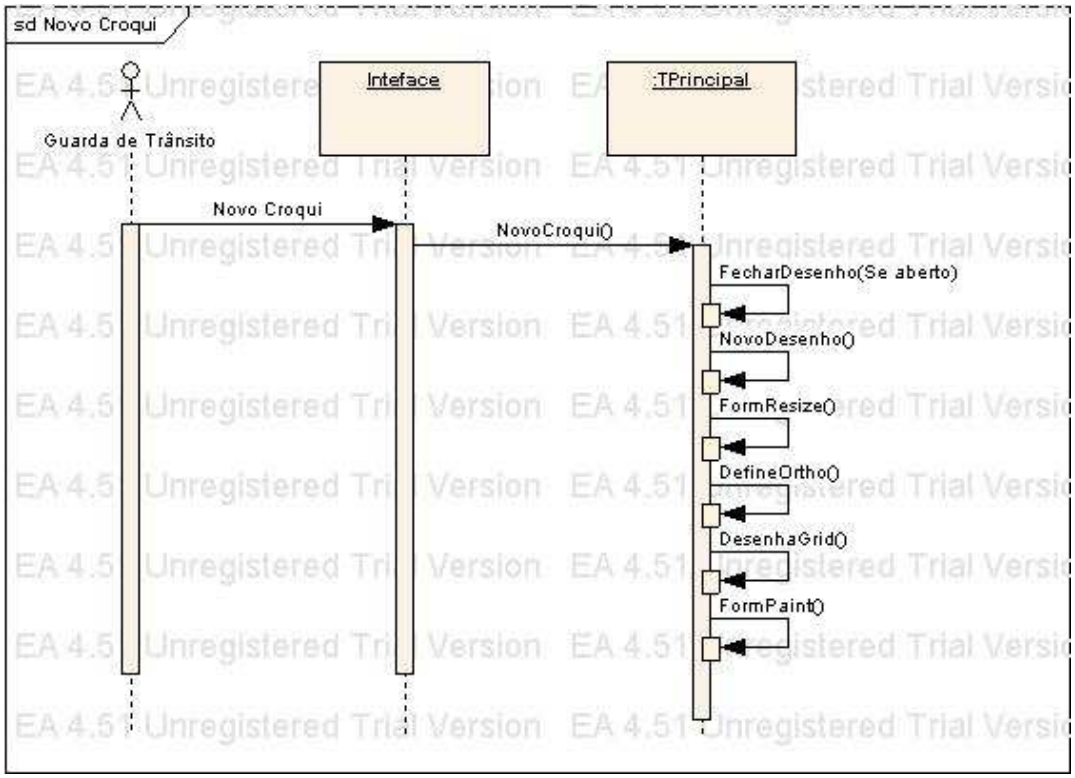


Figura 5: Diagrama de seqüência - Novo croqui.

A figura 6 apresenta a seqüência de mensagens necessárias para que o sistema crie uma nova ocorrência. Na criação de uma ocorrência o sistema também verifica se há um desenho aberto, se existe ele o fecha. Em seguida o sistema apresenta uma tela onde o usuário deve escolher o croqui que deseja abrir. Depois de selecionado o croqui o sistema o abre em um novo desenho, configura o *OpenGL*, cria a grade e mostra o desenho na tela. O último método apresenta um tela onde o usuário deve digitar as informações referentes ao acidente

que serão inseridas pelo sistema no banco de dados.

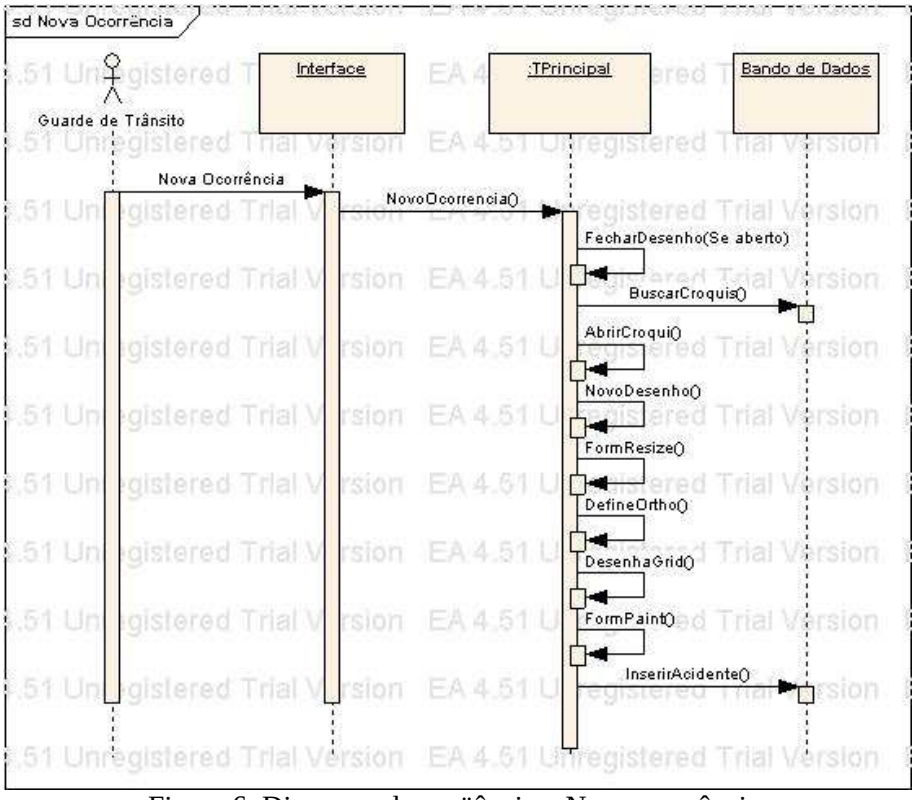


Figura 6: Diagrama de seqüência – Nova ocorrência.

A figura 7 apresenta a seqüência de mensagens necessárias para que o sistema possibilite a escolha de um croqui previamente salvo e posteriormente a abertura do arquivo onde o mesmo esteja armazenado. Ao abrir um croqui o sistema verifica se existe algum desenho aberto, caso haja o sistema fecha o desenho. Em seguida o sistema apresenta um tela onde o usuário seleciona o croqui desejado. Após selecionado o croqui, o sistema o abre, configura o *OpenGL,* cria a grade e mostra o desenho na tela.

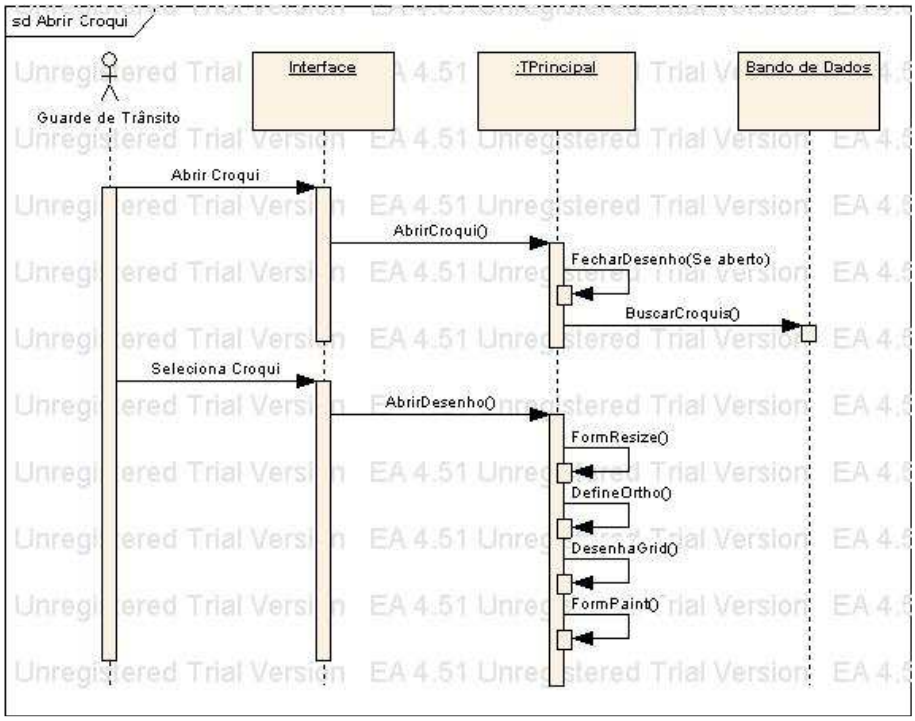


Figura 7: Diagrama de seqüência – Abrir croqui.

A figura 8 apresenta a seqüência de mensagens necessárias para que o sistema possibilite a escolha de uma ocorrência previamente salva e posteriormente a abertura do arquivo onde a mesma esteja armazenada. Ao abrir um ocorrência o sistema também verifica se há um desenho aberto, se existe ele o fecha. Em seguida o sistema apresenta um tela onde o usuário digita as informações referentes ao acidente, com esta informações o sistema busca as ocorrências e apresenta ao usuário. Após selecionada uma ocorrência o sistema a abre, configura o *OpenGL*, cria a grade e mostra o desenho na tela.

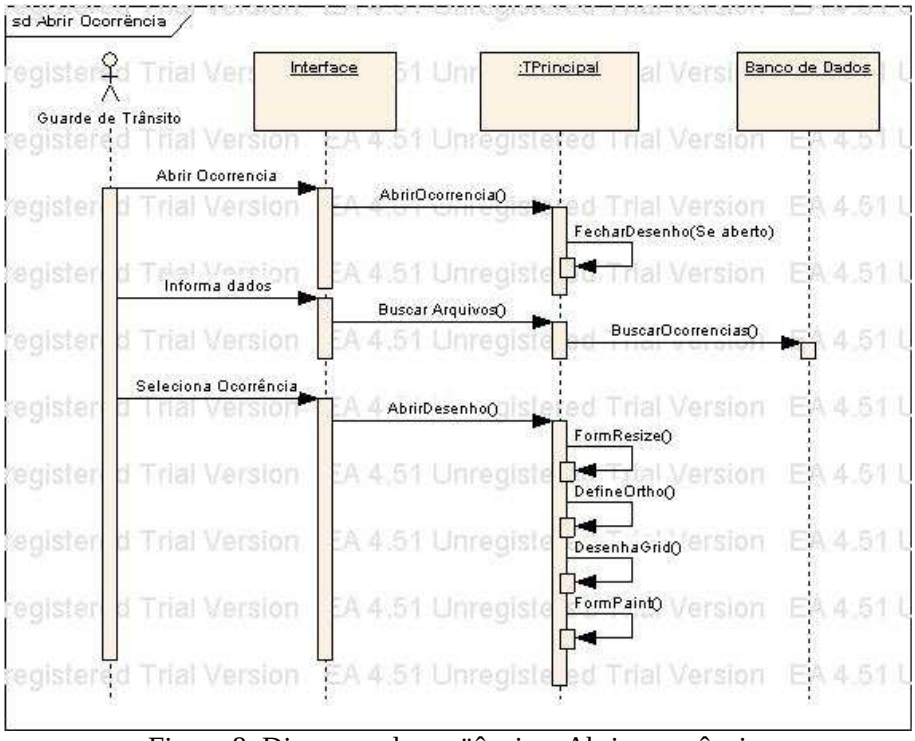


Figura 8: Diagrama de seqüência – Abrir ocorrência.

A figura 9 apresenta a seqüência de mensagens necessárias para que o sistema armazene o desenho criado em um arquivo. Este procedimento é utilizado para os croquis e paras as ocorrências. Ao salvar um desenho o sistema chama o método *SalvarDesenho* que é responsável por gerar e armazenar o arquivo do desenho e em seguida salva as informações cadastradas no banco de dados.

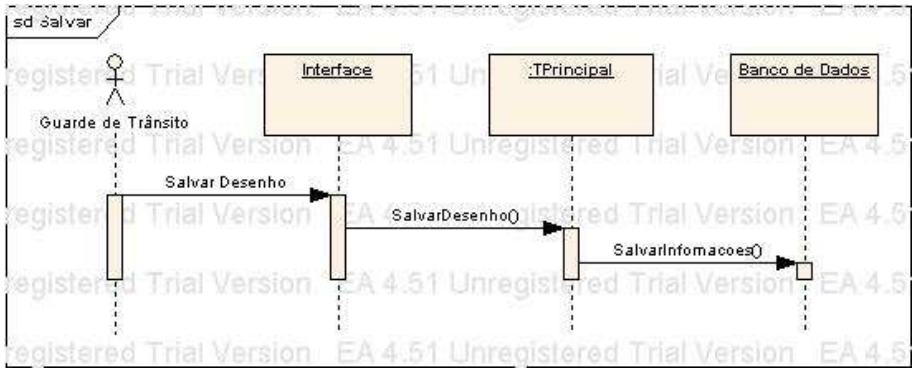


Figura 9: Diagrama de seqüência – Salvar.

A figura 10 simula o usuário executando a criação de três objetos gráficos sendo eles um veículo, um pedestre e uma placa de trânsito. Conseqüentemente é apresentada a seqüência de mensagens necessárias para criação de cada objeto. Quando o usuário cria um destes três objetos o sistema chama o método responsável pela criação do objeto em sua referida classe e logo após define o objeto como selecionado. Em seguida o sistema repinta todo o desenho na tela novamente chamando a função de desenho do objeto que foi criado e também o método responsável por criar e mostrar os pontos de seleção ao redor do objeto.

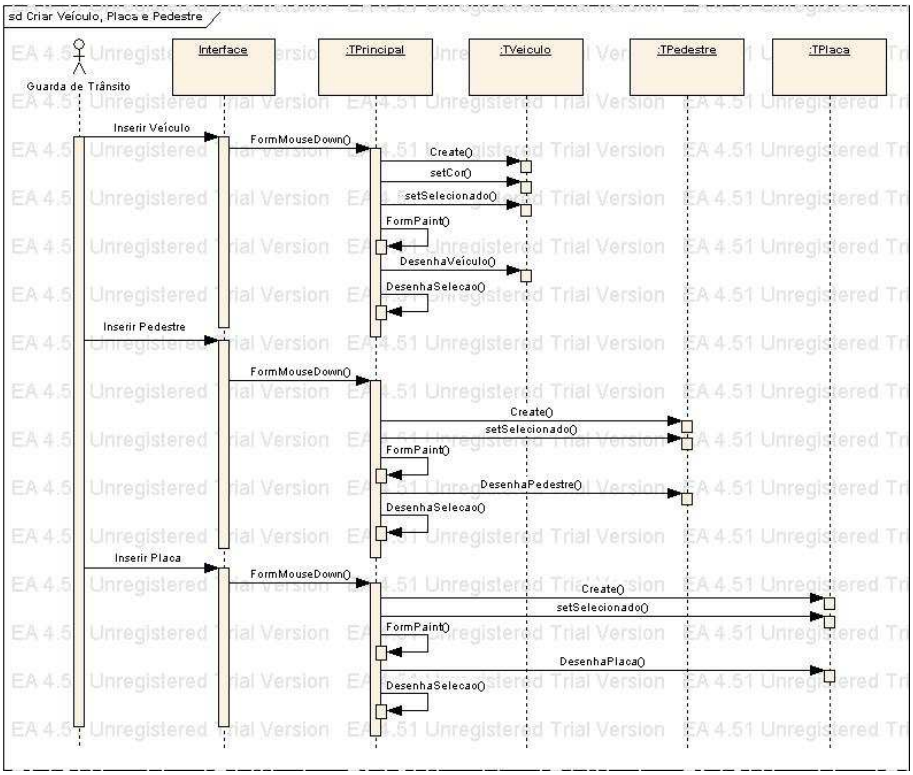


Figura 10: Diagrama de seqüência – Criação de um veículo, de um pedestre e uma placa de trânsito.

A figura 11 simula o usuário executando a criação de três objetos gráficos sendo eles uma linha, uma curva e uma régua. Conseqüentemente é apresentada a seqüência de mensagens necessárias para criação de cada objeto. Deve-se levar em consideração que uma curva é criada a partir de um reta, ou seja, curva-se uma reta previamente criada. Quando o usuário está criando um objeto do tipo linha cada clique do mouse no desenho corresponde a um vértice. No momento do clique o sistema chama o método responsável pela criação dos vértices e armazena os mesmo na lista de vértices do objeto linha. Em seguida o sistema define todas as propriedades do objeto, repinta o desenho chamando o método de desenho do objeto linha e apresenta os pontos de seleção ao redor do objeto. Para criar uma curva o usuário deve clicar duas vezes sobre um dos vértices do objeto linha. Dessa forma o sistema define o objeto como uma curva, repinta novamente o desenho na tela e mostra os pontos de seleção junto com os pontos de controle que são usados para curvar a linha. A criação do objeto régua segue o mesmas seqüência do objeto linha, porém no objeto régua é possível criar somente dois vértices.

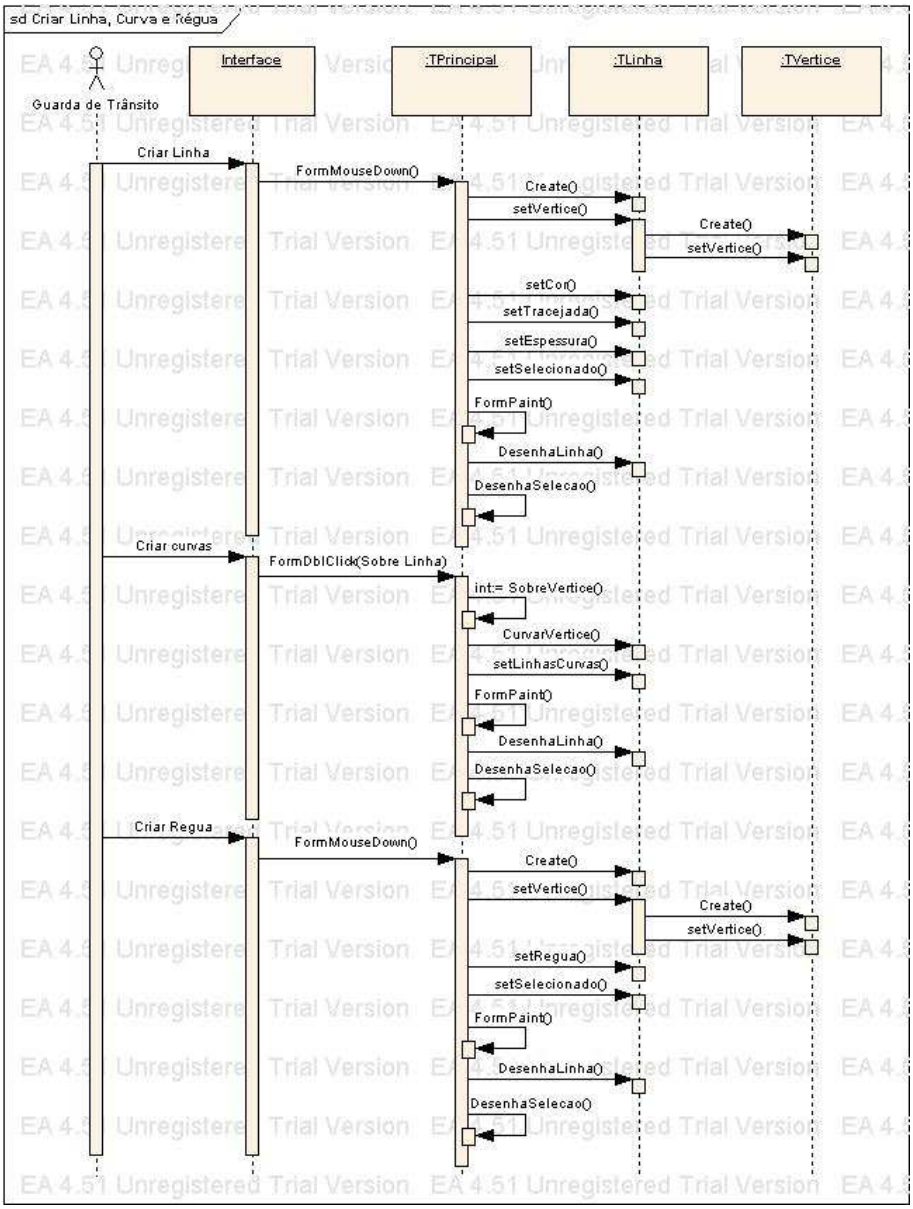


Figura 11: Diagrama de seqüência – Criação de uma linha, uma curva e um régua.

A figura 12 simula o usuário executando a criação de três objetos gráficos sendo eles um retângulo, um círculo e um texto. Conseqüentemente é apresentada a seqüência de mensagens necessárias para criação de cada objeto. Ao criar um retângulo o sistema define o primeiro vértice no local do clique do mouse e o segundo no local onde o usuário solta o botão. Dessa forma ao clicar o botão do mouse o sistema cria o objeto, define a cor e define o primeiro vértice. Quando o usuário arrasta o mouse o sistema desenha o retângulo considerando como segundo vértice a posição do ponteiro do mouse. Ao soltar o botão ele define o local do segundo vértice, repinta o desenho chamando o método de desenho do objeto criado e apresenta os pontos de seleção ao redor do objeto. O objeto círculo segue a mesma seqüência, porém ao pressionar o botão o sistema define o centro do círculo e ao soltar o sistema define o raio da circunferência. Ao criar um objeto do tipo texto o sistema apresenta

um tela onde o usuário deve digitar o texto. Em seguida o sistema cria o objeto, define suas propriedades, repinta o desenho na tela e apresenta os pontos de seleção ao redor do objeto.

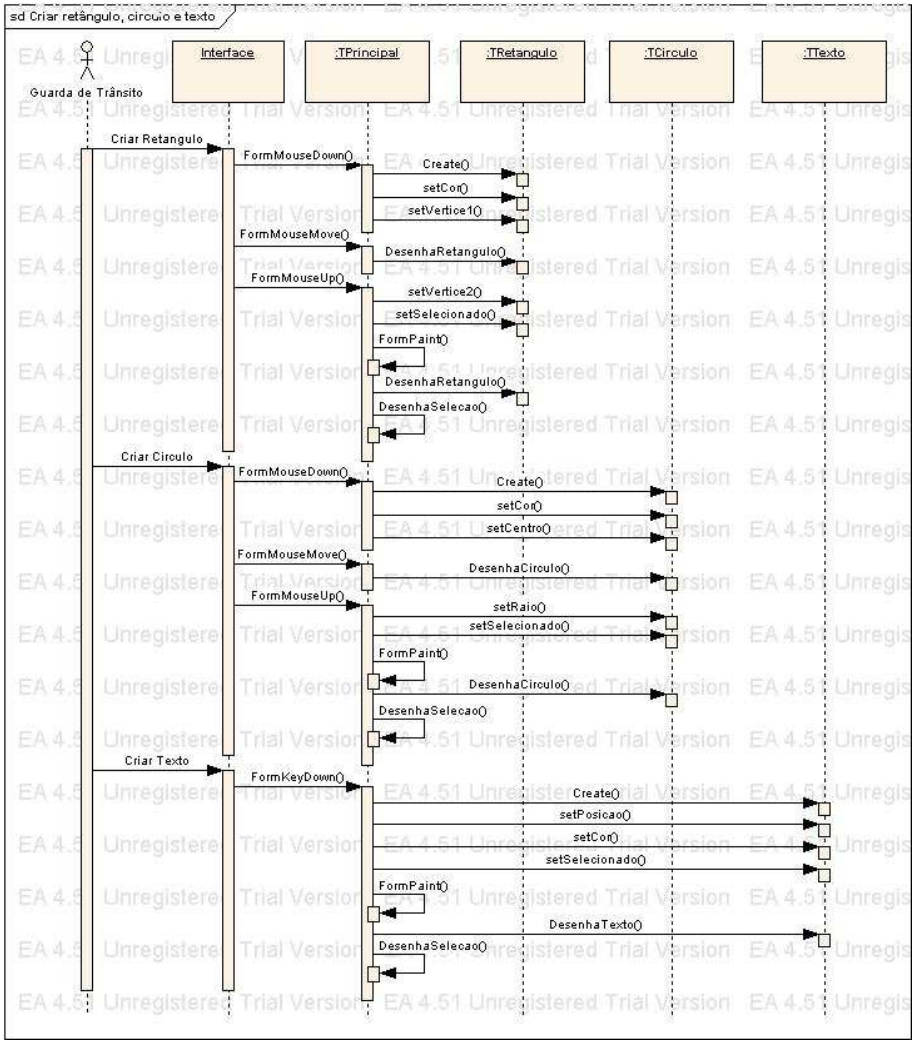


Figura 12: Diagrama de seqüência – Criação de um retângulo, um círculo e um texto.

A figura 13 apresenta a seqüência de mensagens necessárias para que se crie o objeto legenda. A criação deste objeto só é possível com a vinculação do mesmo a um retângulo, um círculo ou uma linha, desta forma é necessário que um objeto deste tipo esteja selecionado para que o vínculo possa ser criado e consequentemente o objeto. Ao se criar um objeto legenda o sistema apresenta um tela onde o usuário deve digitar a descrição desejada. Em seguida o sistema cria o objeto, vincula ele ao objeto selecionado, repinta o objeto na tela chamando o método de desenho correspondente, mostra a legenda na barra de status e apresenta os pontos de seleção em torno do objeto.

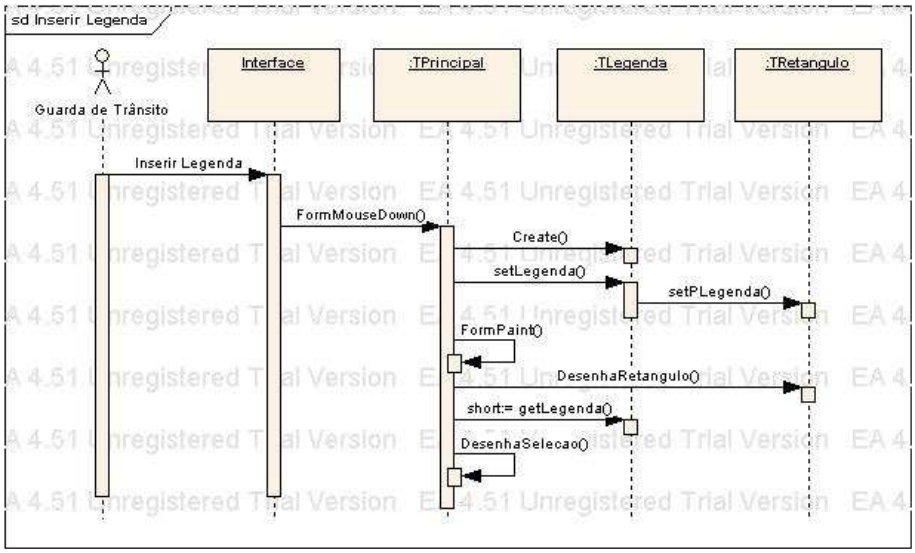


Figura 13: Diagrama de seqüência – Criação de uma legenda.

A figura 14 simula um usuário movendo, redimensionando e rotacionando o objeto retângulo. Conseqüentemente é apresentada a seqüência de mensagens necessárias para cada procedimento. Deve-se levar em consideração que para o inicio desta seqüência o objeto deve estar previamente selecionado. Estes procedimentos se aplicam a todos os outro objetos do sistema e seguem a mesma seqüência de mensagens. Porém os objetos do tipo placa de trânsito, pedestre, veículo e texto não podem ser redimensionados pois possuem padrões de tamanho. Para movimentar um objeto, o usuário deve clicar sobre ele e arrastar. Ao fazer isto o sistema chama o método *MudarPosição* do objeto selecionado enviando a posição de inicio do mouse e a posição final. Este método é responsável por definir a nova posição dos vértices do objeto. Para redimensionar um objeto o usuário deve seleciona-lo e clicar sobre um ponto de seleção e arrastar até o tamanho desejado. Nesse processo o sistema chama o método *MudarTamanho* do objeto selecionado que redefine a posição dos vértices. Para rotacionar um objeto o usuário deve selecionar o objeto, clicar e arrastar o mouse rotacionando o objeto até a posição desejada. Para isto o sistema chama o método *Rotacionar* do objeto selecionado que é responsável por calcular a nova posição dos vértices do objeto. Ao término de um destes três métodos o sistema repinta o desenho na tela e desenha os pontos de seleção em torno dos mesmos.

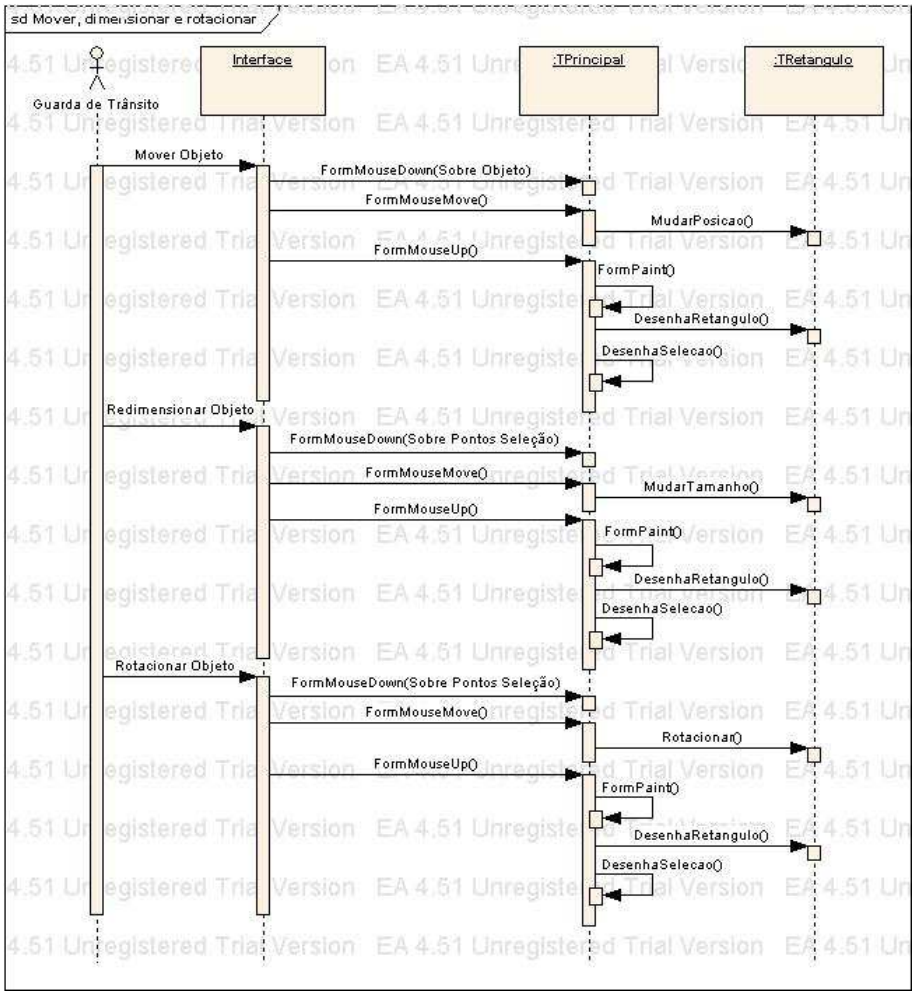


Figura 14: Diagrama de seqüência – Movimentar, redimensionar e rotacionar objetos.

A figura 15 apresenta a seqüência de mensagens necessárias para aumentar a área de trabalho do desenho, ferramenta zoom mais. As ferramentas zoom menos e zoom para desenho seguem a mesma seqüência. Quando o usuário utiliza esta ferramenta o sistema redefine as configurações do OpenGL e repinta o desenho na tela.

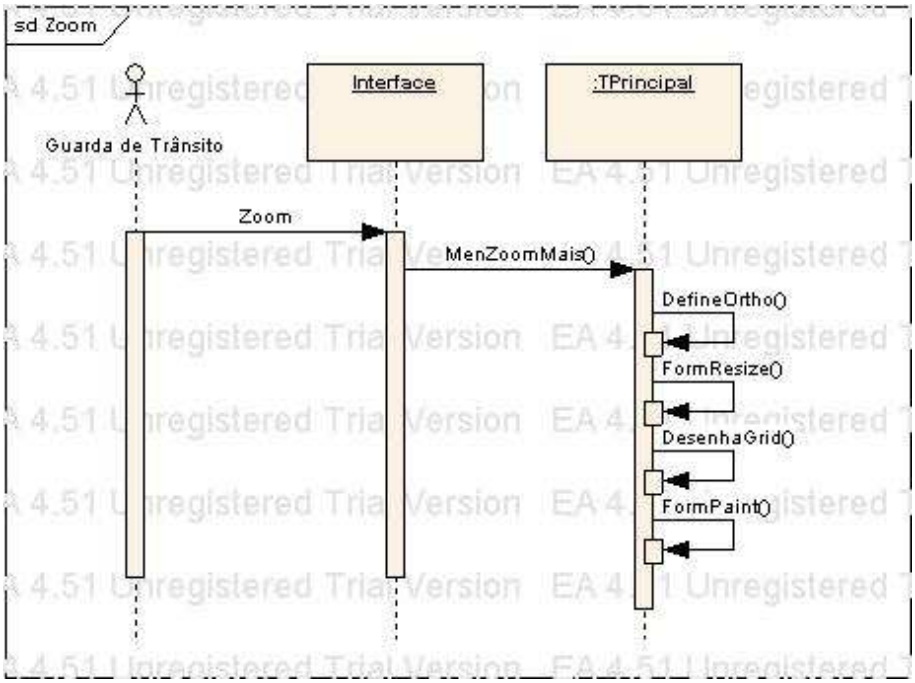


Figura 15: Diagrama de seqüência – Zoom.

A figura 16 simula um usuário realizando o cadastro de pedestre, veículo e declaração. Consequentemente é apresentada a seqüência de mensagens necessárias para cada procedimento. Para cadastrar as informações do veículo o usuário clica duas vezes sobre o mesmo, o sistema apresenta um tela onde os dados devem ser digitados. No momento em que o usuário salvar as informações o sistema as insere no banco de dados. O mesmo vale para o cadastro das declarações dos envolvidos. No cadastro de pedestres o usuário deve clicar duas +vezes sobre o mesmo. Em seguida o sistema apresenta um tela com todos os envolvidos no acidente, onde o usuário deve selecionar a pessoa a qual o desenho se refere.

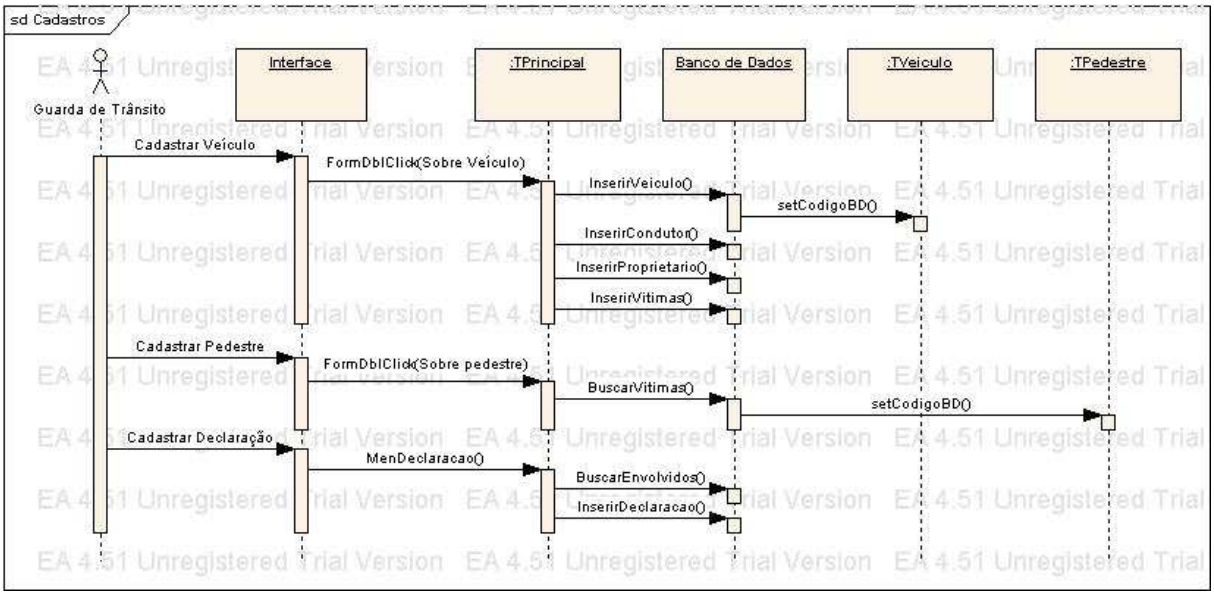


Figura 16: Diagrama de seqüência – Cadastro de pedestre, veículo e declaração.

3.4.4 Modelo de entidades e relacionamentos

O modelo de entidades e relacionamentos, segundo Cougo (1997), descreve o mundo como: “...cheio de coisas que possuem características próprias e que se relacionam entre si”. O MER é um modelo abstrato cuja finalidade é descrever os dados a serem utilizados em um sistema de informações. A figura 17 mostra o MER desenvolvido para atender as necessidades do sistema.

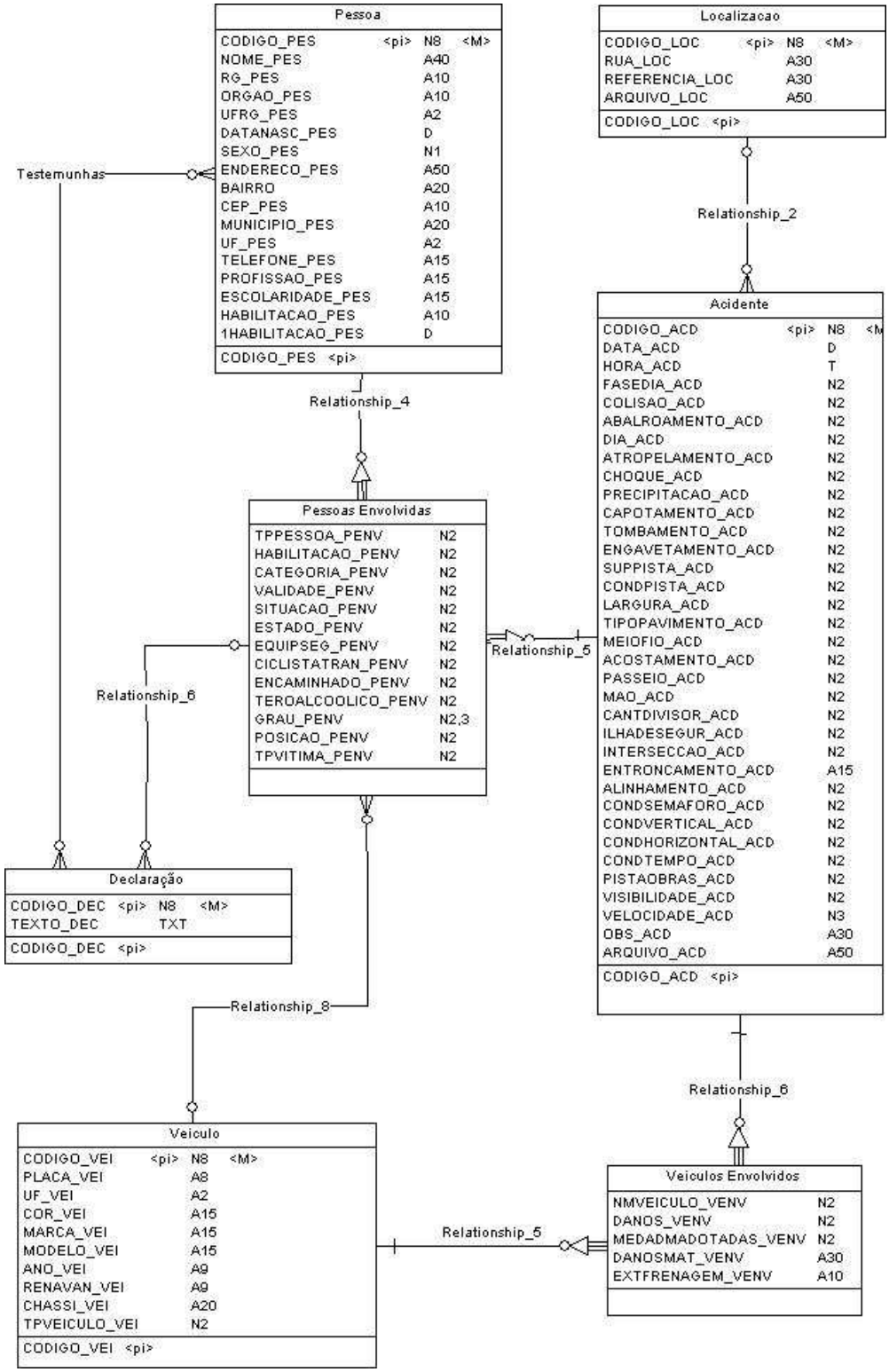


Figura 17: MER utilizado no sistema.

3.5 IMPLEMENTAÇÃO

Neste item são apresentadas as técnicas e ferramentas utilizadas, bem como a explicação das principais rotinas utilizadas para a criação da aplicação.

3.5.1 Técnicas e ferramentas utilizadas

A ferramenta foi implementada em *Delphi 6*, utilizando-se os conceitos de OO para a criação da estrutura de dados das primitivas gráficas. Para exibir os objetos contidos na estrutura de dados o sistema utiliza os comandos da biblioteca gráfica *OpenGL*.

Para armazenar as informações, foi utilizado o banco de dados *MySQL*. Com o auxílio da ferramenta *Sybase PowerDesigner*, foi desenvolvido um MER específico para este banco. Esta mesma ferramenta foi utilizada para gerar um *script* contendo os comandos *SQL* necessários para a criação das entidades e dos relacionamentos referentes ao modelo criado. Utilizando-se o *MySQL-Front* foi possível executar o *script*, e a partir dos comandos contidos nele a ferramenta gerou a base de dados.

3.5.2 Conexão do *OpenGL* com o *Delphi*

Cohen e Manssour (2006) afirmam que, por ser portável, o *OpenGL* não possui funções para gerenciamento de janelas, tratamento de eventos ou manipulação de arquivos. Neste caso, são utilizadas as funções específicas de cada plataforma para tal propósito, como a própria API do *Windows*.

Para utilizar o *OpenGL* em um formulário do *Delphi* é necessária a criação de um *Device Context* (DC), que é uma estrutura utilizada pelo *Windows* que representa a saída para o sistema operacional que neste caso é usado para a interface gráfica da janela (*window*). Após a criação do DC é necessário informar ao *Delphi* que ele é a saída para o formulário do ambiente gráfico de programação. As imagens formadas no DC são compostas por vários *pixels* que devem ser pré-definidos. Para isso é criada uma descrição do formato dos *pixels*. Para armazenar as configurações e comando do *OpenGL* deve-se criar um *Rendering Context* (RC). O quadro 3 apresenta a implementação desta funções no sistema.

```
procedure TFPrincipal.NovoDesenho;
var
    pfd : TPixelFormatDescriptor;
    FormatIndex : integer;
begin
    fillchar(pfd,Sizeof(pfd),0);
    with pfd do
    begin
        nSize      := SizeOf(pfd);              //Tamanho da estrutura
        nVersion   := 1;                        //Versao da estrutura
        dwFlags    := PFD_DRAW_TO_WINDOW        //Desenha na window
                      or PFD_SUPPORT_OPENGL;    //Suporta Opengl na window
        iPixelType := PFD_TYPE_RGBA;            //Modo de cor RGB
        CColorBits := 24;                       //Cores de 24 bits
        cDepthBits := 32;                       //Tamanho do buffer de
                                                  profundidade
        iLayerType := PFD_MAIN_PLANE;           //Tipo de Layer
    end;
    glDC := getDC(handle);                      //Cria o DC
    FormatIndex := ChoosePixelFormat(glDC,@pfd); //Escohe o melhor formato do
     pixel para o DC informado
    SetPixelFormat(glDC,FormatIndex,@pfd);      //Define o formato do pixel
    GLContext := wglCreateContext(glDC);        //Cria o RC
    wglMakeCurrent(glDC,GLContext);             //Define o RC como corrente
    ...
end;
```

Quadro 3 – Implementação do OpenGL com o Delphi.

A partir deste código criasse o universo do *OpenGL* que por definição é infinito, desta forma e necessário definir qual porção deste mundo será mapeada para a tela. Esta área recebe o nome de *window*, e é definida através do comando *gluOrtho2d*. Da mesma forma é necessário definir em que parte do monitor esta *window* será exibida. Esta região é denominada *viewport* e é definida pelo comando *glViewPort.* O quadro 4 mostra o método que implementa estes comandos.

```
procedure TFPrincipal.DefineOrtho;
begin
    {procedimento que define o tamanho do ortho(gluOrtho2D) usando os valores
     definidos na nas variaveis globais OEsquerda, ODireita, OBase, OTopo. Define
     também o tamanho da janela ViewPort(glViewPort) para o tamanho total do
     formulário}
    glViewport(0,20,ClientWidth,ClientHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(OEsquerda,ODireita,OBase,OTopo);
end;
```

Quadro 4 – Implementação da *window* e do *viewport.*

3.5.3 Estrutura de dados

Conforme descrito no item 2.2, computação gráfica consiste em transformar dados em imagens. Por isso, o primeiro e mais importante passo no desenvolvimento de um ferramenta gráfica é criar uma estrutura de dados capaz de armazenar as informações necessárias para a

construção do desenho.

Neste trabalho foram utilizados os conceitos de OO para criar a estrutura de dados. Cada tipo de elemento gráfico do sistema originou uma classe com seus respectivos atributos e métodos. Qualquer objeto inserido no sistema é uma instância de uma dessas classes criadas, e seus atributos armazenam todos os dados necessários para que o mesmo possa ser criado e exibido.

Por serem elementos gráficos diferentes, cada um possui uma forma diferente de armazenamento dos dados referentes aos vértices, porém alguns atributos como cor e seleção são comuns a todos. Este atributos são na verdade as entradas necessárias para os comandos do *OpenGL*. Cada classe possui um conjunto de métodos que além de criar as instâncias são responsáveis por acessar e alterar os atributos de acordo com necessidade. Toda classe possui um método de desenho, neste, estão implementados todos os comandos do *OpenGL* necessários para transformar a estrutura de dados em uma imagem na tela. Outros métodos também são comuns a quase todas as classes, como por exemplo mudar posição, mudar tamanho e rotacionar. Contudo estes métodos não contém comandos *OpenGL*, mas sim, fórmulas matemáticas necessárias para a alterar a estrutura de dados de acordo com a necessidade. Ou seja, com exceção do método de desenho todos os outros métodos utilizam fórmulas matemáticas e/ou comandos do *OpenGL* somente para criar ou modificar a estrutura de dados.

Um desenho pode conter dezenas de objetos instanciados, para armazenar este objetos o sistema utiliza uma classe do *Delphi* denominada *TList*. De acordo com Cantu (2002), a *TList* define um lista de ponteiros, que pode ser usada para armazenar objetos de qualquer classe. Desta forma todos os objetos gráficos criados pelo sistema podem ser armazenados em uma única fila facilitando o processo de busca dos objetos.

3.5.4 Reconhecimento das ferramentas

O sistema desenvolvido possui ferramentas para a criação de objetos bem como ferramentas para transformação do mesmos. Todas as ferramentas podem ser encontradas nos menus, mas cada uma possui um atalho específico na área de trabalho. O funcionamento de cada ferramenta está diretamente relacionado aos eventos de mouse. Estes eventos são *onMouseDown* que ocorre quando o botão é pressionado, *onMouseMove* que ocorre quando se movimenta o mouse e *onMouseUp* que ocorre quando o botão deixa de ser pressionado.

Algumas ferramenta precisam utilizar os três eventos para que o objetivo da mesma seja alcançado. Por causa desta necessidade não foi possível a implementação das funções das ferramentas diretamente nos seus menus de seleção. A solução encontrada foi utilizar uma variável global que armazena a ferramenta selecionada. Essa variável guarda um valor numérico definido quando o usuário seleciona a ferramenta desejada. Mais tarde essa variável é filtrada nos três eventos de mouse e dessa maneira o sistema consegue identificar qual é a ferramenta selecionada e quais as funções que deve executar. Há ainda ferramentas que necessitam utilizar os eventos de *onMouseDown* e *onMouseMove* simultaneamente, porém isto não é possível no ambiente *Delphi*. Para isto foi utilizada uma variável para armazenar o estado do botão do mouse, ou seja, ao pressionar o botão do mouse essa variável é ativada e ao soltar o botão é desativada. Dessa maneira é possível fazer um teste no movimento do mouse para saber se o botão esta pressionado no momento em que ele está sendo movimentado. O quadro 5 apresenta resumidamente um algoritmo exemplificando essas soluções. O algoritmo simula a criação de um retângulo que exige todas as funções citadas acima, deve-se levar em consideração que o usuário clicou no menu criar retângulo antes deste código ser executado.

```
variáveis
       ferramenta  //define qual ferramenta esta selecionada
       arrastando //define se o botão esta pressionado
procedimento onMouseDown
inicio
       caso ferramenta faça
       1 : inicio //ferramenta de seleção
              excetuar funções de seleção
           fim
       2 : inicio //ferramenta desenha retângulo
              criar objeto retângulo
              define cor do objeto
              define o primeiro vértice
              arrastando := verdadeiro
            fim
       fim
fim

procedimento onMouseMove
inicio
       se arrastando = verdadeiro então
       inicio
              caso ferramenta faça
              2 : inicio
                     define o segundo vértice
                     desenha retângulo na tela
                  fim
              fim
       fim
fim
procedimento onMouseUp
inicio
       se arrastando = verdadeiro então
       inicio
              arrastando := falso
              caso ferramenta faça
              2 : inicio
                     salva objeto na lista de objetos do desenho
                     repinta todo o desenho na tela
                  fim
              fim
       fim
fim
```

Quadro 5 – Algoritmo para seleção das ferramentas.

3.5.5 Criação dos objetos

Nesta aplicação a criação de objetos gráficos é feita interativamente, ou seja, o usuário através do mouse desenha os objetos na área de trabalho do sistema. Desta maneira a estrutura de dados de cada objeto e construída a partir dos dados coletados do mouse. Porém as coordenadas capturadas pelo mouse estão em *pixels* e o sistema trabalha com a escala métrica. Outro problema é que o *Windows* define as coordenadas cartesianas 0,0 no topo esquerdo da tela, enquanto que o *OpenGL* define na base esquerda. Para resolver estes problemas foi criado um método que utiliza comando *OpenGL* chamado *gluUnProject.* Este método recebe

as coordenadas da tela em *pixels* e retorna as coordenadas do mundo *OpenGL* em variáveis públicas que podem ser lidas por qualquer outro método do sistema. Outra função deste método inverter o eixo Y do *Windows* do topo para a base da tela. O quadro 6 mostra a implementação deste método.

```
procedure TFPrincipal.cordmundo(winX,WinY: double);
var
    Yreal : GLint;
    viewport : array [0..4] of GLint;
    mvmatrix, projmatrix : array [0..16] of double;
begin
    glGetIntegerv(GL_VIEWPORT,@viewport); //salva no array a matriz que define o
                                            tamanho da area em pixels utilizada
                                            pelo OpenGL
    glGetDoublev(GL_MODELVIEW_MATRIX,@mvmatrix); //salva no array a matriz do
                                                   modelo
    glGetDoublev(GL_PROJECTION_MATRIX,@projmatrix); //salva no array a matriz de
                                                      projeção
    Yreal := viewport[3] - strtoint(floattostr(WinY)) - 1; //inverão de Y do topo
                                                             da tela para a base
    gluUnProject(winX,Yreal,0.0,@mvmatrix,@projmatrix,
    @viewport,mundox,mundoy,mundoz); //transforma as coordenadas
                                       mundoX, mundoY, mundoZ variáveis públicas
end;
```

Quadro 6 – Implementação da conversão de coordenadas de tela para *OpenGL*.

Os quadros 7 e 8 mostram o algoritmo utilizado para coletar os dados da interface e criar os objetos gráficos do sistema. A coleta de dados para a construção da estrutura de dados é diferente para cada tipo de objeto, por isso os próximos itens desta seção explicam com mais detalhes as rotinas deste algoritmo.

```
variáveis
       ferramenta  //define qual ferramenta esta selecionada
       arrastando //define se o botão esta pressionado
       mundoX     //variáveis públicas que contém a coordenada x do mouse
       mundoY     //variáveis públicas que contém a coordenada y do mouse
procedimento onMouseDown
inicio
       se ferramenta >= 1 e ferramenta <= 20 então
       inicio
              caso ferramenta faça
              3 : inicio //ferramenta desenha pedestre
                     criar objeto pedestre(mundoX,mundoY)
                     adicionar objeto a lista de objetos
                     repinta todo o desenho na tela - fim
              6 : inicio //ferramenta desenha linha
                     se desenhando linha entao
                     inicio
                            adicionar vertice(mundoX, mundoY)
                            define espessura do objeto
                            define tracejado do objeto – fim senao
                     inicio
                            criar objeto linha
                            define cor do objeto
                            adicionar vertice(mundoX,mundoY)
                            adicionar objeto a lista de objetos fim
                     repinta todo o desenho na tela - fim
              7 : inicio //ferramenta desenha retângulo
                     criar objeto retângulo
                     define cor do objeto
                     define o primeiro vértice(mundoX,mundoY)
                     arrastando := verdadeiro - fim
              8 : inicio //ferramenta desenha circulo
                     criar objeto circulo
                     define cor do objeto
                     define centro do objeto(mundoX,mundoY)
                     arrastando := verdadeiro - fim
              10 : inicio //ferramenta de texto
                     criar objeto texto
                     definir cor do objeto
                     texto := resultado(abrir caixa de dialogo)
                     definir posicao e texto (mundoX,mundoY,texto)
                     adicionar objeto a lista de objetos
                     repinta todo o desenho na tela - fim
              15 : inicio //ferramena de edição de pontos
                     define o vértice selecionado
                     arrasntando := verdadeiro – fim – fim
       se ferramenta >= 20 e ferramenta <= 28 então
       inicio
              caso ferramenta faça
              20 : criar objeto veiculo(mundoX,mundoY,Tipo20);
              21 : criar objeto veiculo(mundoX,mundoY,Tipo21);
              ... - fim
              define cor do objeto
              adicionar objeto a lista de objetos
              repinta todo o desenho na tela - fim
       se ferramenta >= 100 e ferramenta <= 159 então
       inicio
              caso ferramenta faça
              140 : criar objeto placa(mundoX,mundoY,Tipo140);
              141 : criar objeto placa(mundoX,mundoY,Tipo141);
              ... - fim
              adicionar objeto a lista de objetos
              repinta todo o desenho na tela - fim - fim
```

Quadro 7 - Algoritmo para criação dos objetos.

```
procedimento onMouseMove
inicio
       se arrastando = verdadeiro então
       inicio
              caso ferramenta faça
              7 : inicio //ferramenta desenha retângulo
                     define o segundo vértice(mundoX,mundoY)
                     desenha retângulo na tela
                  fim
              8 : inicio //ferramenta desenha cícrulo
                     define raio(mundoX,mundoY)
                     desenha círculo na tela
                  fim
              15 : inicio // ferramenta de edição de pontos
                     se objeto linha = selecionado entao
                         mover vertice do objeto linha(mundoX,mundoY,
                                                       ponto selecionado)
                  fim
              fim
       fim
fim

procedimento onMouseUp
inicio
       se arrastando = verdadeiro então
       inicio
              arrastando := falso
              caso ferramenta faça
              7 : inicio //ferramenta desenha retângulo
                     adiciona objeto retângulo na lista de objetos
                     repinta todo o desenho na tela
                  fim
              8 : inicio //ferramenta desenha círculo
                     adiciona objeto círculo na lista de objetos
                     repinta todo o desenho na tela
                  fim
              fim
       fim
fim
```

Quadro 8– Algoritmo para criação dos objetos.

3.5.5.1 Criação de retângulos

No contexto deste trabalho geometricamente, um retângulo é representado por quatro vértices, A, B, C e D unidos por segmentos de reta de A para B, de B para C, de C para D e de D para A. Para criar estes vértices o usuário deve selecionar a ferramenta, pressionar o botão do mouse, arrastar o mouse desenhando o objeto da maneira deseja e solta o botão. Ao pressionar o botão, o sistema define o ponto do clique como o primeiro vértice. Ao arrastar, o ponteiro do mouse se torna o terceiro vértice enquanto que o segundo é calculado utilizando-se a coordenada x do primeiro e y do terceiro, e, da mesma maneira o quarto vértice utiliza as coordenadas x do terceiro e y do primeiro. E ao soltar o botão, o objeto é inserido na lista de objetos do sistema. Ao criar o objeto o sistema define sua cor utilizando a cor selecionada na

paleta de cores. Os quadros 7 e 8 mostram o algoritmo utilizado para a criação deste objeto.

Para desenhar um retângulo no *OpenGL* utiliza-se o comando *GL_LINE_LOOP*. Fornecendo-se os quatro vértices este comando se ocupa de traçar os segmentos de reta entre eles e mostrá-los na tela. O quadro 9 mostra a implementação método de desenho da classe *TRetangulo*.

```
procedure TRetangulo.DesenhaRetangulo;
begin
    glColor3f(Red,Green,Blue); //Define a cor
    glBegin(GL_LINE_LOOP);     //Desenha o retãngulo
        glVertex2d(x1,y1);     //1º Vértice
        glVertex2d(x2,y2);     //2º Vértice
        glVertex2d(x3,y3);     //3º Vértice
        glVertex2d(x4,y4);     //4º Vértice
    glEnd;
end;
```

Quadro 9 – Método desenhar retângulo.

A figura 18 apresenta a tela do sistema com o desenho de três objetos retângulo.

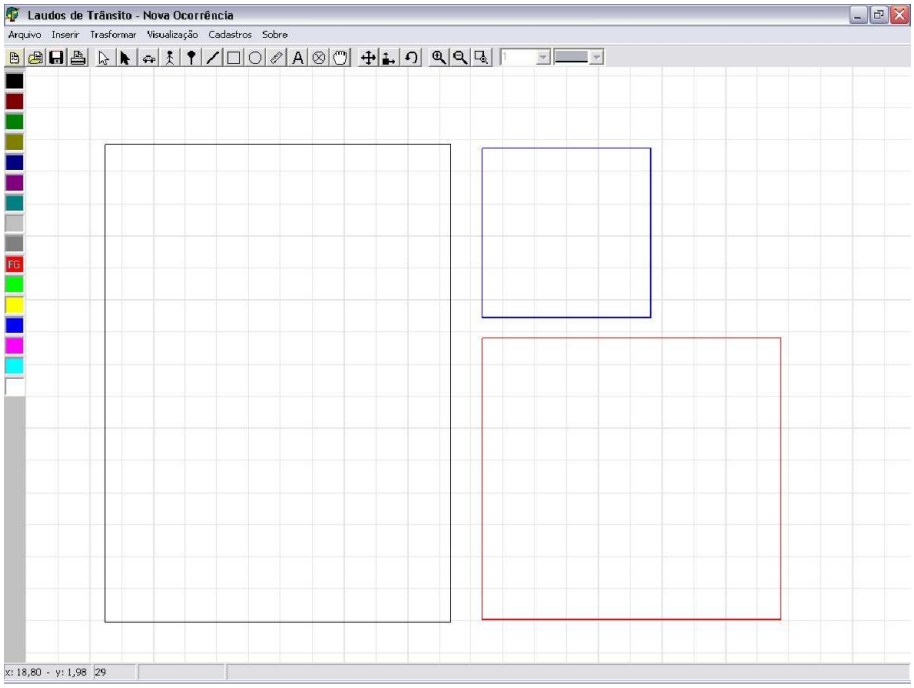


Figura 18 – Desenho de retângulos no sistema

3.5.5.2 Criação de círculos

Na geometria um círculo é definido por um centro e a medida de seu raio. Rotacionando o raio ao redor do ponto central podemos definir vários vértices e a união destes vértices formam um círculo, este processo é melhor visualizado na figura 19.

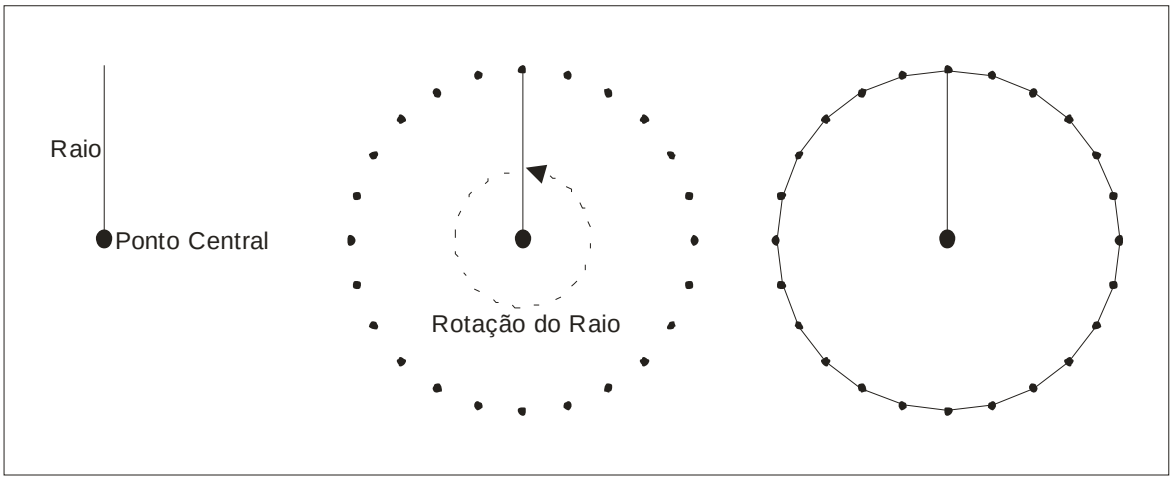


Figura 19 – Construção geométrica de um círculo.

Para criar o círculo o usuário pressiona o botão do mouse, arrasta desenhando o objeto da maneira desejada e solta o botão. Ao pressionar o botão, o sistema define o ponto do clique como ponto central. Ao arrastar, o sistema mede a distância entre o ponto central e o ponteiro do mouse definindo assim o raio. E ao soltar o botão, o objeto é inserido na lista de objetos do sistema. Ao criar o objeto o sistema define sua cor utilizando a cor selecionada na paleta de cores. Os quadros 7 e 8 mostram o algoritmo utilizado para a criação deste objeto.

Para se desenhar um círculo no *OpenGL* é necessário primeiro a criação de todos os vértices ao redor do ponto central como explicado anteriormente. Após a criação utiliza-se o comando *GL_LINE_LOOP*, que se ocupa de traçar os segmentos de reta e desenhar a figura na tela. No sistema este procedimentos estão implementados no método de desenho da classe *TCirculo* que é mostrado no quadro 10.

```
procedure TCirculo.DesenhaCirculo;
var
    pontos : GLint;
    angle : GLdouble;
    i : integer;
    x, y : double;
begin
    pontos := 64;              //Quantidade de vértices ao redor do ponto central
    glColor3f(Red,Green,Blue); //Define a cor
    glBegin(GL_LINE_LOOP);     //Desenha o circulo
        for i := 0 to pontos do
        begin
            angle := 2 * 3.1415926535898 * i / pontos; //Calcula o ângulo de
                                                         rotação do raio
            x := xcentro + raio * cos(angle);          //Define a coordenada x
            y := ycentro + raio * sin(angle);          //Define a coordenada y
            glVertex2f(x,y);                           //Cria o novo vértice
        end;
    glEnd;
end;
```

Quadro 10 – Método desenhar círculo.

A figura 20 apresenta a tela do sistema com o desenho de três objetos círculo.

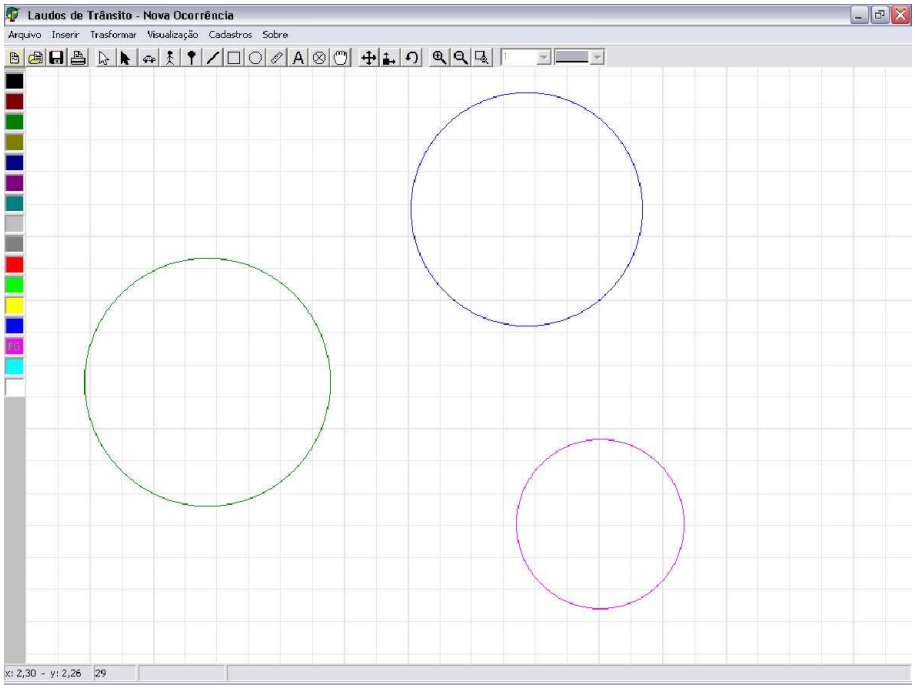


Figura 20 – Desenho de círculos no sistema

3.5.5.3 Criação de Linhas

Na estrutura de dados desenvolvida uma linha é um segmento de reta que une dois vértices. O sistema permite a criação de uma linha com vários vértices, dessa maneira é possível criar as mais variadas figuras geométricas. Para que isso fosse possível foi necessária a criação de uma classe especifica para os vértices, dessa maneira cada vértice criado na linha é uma instâncias dessa classe e essas instancias são todas armazenadas em um atributo do objeto linha que é um lista de objetos *TList*. Ao pressionar o botão do mouse, o usuário define o primeiro vértice e assim por diante com os próximos vértices. Para terminar a linha o usuário pode clicar em cima do primeiro vértice ou selecionar outra ferramenta. Ao criar o objeto o sistema define sua cor utilizando a cor selecionada na paleta de cores. Para este tipo de objeto gráfico o sistema permite que seja alterada a propriedade de espessura da linha e forma de tracejado. O objeto é criado levando em consideração as opções de traçado atualmente selecionadas no sistema, mas podem ser alteradas posteriormente. Este processo pode ser mais bem visualizado nos quadros 7 e 8.

Para desenhar a linha o sistema utiliza o comando *GL_LINE_STRIP*. Fornecendo-se os vértices do objeto este comando se ocupa de traçar os segmentos de reta entre eles e mostrá-

los na tela. O quadro 11 mostra a implementação do método de desenho da classe *TLinha*. Para evitar confusões o trecho de código responsável por desenhar curvas foi ocultado, mas será apresentado no item seguinte.

```
procedure TLinha.DesenhaLinha(TamTelapx, TamTelacm : double);
var
    OVertice : TVertice;
    i, j, fator : integer;
    esp : double;
begin
    glColor3f(Red,Green,Blue); //Define cor do objeto
    if tracejada > 0 then //Seleciona a forma do tracejado da linha
    begin
        fator := round((TamTelapx * tracejada)
                 / (TamTelacm * 100));        //Torna o tracado proporcional ao zoom
        case tracejada of
            1 : glLineStipple(fator, $00ff);
            2 : glLineStipple(fator, $00ff);
            3 : glLineStipple(fator, $00ff);
            4 : glLineStipple(fator, $00ff); //Define a forma do tracejado da linha
        end;
        glEnable(GL_LINE_STIPPLE);           //Ativa o tracejado da linha
    end
    else
        glDisable(GL_LINE_STIPPLE);          //Desativa o tracejado da linha
    esp := (TamTelapx * Espessura)
           / (TamTelacm * 100);              //Torna a espessura proporcional ao
                                               zoom
    glLineWidth(esp);                        //Define a espessura da linha
    if LinhasCurvas then
    begin
    . . .                                    //Comando paras desenhar curvas
                                               mais detalhes no próximo item
    else
    begin
        glBegin(GL_LINE_STRIP);              //Desenha a linha
            for i := 0 to LVertices.Count -1 do  //Percorre lista de vértices
            begin
                OVertice := TObject(LVertices[i]) as TVertice;         //Seleciona
                                                                         vértice
                glVertex2d(OVertice.getXvertice,OVertice.getYvertice); //Cria
                                                                         vértice
            end;
        glEnd;
    end;
    glLineWidth(1);                 //Define o padrão de espessura
    glDisable(GL_LINE_STIPPLE);     //Define o padrao de tracejado
end;
```

Quadro 11 – Método desenhar linha.

A figura 21 apresenta a tela do sistema com o desenho de três objetos linha com diferentes tipos de espessura e tracejados.

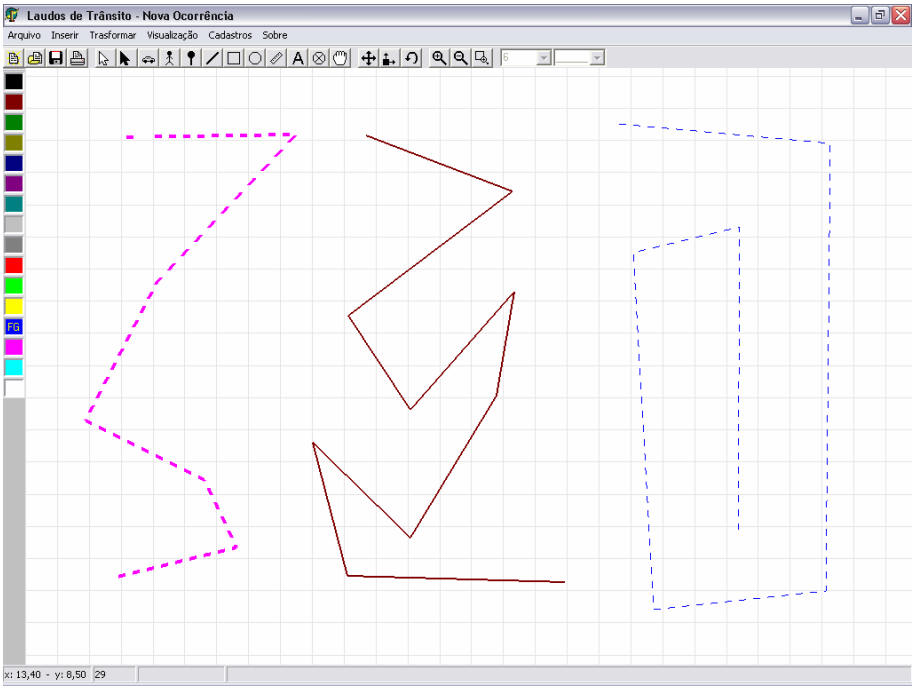


Figura 21 – Desenho de linhas no sistema

3.5.5.4 Criação de Curvas

No presente trabalho a curva utiliza o mesmo conceito do circulo. Para se traçar uma curva é necessário criar vários vértices e traçar segmentos de reta entre eles. No sistema para a criação de uma curva o usuário deve primeiramente criar uma linha e em seguida selecionar a ferramenta de edição de pontos e dar um duplo clique sobre qualquer vértice da linha. Neste momento irão aparecer outros pontos seguidos de uma reta tracejada azul. Ao pressionar o botão do mouse sobre eles e arrastar o usuário pode curvar a linha da maneira desejada. Os quadros 7 e 8 mostram o algoritmo utilizado para a criação de curvas.

Para tornar isto possível o sistema transforma a lista de vértices do objeto em uma matriz. Dentro dessa matriz são armazenados as coordenadas dos vértices, e adicionados os vértices de controle. Estas funções estão implementas no método curvar vértices da classe *TLinha* que é apresentado no quadro 12.

```
procedure TLinha.CurvarVertice;
var
    i : integer;
    auxVertice1, auxVertice2 : Tvertice;
    deltaX, deltaY : double;
begin
    for i := 0  to LVertices.Count - 2 do
    begin
        SetLength(VerticesCurvos,High(VerticesCurvos)+5);   //Define o tamanho do
                                                              array
        auxVertice1 := TObject(LVertices[i]) as TVertice;   //Recebe verice atual
        auxVertice2 := TObject(LVertices[i+1]) as TVertice; //recebe vertice
                                                              seguinte
       {Define a posição dos vértices de controle ao cria-los}
        deltaX := (auxVertice2.getXvertice - auxVertice1.getXvertice) / 3;
        deltaY := (auxVertice2.getYvertice - auxVertice1.getYvertice) / 3;

       {Cria a matriz com os vertices e adiciona os vertices de controle}
        VerticesCurvos[i*4][0] := auxVertice1.getXvertice;
        VerticesCurvos[i*4][1] := auxVertice1.getYvertice;
        VerticesCurvos[i*4][2] := 0;
        VerticesCurvos[i*4+1][0] := auxVertice1.getXvertice + deltaX;
        VerticesCurvos[i*4+1][1] := auxVertice1.getYvertice + deltaY;
        VerticesCurvos[i*4+1][2] := 0;
        VerticesCurvos[i*4+2][0] := auxVertice2.getXvertice - deltaX;
        VerticesCurvos[i*4+2][1] := auxVertice2.getYvertice - deltaY;
        VerticesCurvos[i*4+2][2] := 0;
        VerticesCurvos[i*4+3][0] := auxVertice2.getXvertice;
        VerticesCurvos[i*4+3][1] := auxVertice2.getYvertice;
        VerticesCurvos[i*4+3][2] := 0;
    end;
    LinhasCurvas := True; //Define que esse objeto possui curvas
    LVertices.Clear;      //Limpa a lista de vértices
    LVertices.Free;       //Elimina a lista de vértices
end;
```

Quadro 12 – Método curvar linha.

A matriz criada por este método é carregada para o *OpenGL* utilizando-se o comando *glMap1d*, a seguir é acionada e então utilizada pelo comando *glEvalCoor1d* para criar os vértices necessários para a criação da curva. Após a criação dos vértices e necessário desenhar a linhas na tela utilizando o comando *GL_LINE_STRIP*. Estes comandos estão implementados no método desenhar linha, o quadro 13 mostra o trecho ocultado anteriormente responsável por desenhar curvas.

```
procedure TLinha.DesenhaLinha(TamTelapx, TamTelacm : double);
var
    OVertice : TVertice;
    i, j, fator : integer;
    esp : double;
begin
    ...
        if LinhasCurvas then //Se o objeto contem linhas curvas
        begin
            {A primeira linha contem as coordenadas do primeiro vértice
             a segunda e terceira dos pontos de controle e a
             quarta do próximo vértice, e assim por diante
             Porém é necessário informar ao comando qual
             é o primeiro vértice, por isso se pula quatro linhas}
            for j := 0 to (High(VerticesCurvos) div 4) do
            begin
                glMap1d(GL_MAP1_VERTEX_3,0,30,3,4,
                        @VerticesCurvos[j*4][0]);  //Carrega a matriz para o OpenGL
                glEnable(GL_MAP1_VERTEX_3);        //Ativa o comando
                glBegin(GL_LINE_STRIP);            //Desenha a linha
                for i := 0 to 30 do                //Percorre os vertices
                begin
                    glEvalCoord1d(i);              // Cria os vértices com as
                                                     infomações da matriz
                end;
            glEnd;
            end;
        end
        else
    ...
end;
```

Quadro 13 - Método desenhar linha curva.

A figura 22 apresenta a tela do sistema com o desenho de um objeto linha curva, os pontos azuis ligados aos vértices por linhas tracejadas são os pontos de controles usados para curvar as linhas..

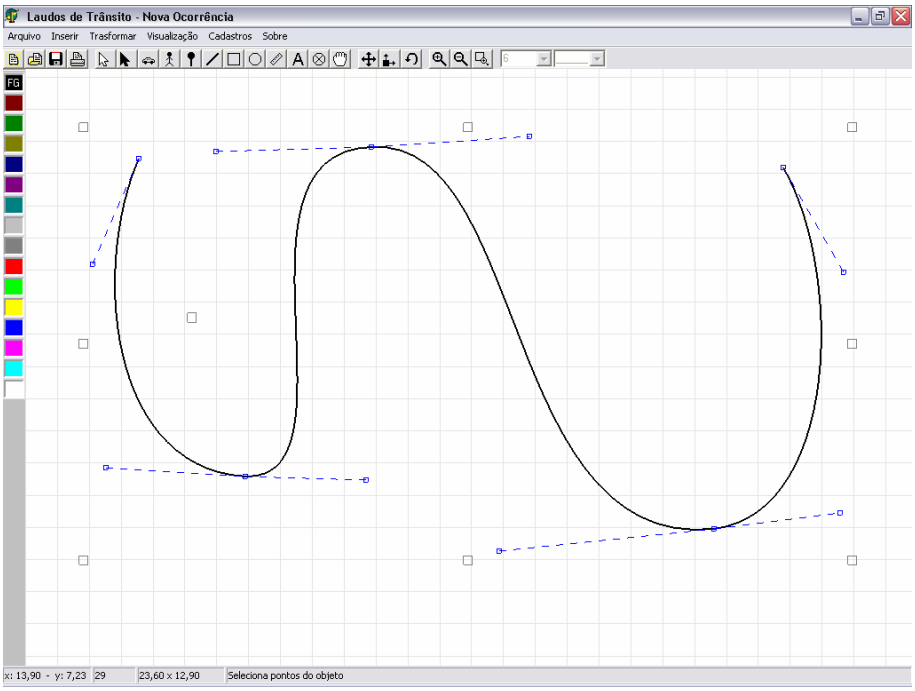


Figura 22 – Desenho de linhas curvas no sistema

3.5.5.5 Criação de Veículos, Placas e Pedestres

Os objetos de veículo, placas e pedestres seguem padrões de tamanho pré-definidos, não podendo ser alterados. Portanto para se criar um deles, o usuário deve selecionar a ferramenta e clicar sobre o desenho. Ao selecionar a ferramenta inserir veículo o sistema apresenta uma lista, onde o usuário seleciona o tipo de veículo que deseja inserir. Enquanto que ao selecionar a ferramenta de adicionar placas uma nova janela se abre e o usuário deve escolher qual placa deseja inserir. No caso do objeto pedestre não existem opções a escolher. Os quadros 7 e 8 mostram o algoritmo utilizado para se criar estes objetos.

Estes objetos têm em comum a forma em como são construídos. Todos os três objetos são agrupamentos de objetos do tipo linha e/ou circulo. Estes agrupamentos são armazenados pelos objetos em um atributo de lista. Por exemplo, um veículo pode ser composto por vários objetos de linha que são armazenados pelo objeto. Ao inserir um dos três objetos, a posição do clique do mouse é enviada ao método *create* para que se torne o ponto central do objeto, e a partir deste ponto o mesmo criará os objetos que o formam. Nos objetos veículo e placa além da posição são enviados ao método o tipo de veículo ou placa que o usuário escolheu em forma de um valor numérico entendível pelo sistema. O quadro 14 apresenta um trecho de código do método *create* da classe *TVeiculo*.

```
constructor TVeiculo.Create(xc, yc : double; tp : integer);
var
    auxLinha : TLinha;
begin
    LObjetos := TList.Create; //Cria lista de objetos
    CodigoBD := -1;           //Inicializa atributo
    XCentro := xc;            //Define coordenada x
    YCentro := yc;            //Define coordenada y
    Tipo := tp;               //Define o tipo de veículo
    case Tipo of              //Seleciona o tipo de veículo que sera criado
        0 : begin
                {Cria objetos que formaram um veículo do tipo 1}
                auxLinha := TLinha.Create;
                auxLinha.setCor(Red, Green, Blue);
                auxLinha.setEspessura(1);
                auxLinha.setObjeto;
                auxLinha.setVertice(xc-0.9, yc-2.1);
                auxLinha.setVertice(xc-0.9, yc+1.96);
                auxLinha.setVertice(xc-0.3, yc+2.1);
                auxLinha.setVertice(xc+0.3, yc+2.1);
                auxLinha.setVertice(xc+0.9, yc+1.96);
                auxLinha.setVertice(xc+0.9, yc-2.1);
                LObjetos.Add(auxLinha);
                auxLinha := TLinha.Create;
                auxLinha.setCor(0.0,0.6,0.8);
                auxLinha.setEspessura(1);
                auxLinha.setObjeto;
                auxLinha.setVertice(xc-0.51, yc+0.2);
                auxLinha.setVertice(xc-0.76, yc+0.84);
                auxLinha.setVertice(xc+0.76, yc+0.84);
                auxLinha.setVertice(xc+0.51, yc+0.2);
                LObjetos.Add(auxLinha);
                ... // Demais Objetos
            end;
```

Quadro 14 – Método *create* da classe *TVeiculo*.

Os métodos *create* das classes *TPlaca* e *TPedestre*, utilizam a mesma lógica de programação da classe *TVeiculo*. Porém o trecho de código que define e seleciona o tipo não existem na classe *TPedestre*, pois existe somente um tipo de pedestre.

O método de desenho dos veículos, placas e pedestres varrem a lista de objetos e chamam o método de desenho dos objetos que o compõe. O quadro 15 mostra a implementação do método desenhar placa. O método desenhar veículo e desenhar pedestre seguem a mesma lógica de implementação.

```
procedure TPlaca.DesenhaPlaca;
var
    i : integer;
    auxLinha : TLinha;
    auxCirculo : TCirculo;
    classe : string;
begin
    for i := 0 to LObjetos.Count -1 do                      //Percorre lista de
                                                              objetos
    begin
        classe := TObject(Lobjetos[i]).ClassName;           //Recebe a classe do
                                                              classe do objeto
        if classe = 'TLinha' then
        begin
            auxLinha := TObject(Lobjetos[i]) as TLinha;     //Recebe objeto linha
                                                              da lista
            auxLinha.DesenhaLinha(0,0);                     //Desenha objeto
        end;
        if classe ='TCirculo' then
        begin
            auxCirculo := TObject(Lobjetos[i]) as TCirculo; //Recebe objeto circulo
                                                              da lista
            auxCirculo.DesenhaCirculo;                      //Desenha Objeto
        end;
    end;
end;
```

Quadro 15 – Método desenhar placa.

A figura 23 apresenta a tela do sistema com o desenho de um objeto veículo, placa e pedestre.

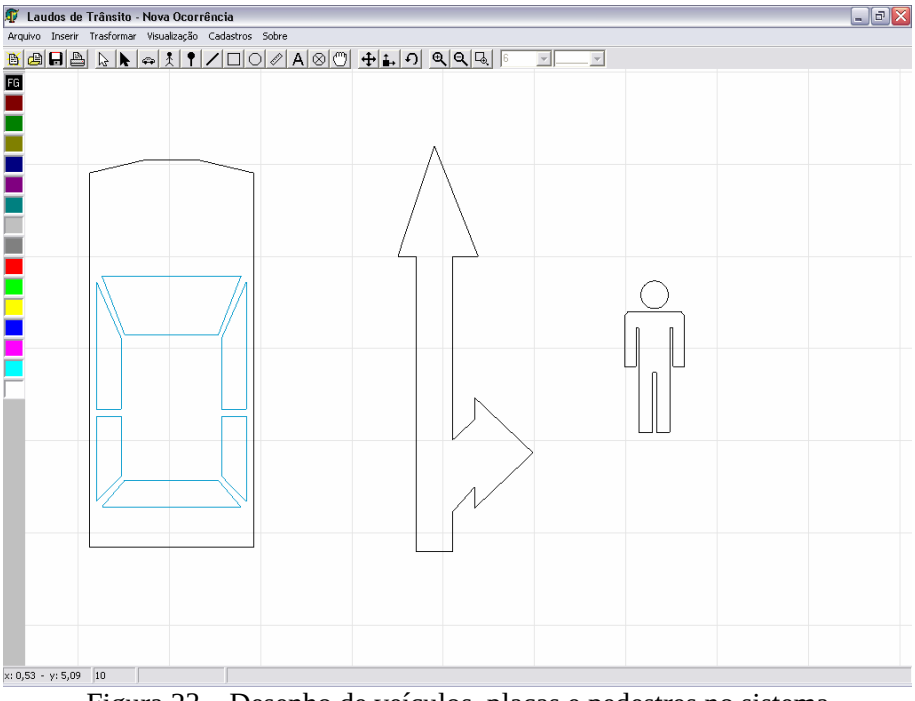


Figura 23 – Desenho de veículos, placas e pedestres no sistema

3.5.5.6 Criação de Legendas

O objeto legenda armazena um texto associado a qualquer objeto do tipo linha, retângulo ou círculo. Para criar uma legenda o usuário deve selecionar o objeto ao qual a legenda será inserida, selecionar a ferramenta e digitar a descrição na caixa de diálogo que o sistema apresenta. O objeto legenda criado é adicionado na lista de objetos do desenho, e em seguida o sistema armazena seu endereço de memória em um atributo do objeto associado a ele. O quadro 16 apresenta o algoritmo utilizado para a criação deste objeto.

```
Procedimento InserirLegenda
Variáveis
       Texto     //Armazena o texto digitado pelo usuário
Inicio
       Abre caixa de diálogo para usuário
       Texto := descrição digitada
       Criar objeto legenda
       Adiciona objeto legenda a lista de objetos
       Objeto legenda := texto
       Se retangulo = selecionado então
              Objeto retangulo := Ponteiro de memória do objeto legenda
       Else se circulo = selecionado então
              Objeto circulo := Ponteiro de memória do objeto legenda
       Else se linha = selecionado então
              Objeto linha := Ponteiro de memória do objeto legenda
Fim
```

Quadro 16 – Algoritmo para criação de objetos legenda.

A figura 24 apresenta a tela do sistema com uma legenda vinculada a um objeto retângulo. A legenda aparece no quarto painel da barra de status.

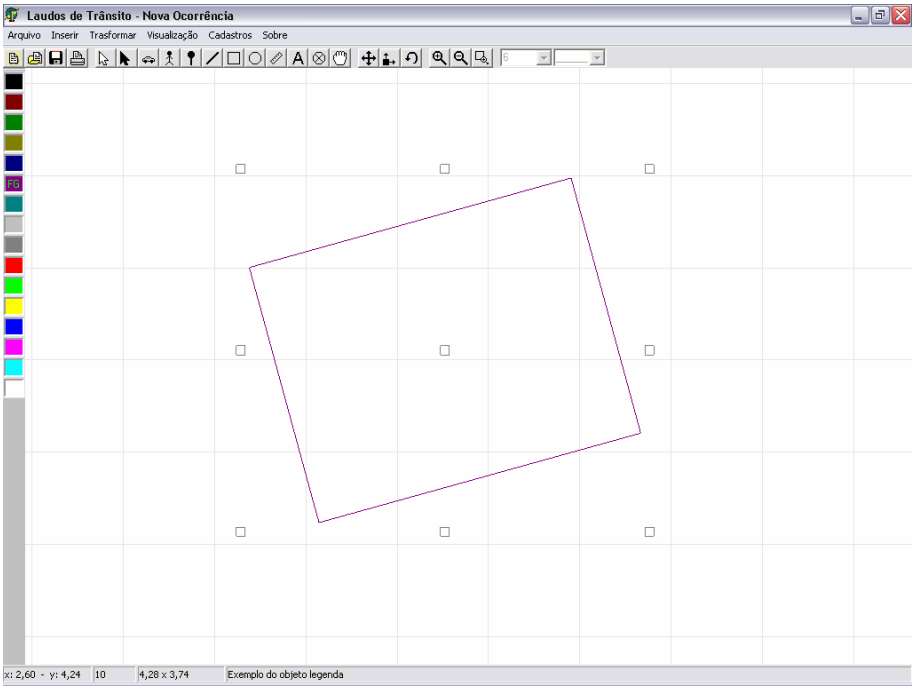


Figura 24 – Agregação de legenda a um objeto

3.5.5.7 Criação de Texto

O objeto texto permite ao usuário inserir um texto no desenho que fique visível a todo o momento. Para criar um objeto texto o usuário deve selecionar a ferramenta, clicar sobre o desenho e preencher a caixa de diálogo com o texto desejado. Ao final deste processo, o sistema define o ponto do clique como as coordenadas de início do objeto e armazena o texto digitado em um atributo. Os quadros 7 e 8 mostram o algoritmo utilizado para se criar estes objetos.

Para desenhar o objeto de texto é necessário primeiramente se criar um lista. Chamando o comando *wglUseFontBitmaps*, o *OpenGL* cria automaticamente o desenho de cada caracter a partir de uma fonte existente no sistema operacional e armazena nesta lista. O comando *glCallLists* identifica os caracteres necessários para a construção do texto armazenado no objeto e busca na lista criada o desenho dos caracteres para serem exibidos na tela. A implementação destes comandos é apresentada no quadro 17.

```
procedure TTexto.DesenhaTexto(AltTelapx, AltTelacm: double);
var
    auxFont : HFONT;         //Variável do tipo fonte do Windows
    auxList : GLuint;        //Variável de tipo lista do OpenGL
    propA, propL : integer;
begin
    propA := round((AltTelapx * 20) / (AltTelacm * 100));
    propL := round(propA * 0.35);                          //Calculos de proporção do
                                                             texto em relação a tela
    auxFont := CreateFont(propA,propL,0,0,FW_NORMAL,
               0,0,0,ANSI_CHARSET,OUT_TT_PRECIS,
               CLIP_DEFAULT_PRECIS,DRAFT_QUALITY,
               DEFAULT_PITCH, 'Times New Roman');        //Criação da fonte e de
                                                             suas características a
                                                             partir de uma fonte do
                                                             sistema
    SelectObject(FPrincipal.glDC,auxFont);                //Configura a fonte de
                                                             acordo do o DC do OpenGL
    auxList := glGenLists(128);                           //Cria uma lista com
                                                             128 posições
    wglUseFontBitmaps(FPrincipal.glDC,0,128,auxList);     //Insere os caracteres da
                                                             fonte na lista
    glColor3f(Red,Green,Blue);                            //Define a cor do texto
    glRasterPos2d(x,y);                                   //Define a posição do
                                                             texto
    glListBase(auxList);                                  //Chama a lista que sera
                                                             usada no método seguinte
    glCallLists(length(texto),
                GL_UNSIGNED_BYTE,pointer(texto));         //Utiliza os caracteres a
                                                             lista para desenhar na
                                                             tela o texto informado
end;
```

Quadro 17 – Método desenha texto.

A figura 25 apresenta a tela do sistema com o desenho de dois objetos texto.

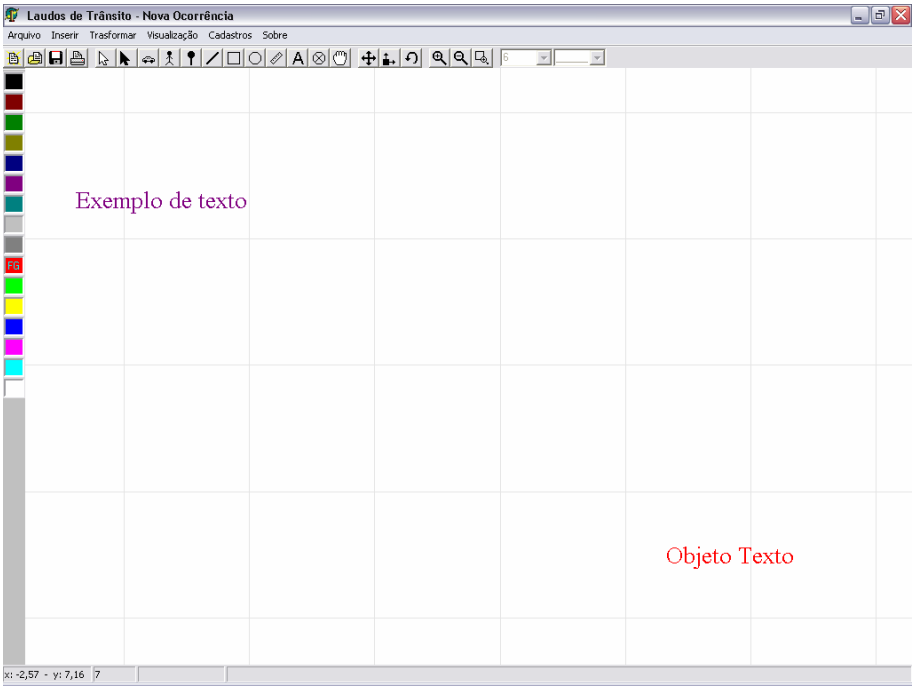


Figura 25 – Desenho de textos.

3.5.6 Seleção dos objetos

Utilizando a ferramenta de seleção o usuário define ao sistema qual objeto ele deseja trabalhar. Para selecionar um objeto o usuário seleciona a ferramenta e então clica sobre o elemento gráfico desejado. Ao selecionar um objeto são criados oito pontos ao seu redor e um no centro que servem para indicar que o objeto esta selecionado e para realizar as transformações de tamanho e rotação. Além desta todas as outras características do objeto podem ser alteradas quando os mesmos estão selecionados.

Para verificar se o clique do mouse foi realizado sobre um objeto ou sobre uma área vazia, o sistema percorre toda a lista de objetos e chama o método *Sobre* de cada objeto. Todos os objetos gráficos do sistema possuem este método que recebe as coordenadas do clique do mouse e verifica se elas estão dentro da área ocupada pelo objeto. Para realizar esta checagem, o método chama as coordenadas limites do objeto e compara com as coordenas enviadas pelo sistema, se estas coordenadas estão dentro da área do objeto o método retorna verdadeiro. Quando o método retorna verdadeiro, o sistema para de percorrer a lista e define o objeto corrente como selecionado. Cada elemento gráfico possui um atributo do tipo *boolean* denominado Selecionado, quando esta atributo esta definido como verdadeiro o sistema

entende que o objeto está selecionado. O método setSelecionado é responsável pelo estado deste atributo, portanto quando o sistema detecta que o objeto esta selecionado ele chama este método enviando um valor verdadeiro. O quadro 18 mostra o algoritmo utilizado pela ferramenta seleção.

```
variáveis
       classe //recebe o tipo de objeto corrente da lista
inicio
       do inicio ao final da lista faça
       inicio
              classe := classe do objeto corrente da lista
              se classe = retângulo então
              inicio
                     se sobreretângulo(mundoX,mundoY) = verdadeiro então
                     inicio
                            setSelecionado(Verdadeiro)
                            sair
                     fim
              fim
              se classe = círculo então
              inicio
                     se sobrecírculo(mundoX,mundoY) = verdadeiro então
                     inicio
                            setSelecionado(Verdadeiro)
                            sair
                     fim
              fim
              ... // Assim por diante com todas as classes
       fim
fim
```

Quadro 18 – Algoritmo para seleção objetos.

A figura 26 apresenta a tela do sistema com o desenho de um objeto círculo e ao redor dele os pontos de seleção incluindo o ponto de rotação que está no centro do objeto.

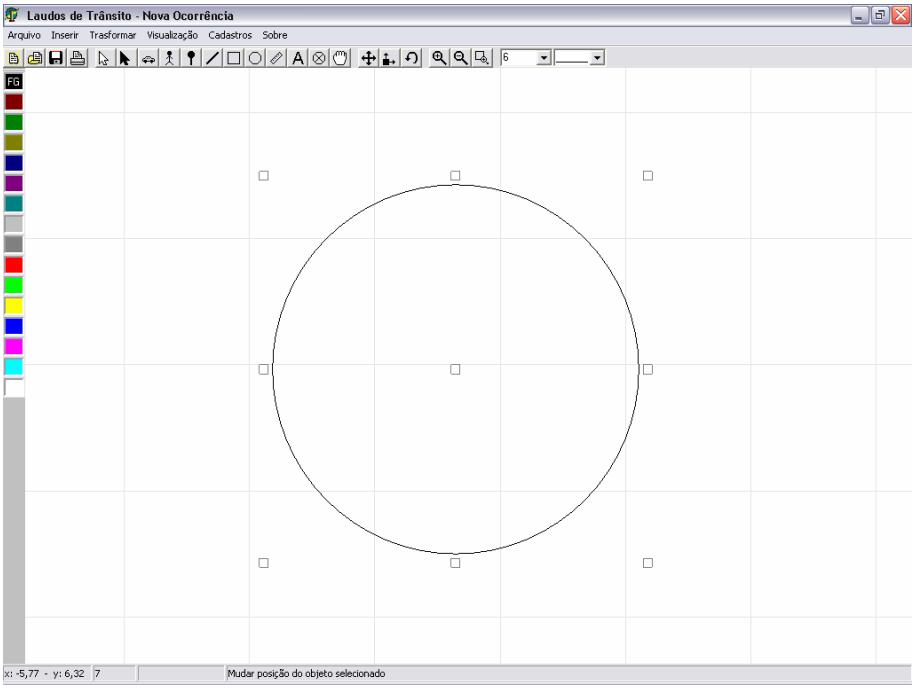


Figura 26 – Pontos de seleção.

Quando existe algum objeto selecionado e o usuário clica em uma área vazia do

desenho, o sistema chama o método *Deselecionar*, que é responsável por retirar a seleção de todos os elementos gráficos do sistema. Este método também é utilizado pelo sistema quando o usuário seleciona ou cria outro objeto. Dessa forma somente o objeto corrente permanece selecionado. O Quadro 19 apresenta o algoritmo utilizado neste método.

```
Procedimento deselecionar
variáveis
       classe //recebe o tipo de objeto corrente da lista
inicio
       do inicio ao final da lista faça
       inicio
              classe := classe do objeto corrente da lista
              se classe = retângulo então
              inicio
                     se sobreretângulo(mundoX,mundoY) = verdadeiro então
                            setSelecionado(Verdadeiro)
              fim
              se classe = círculo então
                     se sobrecírculo(mundoX,mundoY) = verdadeiro então
                            setSelecionado(Verdadeiro)
              fim
              ... // Assim por diante com todas as classes
       fim
fim
```

Quadro 19 – Algoritmo do método *Deselecionar*.

3.5.7 Transformação dos objetos

O sistema desenvolvido possibilita ao usuário manipular os objetos inseridos no desenho através das transformações. O aplicativo possui três tipos de transformações possíveis: redimensionar, rotacionar e reposicionar. Estas transformações estão implementadas dentro das classes do sistema. Porém alguns tipos de objetos possuem restrições referentes a algumas transformações. Nos próximos itens são apresentados maiores detalhes a respeito de cada transformação.

3.5.7.1 Redimensionar

Com esta ferramenta o usuário pode mudar o tamanho do objeto selecionado. Esta transformação está disponível somente para objetos do tipo círculo, linha e retângulo. Para isso, ele deve selecionar a ferramenta, clicar sobre um ponto de seleção e arrastar até o tamanho desejado. Os pontos de seleção influenciam na forma com o sistema altera o tamanho. Ao redimensionar um objeto pelos pontos de seleção localizados nas arestas, o

sistema altera o tamanho mas mantém o objeto proporcional à forma original. De outra forma nos pontos de seleção localizados mais ao meio, o sistema altera somente a altura ou a largura desconsiderando a proporção original do objeto. O quadro 20 apresenta a implementação do método *MudarTamanho* da classe *TLinha*, as outras classes seguem a mesma lógica de implementação.

```
procedure TLinha.MudarTamanho(XAtual, YAtual: double;
  Local: integer);
var
    deltax, deltay, x, y : double;
    i : integer;
    auxVertice : TVertice;
begin
    case Local of  //Variável que armazena o ponto de seleção clicado
       //O ponto de selação a seguir mantém o objeto proporcional
       0 : begin  //Ponto de seleção da aresta inferior esquerda
             deltax := (getMaiorX - getMenorX)
                       / (xatual - getMaiorX);    //Calcula o redimensionamento
             for i := 0 to LVertices.Count -1 do  //Percorre a lista de vértices
             begin
                 auxVertice := TObject(LVertices[i]) as TVertice; //Chama vértice
                 x := auxVertice.getXvertice; //Busca coordenada x do vértice
                 y := auxVertice.getYvertice; //Busca coordenada y do vértice
                 if x <> getMaiorX then //Checa se o vértice não é o ponto ancora
                     x := getMaiorX + ((getMaiorX - x) / deltax); //Calcula a nova
                                                                    coordenada de x
                                                                    do vértice
                 if y <> getMaiorY then //Checa se o vértice não é o ponto ancora
                     y := getMaiorY + ((getMaiorY - y) / deltax); //Calcula a nova
                                                                    coordenada de y
                                                                    do vértice

                 auxVertice.setVertice(x,y); //Salva novas coordenadas do ponto
              end;
            end;
        ...
            //O ponto de selação a seguir descondidera a proporção do objeto
        4 : begin //Ponto de seleção do centro esquerdo
              deltax := (getMaiorX - getMenorX)
                        / (xatual - getMaiorX);   //Calcula o redimensinamento
              for i := 0 to LVertices.Count -1 do // Percorre lista de vértices
              begin
                  auxVertice := TObject(LVertices[i]) as TVertice; //Chama vértice
                  x := auxVertice.getXvertice; //Busca coordenada x do vértice
                  y := auxVertice.getYvertice; //Busca coordenada y do vértice
                  if x <> getMaiorX then //Checa se o vértice não é o ponto ancora
                  begin
                     x := getMaiorX + ((getMaiorX - x) / deltax); //Calcula a nova
                                                                    coordenada de x
                                                                    do vértice
                     auxVertice.setVertice(x,y); //Salva novas coordenadas do ponto
                  end;
              end;
        ...
        end;
end;
//Ancora é o vértice oposto ao ponto de seleção clicado, esse ponto deve permanecer
com as mesmas coordenadas.
```

Quadro 20 – Método MudarTamanho da classe *TLinha*.

3.5.7.2 Rotacionar

Com esta ferramenta o usuário rotaciona o objeto ao redor de um ponto central de rotação. Esta transformação está disponível para todos os objetos gráficos com exceção do objeto texto. Para rotacionar um objeto, o usuário deve selecionar a ferramenta, clicar sobre um ponto de seleção e arrastar a até posição desejada. Ao selecionar um objeto o sistema define o centro do objeto como o ponto de rotação, mas este ponto pode ser alterado pelo usuário. Para isso o usuário deve clicar sobre o ponto e arrastá-lo para o local desejado. O sistema utiliza as fórmulas de produto escalar de vetores e produto vetorial para calcular a rotação de todos os vértices do objeto. O quadro 21 apresenta a implementação do método *Rotacionar* da classe *TRetangulo*, as outras classes seguem a mesma lógica de implementação.

```
procedure TRetangulo.Rotacionar(xantigo, yantigo, xatual, yatual, xc, yc: double);
var
    coseno, seno,xa, ya, xb, yb, ma, mb, auxX  : double;
begin
    {xantigo e yantigo armazenam as coordenadas do mouse no momento do clique,
     xatual e yatual armazenam as coordenadas do mouse depois de arrasta-lo,
     xc e yc armazenam as coordenadas do ponto de rotação}
    xa := xantigo - xc;
    ya := yantigo - yc; //Calcula A
    xb := xatual - xc;
    yb := yatual - yc;  //Calcula B
    ma := sqrt(sqr(yantigo-yc)+sqr(xantigo-xc)); //Calcula o módulo de A
    mb := sqrt(sqr(yatual-yc)+sqr(xatual-xc));   //Calcula o módulo de B
    coseno := ((xa * xb) + (ya * yb)) / (ma * mb); // Fórmula de produto escalar
                                                      para descobrir o coseno
if ((xa * yb) - (xb * ya)) > 0 then //Fórmula de produto vetorial para descobir o
                                      sentido da rotação do objeto
    begin
        seno := 1 * (sqrt(1 - sqr(coseno))); //Fórmula para calcular o seno
    end
    else
    begin
        seno := -1 * (sqrt(1 - sqr(coseno))); //Fórmula para calcular o seno
    end;
    {Fórmulas utilizadas para rotacionar os vértices do objeto, utilizam o seno e o
     coseno das fórmulas acima. Nessa fórmula é calculado as novas coordenadas de
     todos os vértices do objeto}
    auxX := x1;
    x1 := ((auxX - xc) * coseno) - ((y1 - yc) * seno) + xc; //Calcula o novo X
    y1 := ((auxX - xc) * seno) + ((y1 - yc) * coseno) + yc; //Calcula o novo Y
    auxX := x2;
    x2 := ((auxX - xc) * coseno) - ((y2 - yc) * seno) + xc;
    y2 := ((auxX - xc) * seno) + ((y2 - yc) * coseno) + yc;
    auxX := x3;
    x3 := ((auxX - xc) * coseno) - ((y3 - yc) * seno) + xc;
    y3 := ((auxX - xc) * seno) + ((y3 - yc) * coseno) + yc;
    auxX := x4;
    x4 := ((auxX - xc) * coseno) - ((y4 - yc) * seno) + xc;
    y4 := ((auxX - xc) * seno) + ((y4 - yc) * coseno) + yc;
end;
```

Quadro 21 – Método *Rotacionar* da classe *TRetangulo*.

3.5.7.3 Reposicionar

Com esta ferramenta o usuário pode alterar a posição do objeto dentro do desenho. Esta transformação está disponível para todos os objetos gráficos do sistema. Para reposicionar o objeto, o usuário deve selecionar a ferramenta, clicar sobre o objeto e arrastá-lo para o local desejado. O sistema também captura as coordenadas do clique do mouse e subtrai as coordenadas depois do arrasto, desta maneira ele calcula o deslocamento do arrasto e aplica este valor a todos os vértices do objeto. O quadro 22 apresenta a implementação do método *MudarPosicao* da classe *TRetangulo.* As outras classes seguem a mesma lógica de implementação

```
procedure TRetangulo.MudarPosicao(XAntigo, YAntigo, XAtual,
  YAtual: double);
var
    deltaX, deltaY : double;
begin
    {xantigo e yantigo armazenam as coordenadas do mouse no momento do clique,
     xatual e yatual armazenam as coordenadas do mouse depois de arrasta-lo}

    deltaX := XAtual - XAntigo; //calcula deslocamento de X
    deltaY := YAtual - YAntigo; //calcula deslocamento de Y
    x1 := x1 + deltaX; y1 := y1 + deltaY; //Aplica deslocamento aos vértices
    x2 := x2 + deltaX; y2 := y2 + deltaY;
    x3 := x3 + deltaX; y3 := y3 + deltaY;
    x4 := x4 + deltaX; y4 := y4 + deltaY;
end;
```

Quadro 22 – Método *MudarPosicao* da classe *TRetangulo*.

3.5.8 Exibição do desenho

Em um sistema de computação gráfica, o desenho exibido na tela é um imagem gerada a partir do dados armazenados na estrutura de dados. Por isso, toda vez que a estrutura é alterada, o sistema redesenha os objetos na tela. Estas alterações na estrutura de dados podem ser ocasionadas por alterações nas propriedades dos objetos inseridos e alterações na janela em que o desenho está sendo exibido. Para desenhar os objetos na tela, o sistema desenvolvido possui um método denominado *FormPaint*. Este método é responsável por limpar todos os objetos desenhados no universo *OpenGL*, percorrer a lista de objetos do desenho e acionar o método de desenho de cada um, conforme visto no item 3.5.5. Estas rotinas são executadas toda vez que este método é chamado pelo sistema. Quando um objeto está selecionado, este método aciona um outro método denominado *DesenhaSelecao*, que é

responsável por criar os pontos de seleção ao redor do objeto. O algoritmo do método *FormPaint* é apresentado no quadro 23.

```
procedimento FormPaint;
variáveis
       classe //recebe o tipo de objeto corrente da lista
inicio
       define cor de fundo do universo OpenGL
       limpa o universo OpenGL com a cor de fundo

       do inicio ao final da lista faça
       inicio
              classe := classe do objeto corrente da lista
              se classe = Retangulo então
              inicio
                     ObjetoRetangulo.DesenharRetangulo
                     se ObjetoRetangulo.getSelecionado = verdadeiro então
                            DesenhaSelecao(ObjetoRetangulo)
              fim
              se classe = Linha então
              inicio
                     ObjetoLinha.DesenharLinha
                     se ObjetoLinha.getSelecionado = verdadeiro então
                            DesenhaSelecao(ObjetoLinha)
              fim
              ... //Assim por diante com os demais tipo de objeto
       fim
fim
```

Quadro 23 – Algoritmo do método *FormPaint*

3.5.9 Manipulação de arquivos

O sistema desenvolvido permite ao usuário desenhar croquis e ocorrências de trânsito. Os dois tipos de desenho são tratados de forma diferente pelo sistema ao serem salvos. Cada um possui uma extensão de arquivo e sua própria pasta de armazenamento. Para os arquivos de ocorrência o sistema utiliza a extensão .ocr e são armazenados na pasta ocorrências. Para os croquis, utiliza a extensão .crq, que são armazenados na pasta croquis. Quando o usuário salva um desenho, o sistema identifica qual o tipo de desenho e define a pasta, o nome e a extensão do arquivo. A união destes três parâmetros forma o caminho de armazenamento do arquivo que serve de entrada para o método *SalvarDesenho*, responsável por transformar a lista de objetos que está armazenada na memória em um arquivo de texto. Para isto, o método cria o arquivo, percorre a lista de objetos e para cada objeto da lista escreve no arquivo o tipo do objeto e os seus atributos. Este método é comum para os dois tipos de desenho, pois são escritos da mesma forma no arquivo mas com extensões e pastas diferentes, definidos antes da chamada deste método. O quadro 24 mostra a algoritmo implementado neste método.

```
Procedimento SalvarDesenho(Caminho)
Variável
    Arquivo : TextFile;
    ...
inicio
    associa do arquivo externo a váriavel
    se o arquivo existe faça
        deletar o arquivo
    cria o arquivo e abre para inserir os dados
    se a lista não esta vazia faça
    inicio
        do inicio ao final da lista faça
            inicio
                classe := classe do objeto corrente da lista
                se classe = retângulo então
                 inicio
                    associa variável ao objeto da lista
                    escreve no arquivo o tipo do objeto
                    chama os atributos do objeto
                    escreve no arquivo os atributos do objeto
                    termina descrição do objeto
                fim
                se classe = círculo então
                inicio
                    associa variável ao objeto da lista
                    escreve no arquivo o tipo do objeto
                    chama os atributos do objeto
                    escreve no arquivo os atributos do objeto
                    termina descrição do objeto
                fim
                ...//Assim por diante com demais tipos de objetos
            fim
        fim
        termina a associação da variável ao arquivo
fim
```

Quadro 24 – Algoritmo do método *SalvarDesenho*

Para abrir os arquivos salvos, o sistema possui o método *AbrirDesenho*. Este método recebe do sistema o caminho do arquivo a ser aberto, lê os dados gravados no arquivo e cria os objetos na lista do desenho. Este método funciona da mesma forma para os dois tipos de desenho. O tipo de desenho que será aberto é definido antes da chamada do método por outra rotina do sistema que prepara as demais funcionalidades para receber o arquivo. O quadro 25 mostra o algoritmo deste método.

```
Procedimento AbrirDesenho(Caminho)
Variável
    Arquivo : TextFile;
    ...
inicio
    cria novo desenho
    associa do arquivo externo a váriavel
    abre o arquivo
    enquanto nao for o final do arquivo faça
        inicio
            le arquivo e armazena na variável
            se variavel = retângulo então
             inicio
                cria objeto
                lê atributos do objeto escritos no arquivo
                insere os atributos lidos no objeto criado
                insere objeto na lista
            fim
            se variável = círculo então
            inicio
                cria objeto
                lê atributos do objeto escritos no arquivo
                insere os atributos lidos no objeto criado
                insere objeto na lista

            fim
             ...//Assim por diante com demais tipos de objetos
        fim
    termina a associação da variável ao arquivo
fim
```

Quadro 25 – Algoritmo do método AbrirDesenho

3.5.10 Armazenamento dos arquivos no banco de dados

Os croqui são um mapeamento de um determinado local, e para isso devem armazenar a rua e um ponto de referência do mesmo, para que mais tarde o usuário possa localizá-los. No momento em que o usuário salva um croqui, o sistema solicita estas informações para o usuário e as armazena no banco de dados junto com o caminho de armazenamento do arquivo referente àquele desenho. Da mesma maneira, quando o usuário salva uma ocorrência, o sistema armazena junto com as informações do acidente o caminho de armazenamento do arquivo gerado. Desta forma o usuário localiza os croquis e ocorrências salvas através das informações referentes a cada um, e ao abrir o desenho selecionado, o sistema busca o caminho do arquivo no banco de dados e o envia ao método *AbrirDesenho*.

Nos desenhos do tipo ocorrência, o usuário tem a possibilidade de inserir objetos do tipo veículo e pedestre. Os objetos do tipo veículo possuem todas as informações a respeito do veículo, condutor, proprietário e vítimas. Estas informações são armazenadas no banco de dados juntamente com um código gerado automaticamente pelo sistema. O objeto desenhado

na tela é associado ao banco de dados através deste código, que fica armazenado em um atributo do objeto. Ou seja, toda vez que o usuário deseja ver as informações referentes um veículo, o sistema chama o código do objeto e busca por ele no banco de dados. Da mesma forma, o objeto pedestre armazena as informações referentes a uma pessoa e possui um atributo onde é salvo o código gerado pelo sistema.

3.5.11 Operacionalidade da implementação

Nesta seção é apresentada a operacionalidade da ferramenta desenvolvida. A figura 27 apresenta a tela inicial do sistema onde o usuário pode optar entre abrir ou criar uma nova ocorrência.

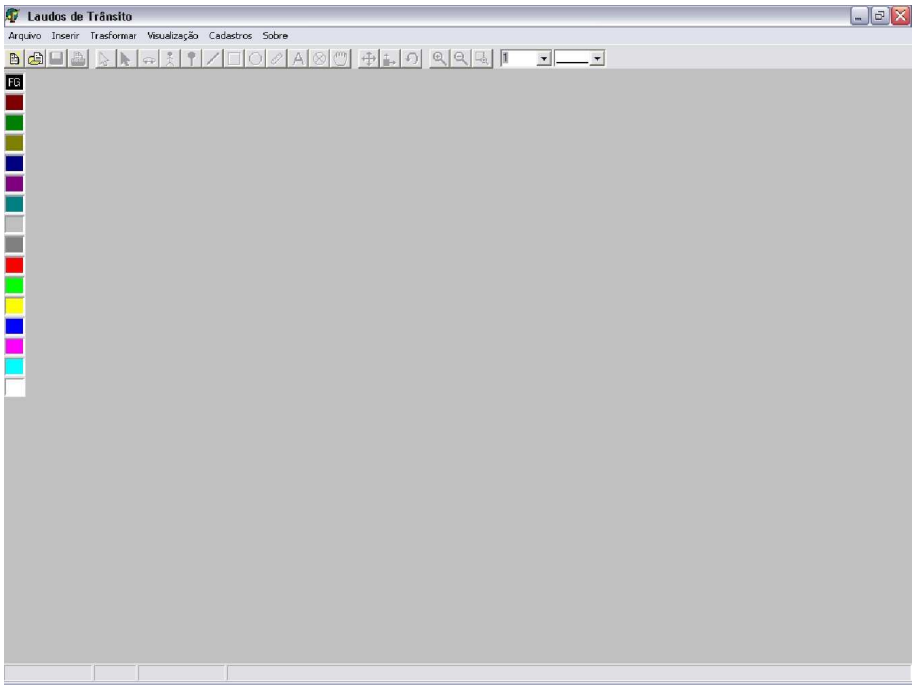


Figura 27 – Tela inicial da ferramenta.

Após abrir ou criar um arquivo as ferramentas do sistema podem ser acessadas. A figura 28 apresenta a localização das ferramentas na aplicação.

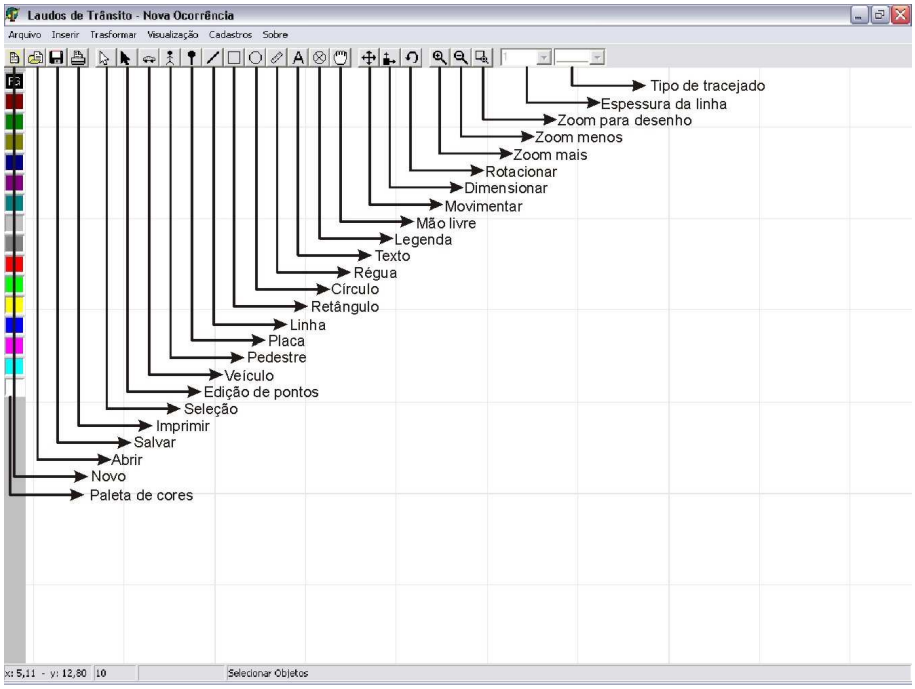


Figura 28 – Localização das ferramentas.

Para criar um objeto do tipo retângulo o usuário deve clicar onde deseja o primeiro vértice, arrastar até onde deseja o segundo vértice e então soltar o botão do mouse. Da mesma maneira para se criar um círculo o usuário deve clicar onde deseja o centro e arrastar até o tamanho desejado e então soltar o botão. Para se criar um objeto de texto o usuário deve clicar sobre o desenho onde deseja que o texto seja inserido e preencher a caixa de diálogo que aparecer com o texto desejado. Para criar um objeto linha o usuário deve clicar sobre o desenho, onde deseja colocar os vértices que formaram o elemento. Após desenhar um objeto do tipo linha o usuário pode clicar duas vezes sobre um vértice com a ferramenta de edição de pontos para transformar a linha em uma curva e através dos pontos de controle curvar conforme desejado. A figura 29 apresenta a criação destes objetos. Nesta mesma figura há um objeto linha selecionado e nele estão visíveis seus dois vértices, seus dois pontos de controle e os pontos de seleção. Este mesmo objeto também possui uma legenda que pode ser observada no quarto painel da barra de status. Para inserir uma legenda o usuário deve selecionar o objeto que ira recebe-la e clicar sobre a ferramenta legenda.

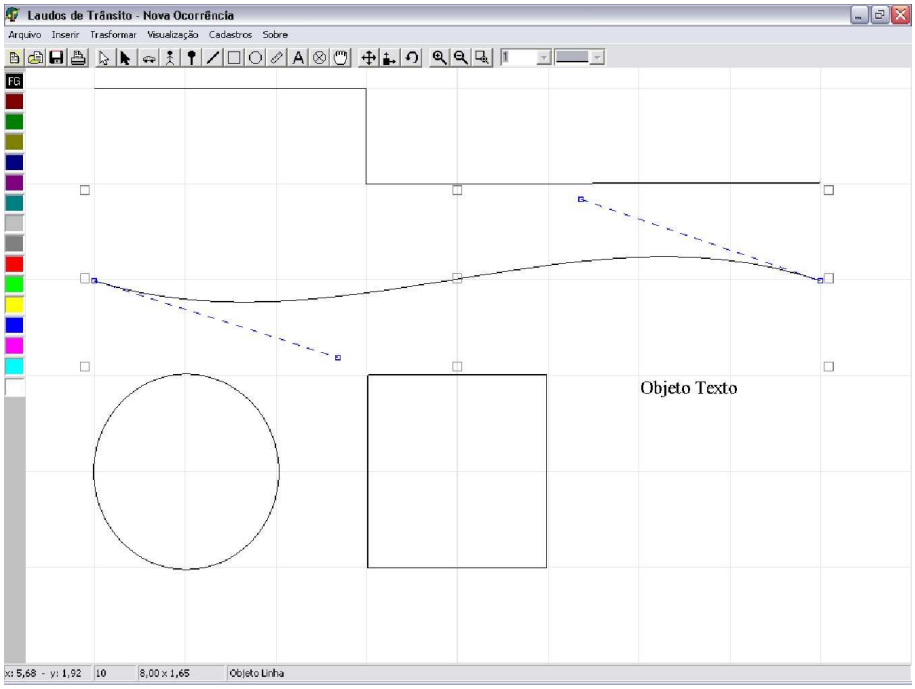


Figura 29 – Criação de objetos.

Para inserir um objeto do tipo placa é necessário escolher o tipo de placa desejada em um lista de opções que aparece em uma caixa de diálogo ao clicar sobre a ferramenta. Após selecionar a placa o usuário deve clicar sobre o desenho para que a mesma seja inserida. Essa caixa de diálogo é apresentada na figura 30.

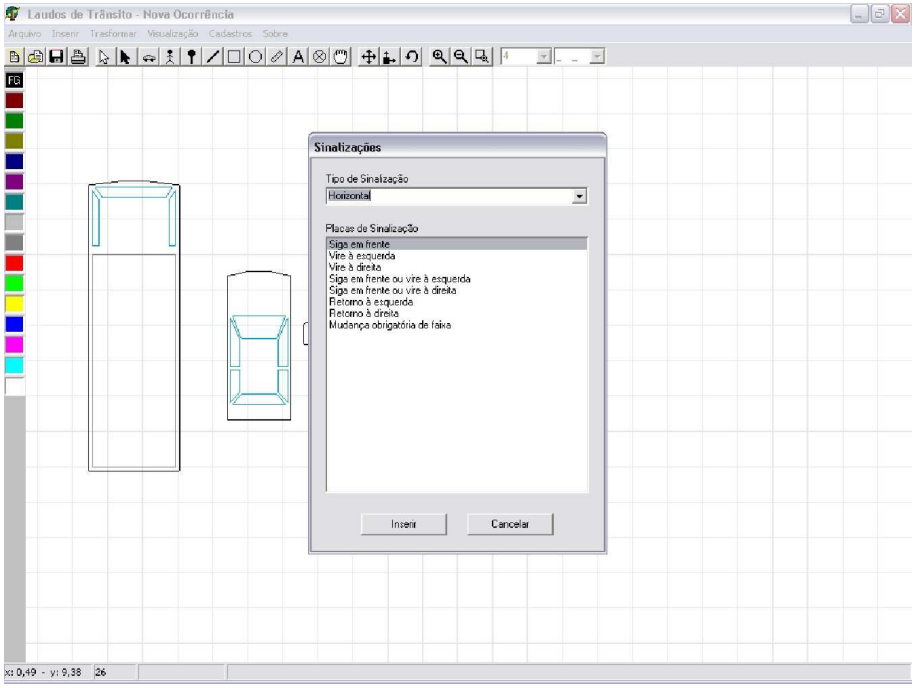


Figura 30 – Caixa de diálogo da ferramenta placas de trânsito.

A figura 31 apresenta a criação de dois objetos do tipo veículo um objeto do tipo pedestre e outro do tipo placa de trânsito. Para inserir um veículo o usuário deve selecionar o tipo de veículo desejado e clicar sobre o desenho. Para inserir um pedestre o usuário deve

selecionar a ferramenta e clicar sobre o desenho. Nesta mesma figura há um objeto do tipo régua selecionado e a distância medida por esta régua é mostrada no segundo painel da barra de status. Para inserir uma régua o usuário deve clicar sobre o ponto inicial e em seguida sobre o ponto final da medição.

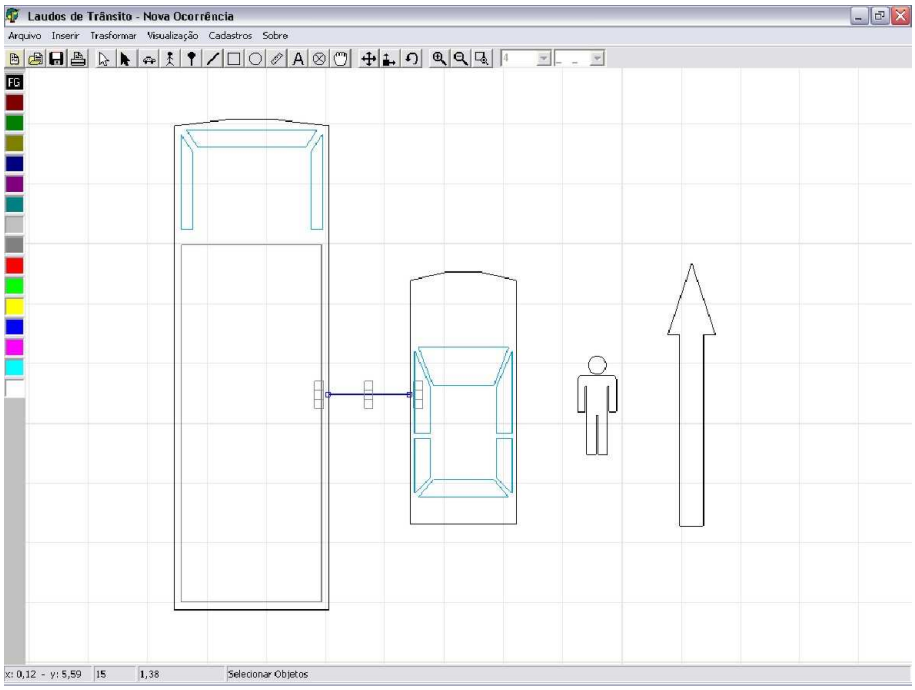


Figura 31 – Criação de objetos.

Utilizando estas ferramentas o usuário pode realizar o desenho dos croquis e em seguida armazená-los no sistema. Ao salvar o croqui, uma janela se abre solicitando ao usuário a digitação da rua e um ponto de referência do local desenhado. Esta janela é apresentada na figura 32.

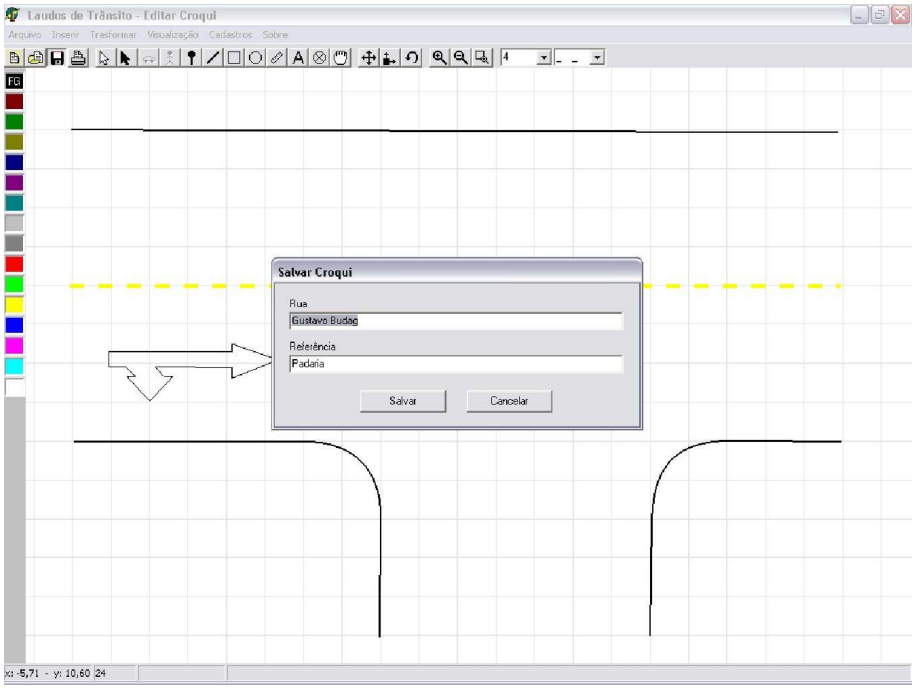


Figura 32 – Salvar croqui.

Na criação de uma nova ocorrência o usuário deve selecionar o croqui previamente desenhado do local. A figura 33 apresenta a tela onde o usuário seleciona o croqui.

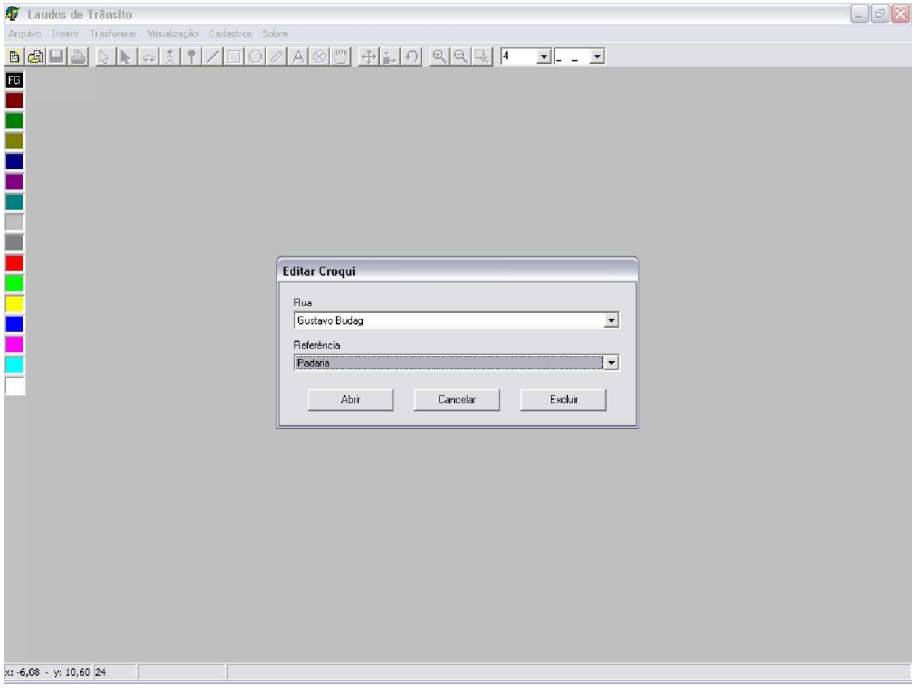


Figura 33 – Seleção do croqui da ocorrência.

Depois do usuário selecionar o croqui o sistema abre um novo desenho baseado no croqui e em seguida apresenta uma janela onde o usuário deve preencher as informações do acidente. Após fechar a janela o usuário pode acessá-la novamente clicando duas vezes sobre uma parte vazia do desenho. A figura 34 apresenta a janela onde são cadastradas as informações do acidente.

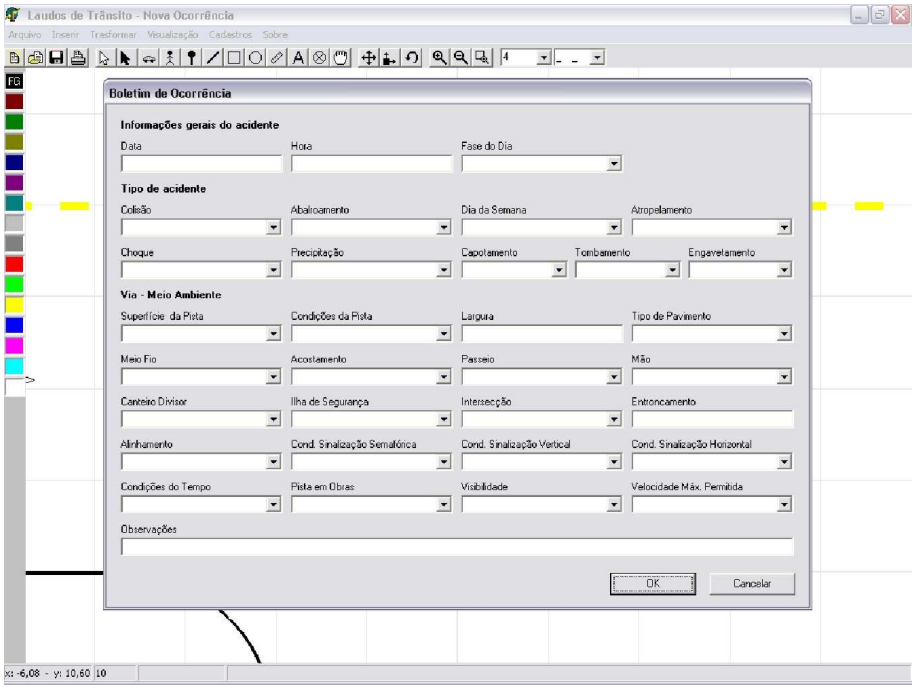


Figura 34 – Cadastro das informações do acidente.

Utilizando as ferramenta do sistema o usuário pode montar o desenho do local do acidente sobre o croqui aberto, como mostra a figura 35.

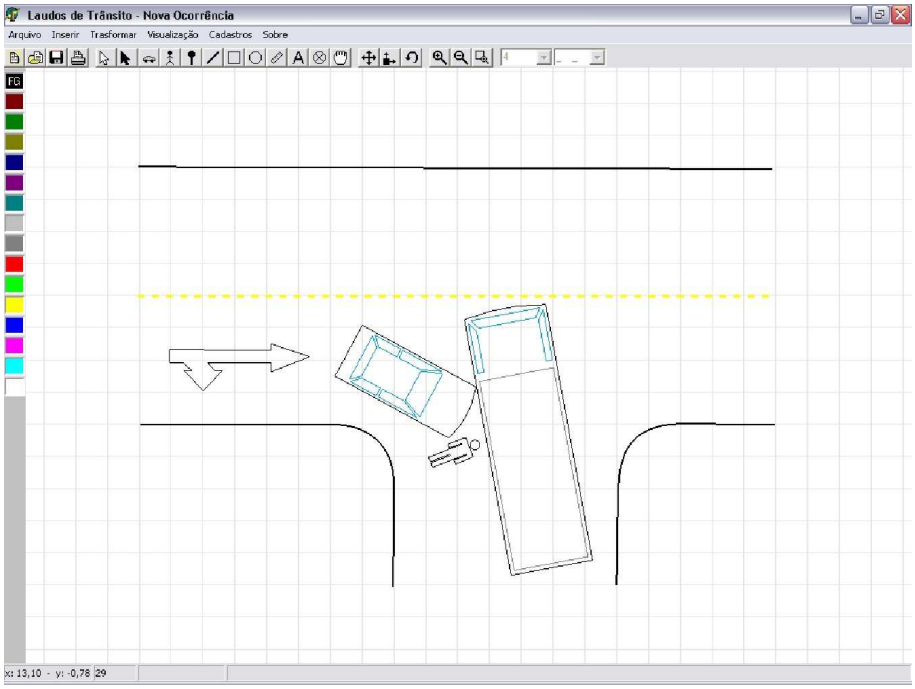


Figura 35 – Desenho de uma ocorrência.

Para inserir as informações referentes a cada veículo o usuário deve selecionar o mesmo e clicar duas vezes sobre ele, em seguida o sistema apresenta uma janela onde o usuário deve preencher as informações solicitadas. A figura 36 apresenta a janela onde são inseridas as informações do veículo.

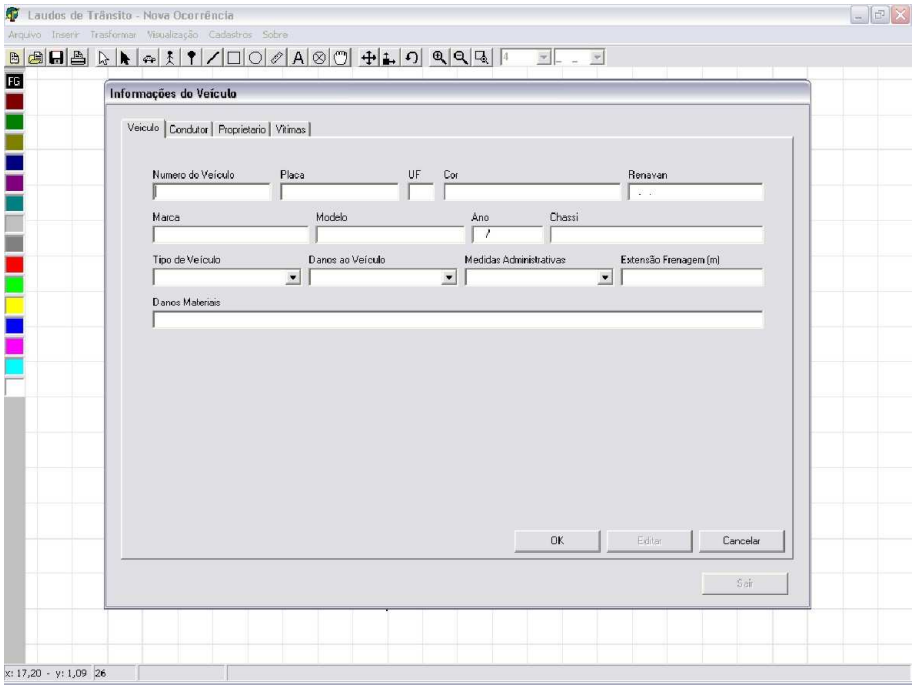


Figura 36 – Cadastro das informações do veículo.

A figura 37 apresenta a janela onde são inseridas as informações do condutor do veículo.

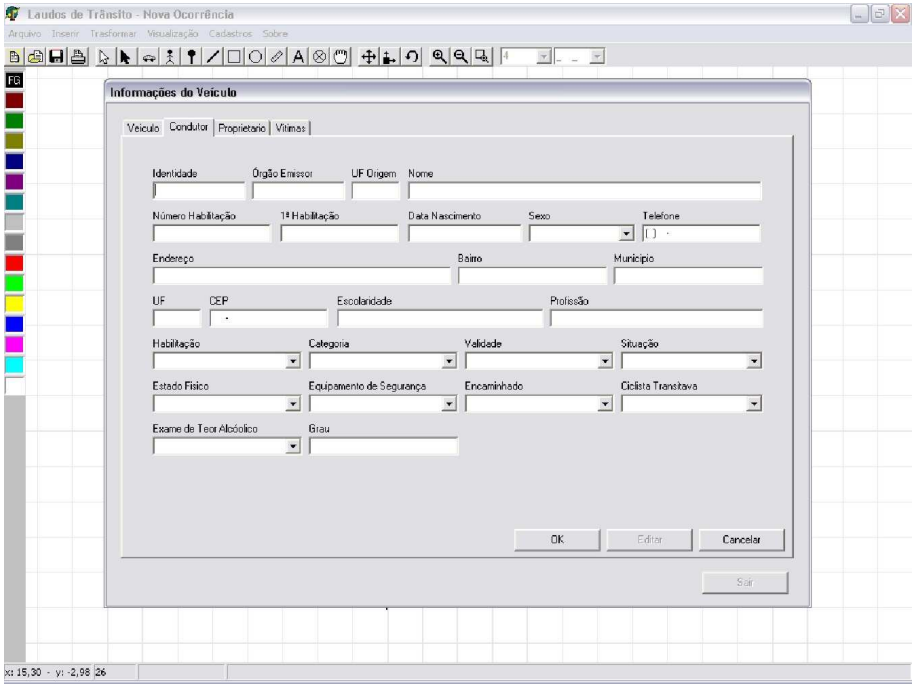


Figura 37 – Cadastro das informações do condutor.

A figura 38 apresenta a janela onde são inseridas as informações do proprietário do veículo.

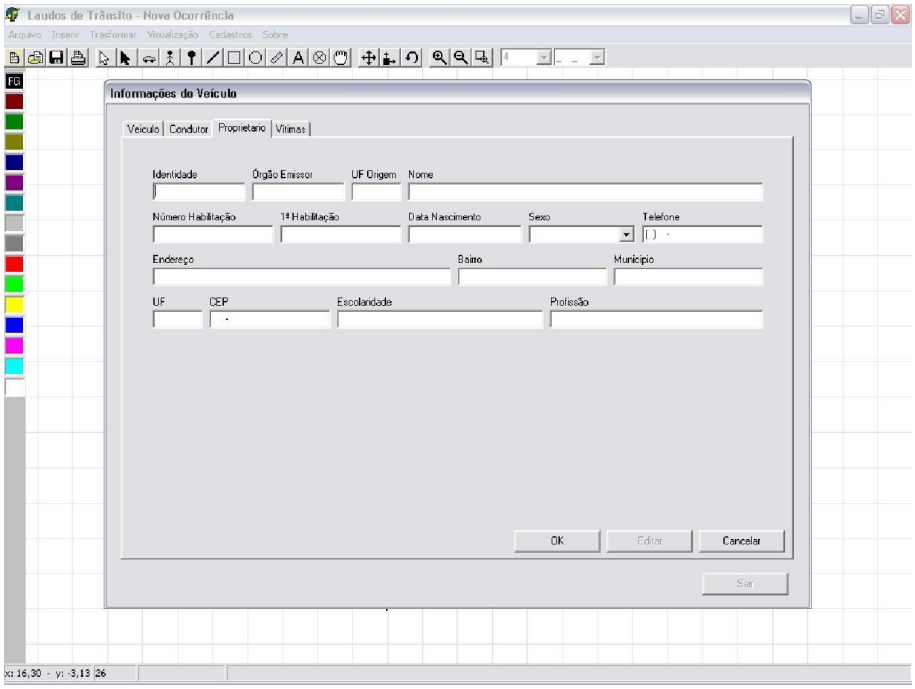


Figura 38 – Cadastro das informações do proprietário.

A figura 39 apresenta a janela onde são inseridas as informações das possíveis vítimas do veículo

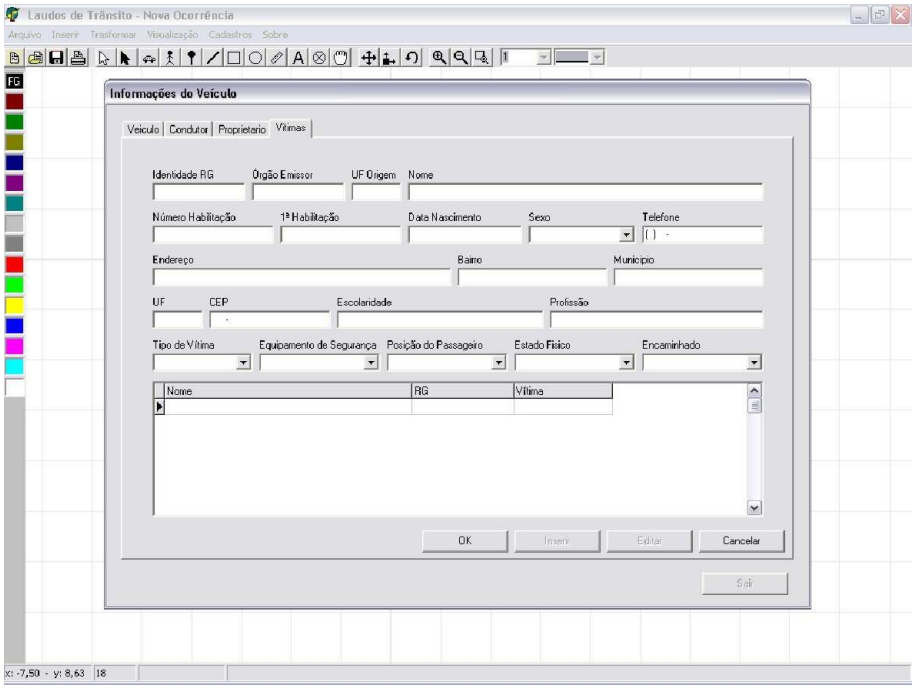


Figura 39 – Cadastro das informações das vítimas.

Os informações dos pedestres envolvidos no acidente são armazenadas no cadastro de vítimas, mas podem ser associadas ao objeto. Para isso o usuário deve selecionar o objeto pedestre e clicar duas vezes sobre ele, uma janela se abrirá contendo a lista de nomes dos envolvidos no acidente. Nesta lista o usuário seleciona a pessoa que deseja associar àquele elemento gráfico. A figura 40 apresenta a janela onde são inseridas as informações dos

pedestres.

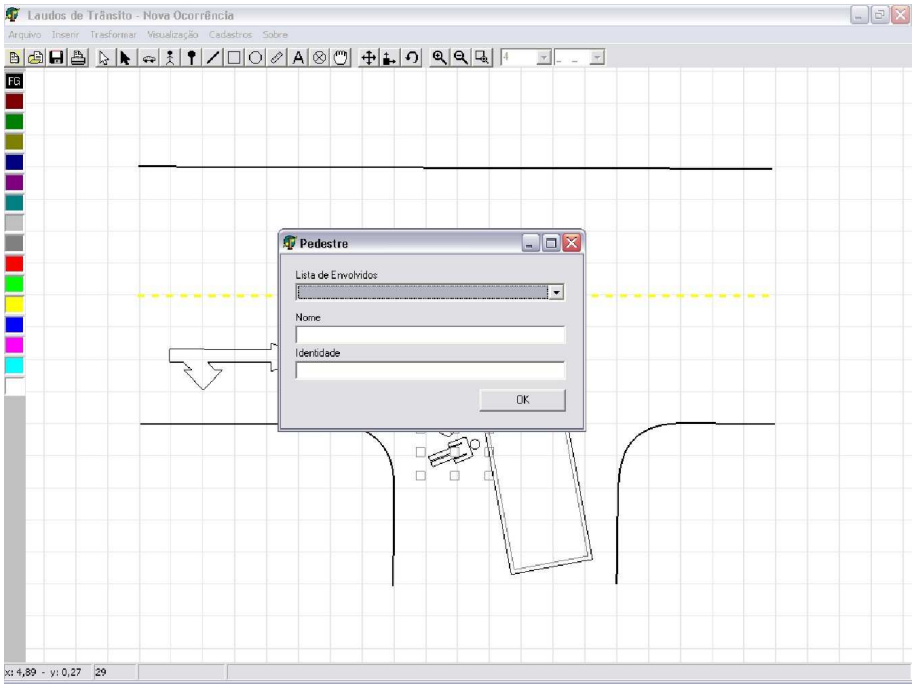


Figura 40 – Cadastro das informações dos pedestres.

Todas as pessoas envolvidas no acidente podem descrever a sua versão do acidente. O usuário pode inserir estas versões no cadastro de declarações onde ele seleciona a pessoa envolvida e clica sobre o botão de inserir. As pessoa que já possuem declaração ficam com seus nomes listados na tabela da janela. A figura 41 apresenta a janela de declarações.

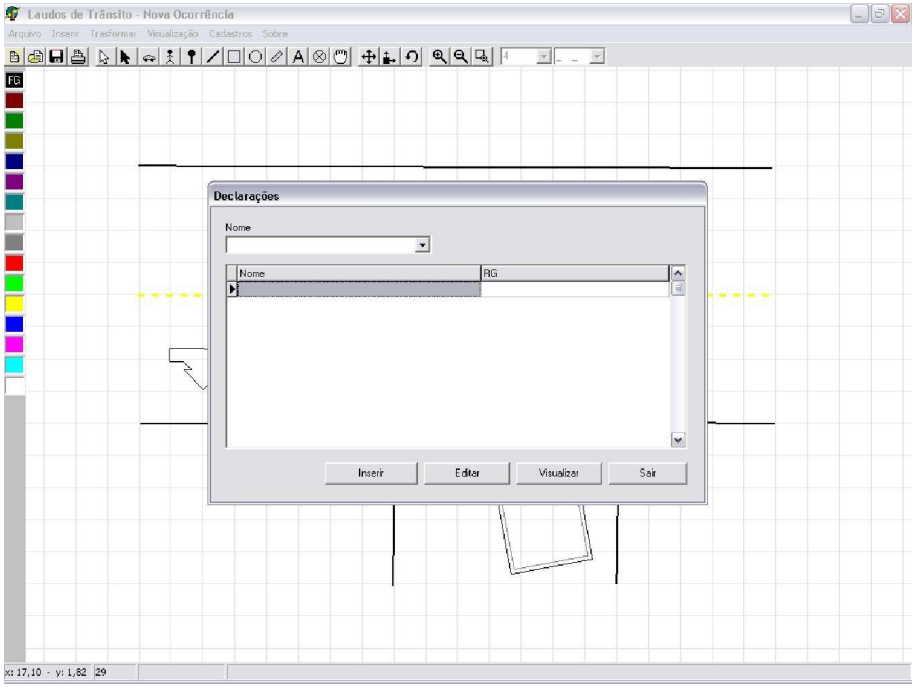


Figura 41 – Janela de declarações.

Ao clicar sobre botão inserir uma outra janela se abre onde o usuário pode digitar o relato do envolvido. A figura 42 apresenta a janela onde é cadastrada a declaração do

envolvido.

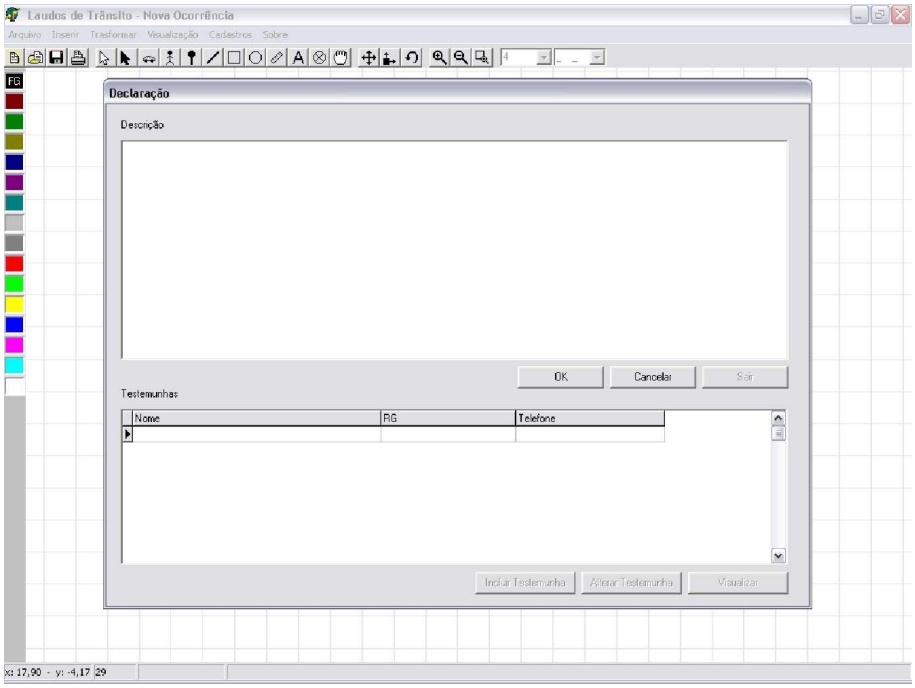


Figura 42 – Cadastro das declarações dos envolvidos

Ao salvar a declaração, o usuário pode adicionar as testemunhas da versão relatada pelo envolvido clicando sobre o botão incluir testemunha. Ao clicar sobre este botão o sistema apresenta uma janela onde o usuário deve preencher as informações da testemunha. As testemunhas cadastradas são apresentadas na tabela da janela de declarações. A figura 43 apresenta a janela onde são cadastradas as testemunhas.

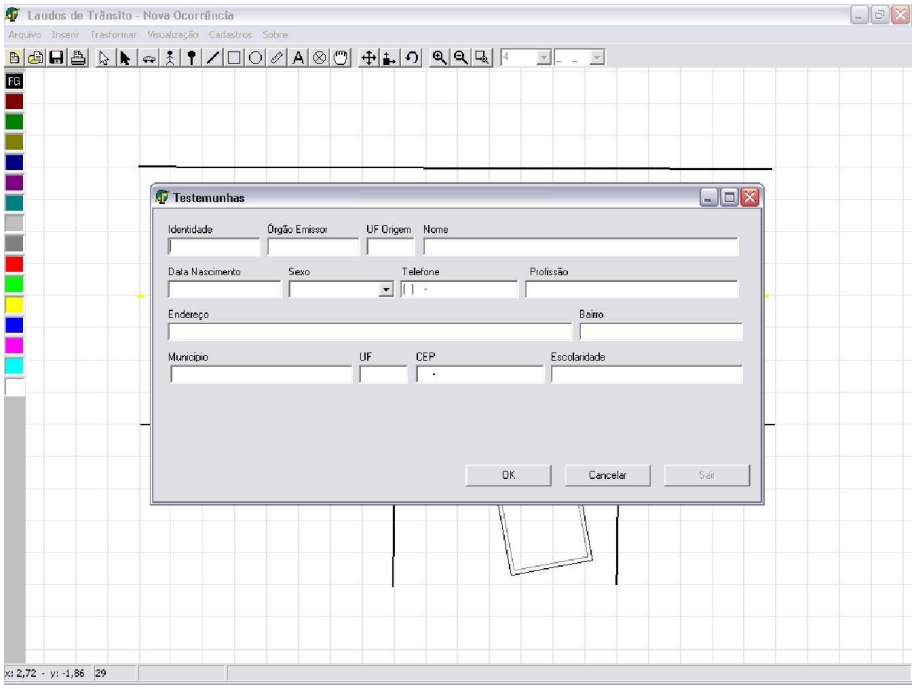


Figura 43 – Cadastro das informações da testemunha

Após terminar o desenho e inserir as informações necessárias o usuário pode salvar a

ocorrência clicando sobre o botão salvar da tela principal. Ao abrir uma ocorrência salva o sistema solicita ao usuário a digitação da placa do veículo envolvido ou a identidade de uma das pessoas envolvidas. Quando o usuário digita a placa do veículo o sistema lista todos os acidentes que este veículo se envolveu junto com a data e o local. Da mesma forma, se o usuário digitar a identidade de uma pessoa o sistema lista todos os acidentes que esta pessoa se envolveu junto com a data e o local. A figura 44 mostra a janela para abrir as ocorrências.

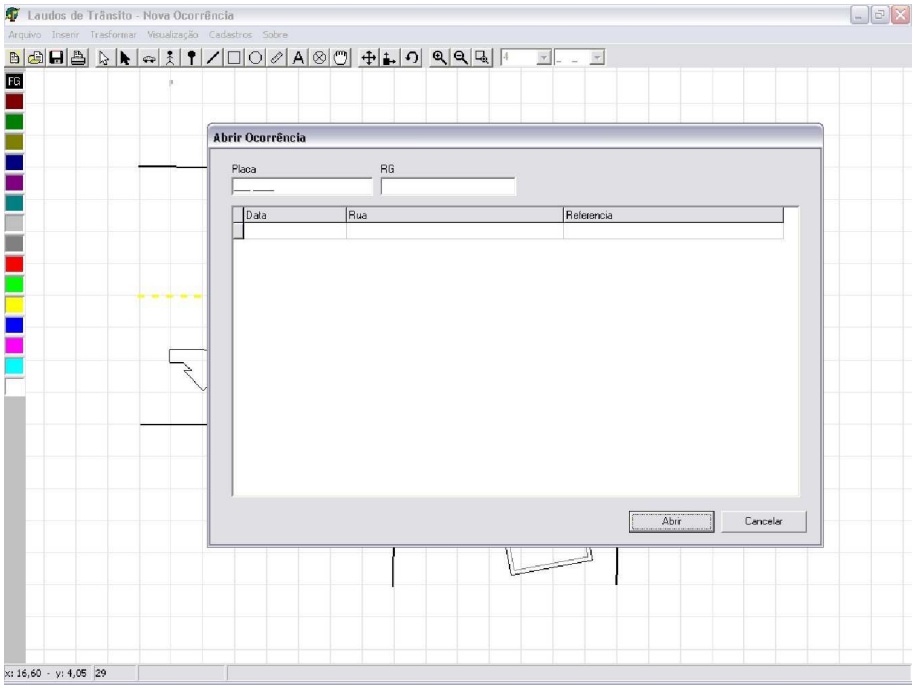


Figura 44 – Abrir uma ocorrência.

3.6 RESULTADOS E DISCUSSÃO

O sistema desenvolvido fornece ao agente de trânsito um conjunto de ferramentas gráficas que possibilita o desenho do local do acidente. Para facilitar e agilizar o processo de desenho, a ferramenta possui dois tipos de desenho: os croquis e as ocorrências. Os croquis são desenhos que armazenam um trecho de via com todos os detalhes como canteiros, calçadas, placas e etc. Estes desenhos são armazenados pelo sistema utilizando-se o nome da via e um ponto de referência, desta forma um agente pode abrir o croqui e montar o desenho do acidente de trânsito quando for atender a uma ocorrência neste local. Com a utilização dos croquis, o agente de trânsito precisa desenhar o trecho da via uma única vez e todas as ocorrências acontecidas naquele local podem ser documentadas utilizando-se sempre o

mesmo croqui, eliminado assim o retrabalho. Para desenhar uma ocorrência, a ferramenta disponibiliza ao agente objetos gráficos prontos para montar o acidente, tais como veículos, pedestre e placas de sinalização, dessa forma facilitando e agilizando o processo de criação do desenho. O sistema permite ao guarda cadastrar e armazenar em um banco de dados todas as informações não gráficas do acidente associadas ao desenho da ocorrência.

Durante a etapa de validação, o sistema foi apresentado à supervisora especialista da aplicação Sra. Odete Brancher Becker, e ao analista de sistemas do SETERB, Sr. José Vilson Bertoldi. Na apresentação foi simulado um atendimento a uma ocorrência onde não havia um croqui pronto e uma ocorrência onde já havia um croqui. Dessa forma foi possível mostrar a criação de ambos os tipos de desenho, bem como a forma de ser trabalhar com os croquis no sistema. Durante a apresentação das ocorrências foram inseridos dados fictícios no sistema para que fosse possível apresentar as suas funcionalidades de cadastro de informações. Segundo Becker (2006), a utilização dos croquis é muito interessante já que dessa maneira é possível o desenho imediato dos locais onde acontecem um maior número de acidentes, e mais tarde de todos os outros pontas da cidade. Bertoldi (2006) completa dizendo que o aplicativo atende perfeitamente as necessidades da guarda em termos de desenho, pois as ferramentas gráficas disponíveis no mercado não são tão especializadas como a ferramenta desenvolvida. A parte de armazenamento das informações do acidente também obteve um resultado positivo durante a validação. Segundo Becker (2006), com os dados coletados e inseridos no sistema no local do acidente o processo de emissão deve se tornar mais rápido e seguro, dessa forma eliminando o retrabalho necessário no sistema atual, além de eliminar também os erros gerados pela redigitação de informações.

O sistema desenvolvido obteve um resultado positivo em sua conclusão. Os principais problemas encontrados durante a etapa de análise, que eram o retrabalho e o trabalho manual, podem ser eliminados com a utilização da aplicação proposta no presente trabalho.

.

4 CONCLUSÕES

O presente trabalho apresentou o desenvolvimento de um protótipo de sistema cujo principal objetivo é eliminar o trabalho manual na elaboração dos laudos de trânsito. O protótipo de aplicativo desenvolvido oferece ao usuário um conjunto de ferramentas gráficas que possibilitam o desenho dos acidentes. Para facilitar a confecção do desenho a ferramenta utiliza o sistema de croquis, ou seja, o usuário desenha um local e armazena ele com o nome da rua e um ponto de referência e dessa forma o mesmo desenho pode ser utilizado no desenvolvimento de várias ocorrências. Os objetos mais usados pela guarda como veículos, placas e pedestres já estão prontos no sistema, dessa forma ao montar o desenho de um acidente o usuário pode inserir estes objetos e posicioná-los da maneira desejada sem precisar redesenhá-los toda vez que for fazer um novo croqui. O sistema permite ainda ao agente de trânsito cadastrar diretamente no desenho, associadas aos objetos, todas as informações referentes ao acidente, incluindo dados sobre as pessoas envolvidas.

Um dos objetivos específicos da proposta se referia à impressão dos laudos criados pelos sistema de acordo com padrões utilizados pela guarda. Porém este objetivo não pôde ser alcançado, pois durante o desenvolvimento do trabalho percebeu-se que o *OpenGL* gerava uma imagem a partir da estrutura de dados que poderia ser mostrada no monitor mas não impressa. Por isso para atender este requisito é necessário uma pesquisa mais aprofundada afim de descobrir como transformar os objetos gerados na tela pelo *OpenGL* em um desenho possível de ser impresso.

A utilização da biblioteca *OpenGL* foi muito eficaz para transformar a estrutura de dados em objetos e exibi-los na tela. Apesar de ser um biblioteca com vários comandos, o *OpenGL* mostrou-se muito simples e confiável. O maior problema encontrado foi a falta de literatura em português, a maior parte dela está em inglês, o que dificultou um pouco o desenvolvimento da aplicação. Outra dificuldade encontrada foi a utilização desta biblioteca no ambiente de programação *Delphi*. Devido aos comandos do *OpenGL* serem implementados na linguagem *C* a migração destes comando entre as linguagens é lenta sendo perceptível ao usuário. Constatou-se também durante o desenvolvimento do trabalho que algumas funções da biblioteca não funcionavam no ambiente *Delphi*.

4.1 EXTENSÕES

Como extensões da ferramenta sugere-se:

a) implementar a função de impressão dos laudos criados pelo sistema de acordo com os padrões utilizados pela guarde de trânsito;
b) desenvolver e implementar funções no editor gráfico afim de melhorar a confecção dos desenhos, por exemplo as funções de copiar, colar, refletir, desfazer entre outras;
c) implementar uma rotina capaz de importar desenhos criados no *AutoCAD* para o sistema;
d) permitir a usuário inserir fotos do acidente no sistema;
e) migrar o sistema para a linguagem de programação C afim de aprimorar a velocidade de desenho dos objetos.
f) gravar em áudio as declarações dos envolvidos
g) cadastrar a posição do sol no momento do acidente

REFERÊNCIAS BIBLIOGRÁFICAS

AZEVEDO, Eduardo; CONCI, Aura. **Computação gráfica**: teoria e prática. Rio de Janeiro: Elsevier, 2003.

BECKER, Odete Brancher. **[Entrevista realizada em 06 de novembro de 2006, no SETERB].** 2006

BECKER, Odete Brancher. **[Entrevista realizada em 10 de agosto de 2005, no Departamento de Trânsito de Blumenau (DETRAN)].** 2005.

BERTHOLDI, Geferson. **Editor gráfico de ruas para o sistema de controle de tráfego de automóveis em uma malha rodoviária urbana**. 2004.55 f, il. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

BERTOLDI, José Wilson. **[Entrevista realizada em 06 de novembro de 2006, no SETERB].** 2006

CANTU, Marco. **Dominando o Delphi 6: a bíblia.** São Paulo : Makron Books, 2002. Traducão de: Mastering Delphi 6.

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

COMPUTAÇÃO gráfica. In: WIKIPEDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2006. Disponível em: <http://pt.wikipedia.org/wiki/Computa%C3%A7%C3%A3o_gr%C3%A1fica>. Acesso em: 11 de outubro 2006.

COMPUTAÇÃO móvel. In: WIKIPEDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2006. Disponível em: < http://pt.wikipedia.org/wiki/Comp_Movel>. Acesso em: 11 de outubro 2006.

COUGO, Paulo Sergio. **Modelagem conceitual e projeto de bancos de dados.** Sao Paulo : Campus, 1997.

DALFOVO, Oscar. **Sistemas de informação**: estudos e casos. Blumenau: Acadêmica, 2004.

DENIS, Michel; ANDRE, Regis. **O desenho assistido por computador CAD.** Sao Paulo : Aleph, 1992. Traducao de: Le dessin assiste par ordinateur.

FREIRE, Jocemar José. **Simulação do controle de tráfego de automóveis em uma malha rodoviária urbana.** , 2004. 47 p, il.. Trabalho de Conclusão de Curso (Bacharelado 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 em uma malha rodoviária urbana: versão 2.0.** 2006.72 f, il. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

GOMES, Jonas; VELHO, Luiz. **Fundamentos da computação gráfica**. Rio de Janeiro : IMPA, 2003.

MATEUS, Geraldo Robson; LOUREIRO, Antônio Alfredo F., **Introdução à Computação Móvel**, 11a Escola de Computação, COPPE/Sistemas, NCE/UFRJ, 1998.

MENEZES, Eduardo Diatahy Bezerra de. **Princípios de análise e projeto de sistemas com UML**. Rio de Janeiro : Campus, 2002.

PACHECO, Osvaldo. **[Entrevista realizada em 13 de setembro de 2005, no Departamento de Trânsito de Blumenau (DETRAN)].** 2005.

RHENIUS, Jean Paul. **[Entrevista realizada em 13 de setembro de 2005, no Departamento de Trânsito de Blumenau (DETRAN)].** 2005.

SETERB. **Guarda de trânsito**. Blumenau. Disponível em: <http://www.seterb.com.br/transito03.htm>. Acesso em: 10 out. 2006.

SOUZA, Silvandro José. **[Entrevista realizada em 15 de agosto de 2005, no Departamento de Trânsito de Blumenau (DETRAN)].** 2005.

WITTE, Tommy. **Sistema de automação de recuperadoras de pneus com a coleta de dados via Palm Top**. 2005.69 f, il. Trabalho de conclusão de curso - Universidade Regional de Blumenau, Curso de Sistemas de Informação, Blumenau.

WOO, Mason; OPENGL ARCHITECTURE REVIEW BOARD. **OpenGL programming guide: the official guide to learning OpenGL, version 1.2.** 3rd ed. Boston : Addison Wesley, 1999.

ANEXO A – Formulários

A figura 45 apresenta o formulário usado atualmente pela guarda de trânsito para coletar as informações referentes ao acidente.

BOLETIM DE OCORRÊNCIA DE ACIDENTES DE TRÂNSITO

FORMULÁRIO Nº_____

INFORMAÇÕES GERAIS DO ACIDENTE

DATA:___/___/___ HORÁRIO DA OCORRÊNCIA:___:___ FASE DO DIA: ☐ DIA ☐ NOITE ☐ MADRUGADA

LOCAL:_____ MUNICÍPIO: BLUMENAU UF: SC

PONTO REF.:_____

TIPOS DE ACIDENTE

COLISÃO ☐ FRONTAL ☐ TRASEIRA ABALROAMENTO ☐ LONGITUDINAL ☐ TRANSVERSAL

ATROPELAMENTO ☐ PESSOA ☐ ANIMAL

CHOQUE ☐ ÁRVORE ☐ POSTE ☐ CERCA ☐ CANTEIRO DIVISOR ☐ VEÍCULO ☐ MURO ☐ BARRANCO ☐ CASA ☐ MEIO FIO

DIA
☐ DOMINGO
☐ SEGUNDA
☐ TERÇA
☐ QUARTA
☐ QUINTA
☐ SEXTA
☐ SÁBADO

PRECIPITAÇÃO SOLO ☐ RIBANCEIRA

☐ CAPOTAMENTO ☐ TOMBAMENTO ☐ ENGAVETAMENTO

SEVERIDADE DO ACIDENTE
☐ DANOS MATERIAIS
☐ COM FERIDO
☐ COM FERIDO PEDESTRE
☐ COM VÍTIMA FATAL

QUANTIDADE DE VEÍCULOS
___ COM DANOS
___ SEM DANOS
___ EVADIDOS

QUANTIDADE DE VÍTIMAS
___ CONDUTORES FERIDOS
___ CONDUTORES MORTOS
___ PASSAGEIROS FERIDOS
___ PASSAGEIROS MORTOS
___ PEDESTRES FERIDOS
___ PEDESTRES MORTOS

VIA - MEIO AMBIENTE

SUPERFÍCIE DA PISTA	☐ SECA	☐ MOLHADA				
CONDIÇÕES DA PISTA	☐ OLEOSA	☐ INUNDADA	☐ ENLAMEADA	☐ BOA	☐ DANIFICADA	LARGURA_____
TIPO DE PAVIMENTO	☐ ASFALTO	☐ CONCRETO	☐ PARALELEPÍPEDO	☐ CASCALHO	☐ BRITA	
	☐ ARENOSO	☐ LAJOTA	☐ TERRA	☐ OUTROS		
MEIO FIO	☐ SIM	☐ NÃO	☐ LADO DIREITO	☐ LADO ESQUERDO		
ACOSTAMENTO	☐ SIM	☐ NÃO	☐ LADO DIREITO	☐ LADO ESQUERDO		
PASSEIO	☐ SIM	☐ NÃO	☐ LADO DIREITO	☐ LADO ESQUERDO		
MÃO	☐ ÚNICA	☐ INGLESA	☐ DUPLA			
CANTEIRO DIVISOR	☐ SIM	☐ NÃO				
ILHA DE SEGURANÇA	☐ SIM	☐ NÃO				
INTERSECÇÕES	☐ CRUZAMENTO	☐ BIFURCAÇÃO	☐ TREVO			
ENTRONCAMENTO CADASTRADO	☐ SIM CÓD: _____		☐ NÃO NOME:			
ALINHAMENTO	☐ RETA	☐ RETA ACLIVE	☐ RETA DECLIVE	☐ CURVA	☐ CURVA ACLIVE	☐ CURVA DECLIVE
	☐ TÚNEL	☐ VIADUTO	☐ PONTE			

COND. SINALIZAÇÃO SEMAFÓRICA
☐ NORMAL
☐ INTERMITENTE
☐ COM DEFEITO
☐ DESLIGADO
☐ INEXISTENTE
☐ FOTOSSENSOR
☐ MEDIDOR FIXO

COND. SINALIZAÇÃO VERTICAL
☐ BOA
☐ DEFICIENTE
☐ INEXISTENTE
☐ FOTOSPEED
☐ ESTAC. PROIBIDO ☐ LADO DIREITO
☐ LADO ESQUERDO ☐ AMBOS OS LADOS

COND. SINALIZAÇÃO HORIZONTAL
☐ BOA
☐ DEFICIENTE
☐ INEXISTENTE

FAIXA DE PEDESTRE CADASTRADA
☐ SIM CÓDIGO:_____
☐ NÃO NOME:_____

CONDIÇÕES DO TEMPO
☐ BOM
☐ CHUVA

PISTA EM OBRAS
☐ SIM
☐ NÃO

VISIBILIDADE
☐ BOA
☐ REGULAR
☐ RUIM MOTIVO: ☐ NEBLINA ☐ FUMAÇA ☐ POEIRA ☐ CHUVA

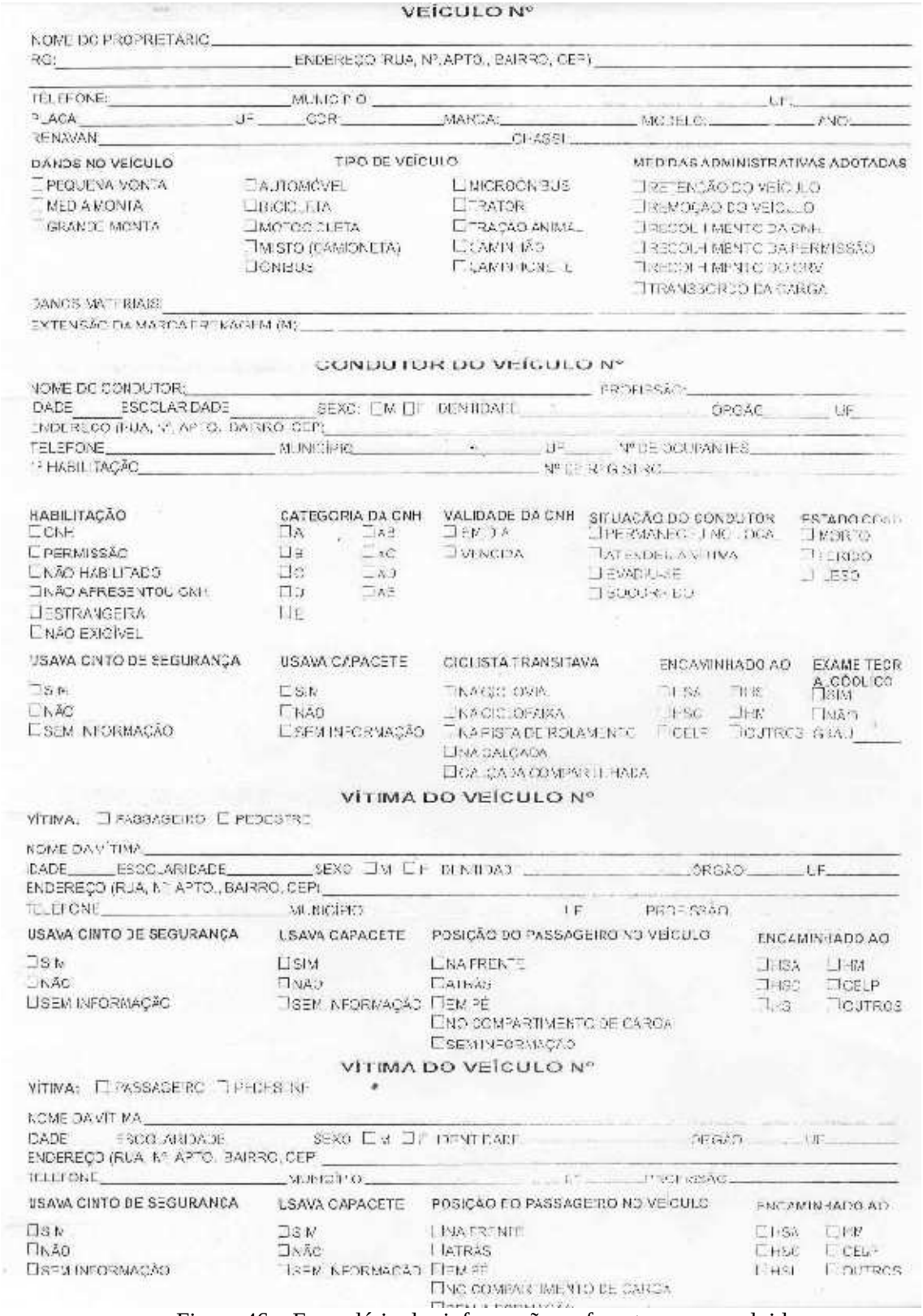
VELOCIDADE MÁXIMA PERMITIDA: ☐ 60 KM/H ☐ 60 KM/H ☐ 50 KM/H ☐ 40 KM/H ☐ OUTRAS _____

Obs.:_____

Figura 45 – Formulário das informações referentes ao acidente

A figura 46 apresenta o formulário usado atualmente pela guarda de trânsito para

coletar as informações referentes ao veículo, proprietário, condutor e vítimas.



VEÍCULO Nº

NOME DO PROPRIETÁRIO:
RG: ENDEREÇO (RUA, Nº, APTO., BAIRRO, CEP):
TELEFONE: MUNICÍPIO: UF:
PLACA: UF: COR: MARCA: MODELO: ANO:
RENAVAN: CHASSI:

DANOS NO VEÍCULO
☐ PEQUENA MONTA
☐ MÉDIA MONTA
☐ GRANDE MONTA

TIPO DE VEÍCULO
☐ AUTOMÓVEL
☐ BICICLETA
☐ MOTOCICLETA
☐ MISTO (CAMIONETA)
☐ ÔNIBUS
☐ MICROÔNIBUS
☐ TRATOR
☐ TRAÇÃO ANIMAL
☐ CAMINHÃO
☐ CAMINHONETE

MEDIDAS ADMINISTRATIVAS ADOTADAS
☐ RETENÇÃO DO VEÍCULO
☐ REMOÇÃO DO VEÍCULO
☐ RECOLHIMENTO DA CNH
☐ RECOLHIMENTO DA PERMISSÃO
☐ RECOLHIMENTO DO CRLV
☐ TRANSBORDO DA CARGA

DANOS MATERIAIS:
EXTENSÃO DA MARCA DE FRENAGEM (M):

CONDUTOR DO VEÍCULO Nº

NOME DO CONDUTOR: PROFISSÃO:
IDADE ESCOLARIDADE SEXO: ☐ M ☐ F IDENTIDADE ÓRGÃO UF
ENDEREÇO (RUA, Nº, APTO., BAIRRO, CEP):
TELEFONE MUNICÍPIO UF Nº DE OCUPANTES
Nº HABILITAÇÃO Nº DE REGISTRO:

HABILITAÇÃO
☐ CNH
☐ PERMISSÃO
☐ NÃO HABILITADO
☐ NÃO APRESENTOU CNH
☐ ESTRANGEIRA
☐ NÃO EXIGÍVEL

CATEGORIA DA CNH
☐ A ☐ AB
☐ B ☐ AC
☐ C ☐ AD
☐ D ☐ AE
☐ E

VALIDADE DA CNH
☐ VÁLIDA
☐ VENCIDA

SITUAÇÃO DO CONDUTOR
☐ PERMANECEU NO LOCAL
☐ ATENDEU A VÍTIMA
☐ EVADIU-SE
☐ SOCORREU

ESTADO COND.
☐ MORTO
☐ FERIDO
☐ ILESO

USAVA CINTO DE SEGURANÇA
☐ SIM
☐ NÃO
☐ SEM INFORMAÇÃO

USAVA CAPACETE
☐ SIM
☐ NÃO
☐ SEM INFORMAÇÃO

CICLISTA TRANSITAVA
☐ NA CICLOVIA
☐ NA CICLOFAIXA
☐ NA FAIXA DE ROLAMENTO
☐ NA CALÇADA
☐ CALÇADA COMPARTILHADA

ENCAMINHADO AO
☐ HSA ☐ HM
☐ HSC ☐ HR
☐ CELP ☐ OUTROS

EXAME TEOR ALCÓOLICO
☐ SIM
☐ NÃO
G/A.) _____

VÍTIMA DO VEÍCULO Nº

VÍTIMA: ☐ PASSAGEIRO ☐ PEDESTRE

NOME DA VÍTIMA:
IDADE ESCOLARIDADE SEXO ☐ M ☐ F IDENTIDADE ÓRGÃO UF
ENDEREÇO (RUA, Nº, APTO., BAIRRO, CEP):
TELEFONE MUNICÍPIO UF PROFISSÃO

USAVA CINTO DE SEGURANÇA
☐ SIM
☐ NÃO
☐ SEM INFORMAÇÃO

USAVA CAPACETE
☐ SIM
☐ NÃO
☐ SEM INFORMAÇÃO

POSIÇÃO DO PASSAGEIRO NO VEÍCULO
☐ NA FRENTE
☐ ATRÁS
☐ EM PÉ
☐ NO COMPARTIMENTO DE CARGA
☐ SEM INFORMAÇÃO

ENCAMINHADO AO
☐ HSA ☐ HM
☐ HSC ☐ CELP
☐ HR ☐ OUTROS

VÍTIMA DO VEÍCULO Nº

VÍTIMA: ☐ PASSAGEIRO ☐ PEDESTRE

NOME DA VÍTIMA:
IDADE ESCOLARIDADE SEXO ☐ M ☐ F IDENTIDADE ÓRGÃO UF
ENDEREÇO (RUA, Nº, APTO., BAIRRO, CEP):
TELEFONE MUNICÍPIO UF PROFISSÃO

USAVA CINTO DE SEGURANÇA
☐ SIM
☐ NÃO
☐ SEM INFORMAÇÃO

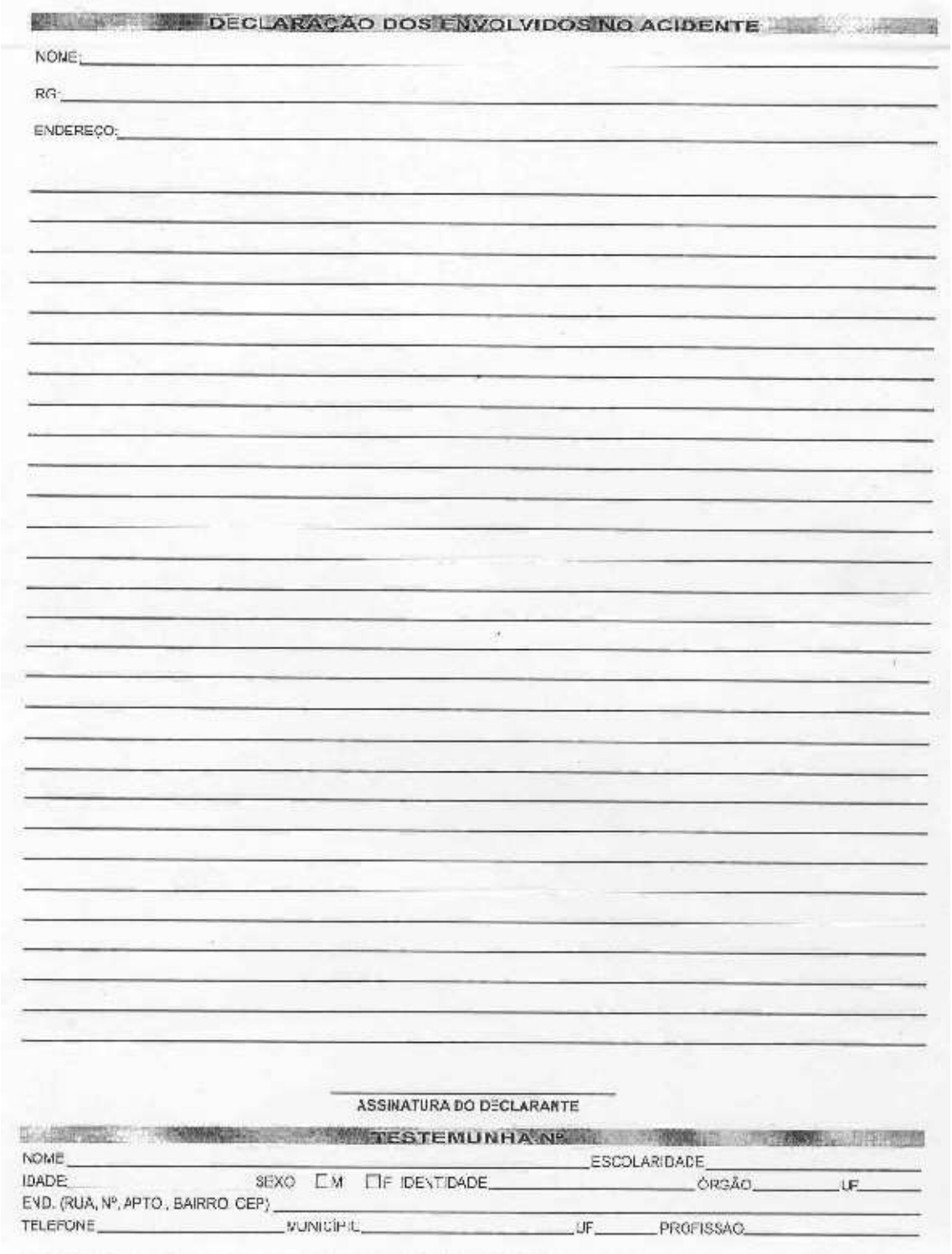
USAVA CAPACETE
☐ SIM
☐ NÃO
☐ SEM INFORMAÇÃO

POSIÇÃO DO PASSAGEIRO NO VEÍCULO
☐ NA FRENTE
☐ ATRÁS
☐ EM PÉ
☐ NO COMPARTIMENTO DE CARGA
☐ SEM INFORMAÇÃO

ENCAMINHADO AO
☐ HSA ☐ HM
☐ HSC ☐ CELP
☐ HR ☐ OUTROS

Figura 46 – Formulário das informações referentes aos envolvidos

A figura 47 apresenta o formulário usado atualmente pela guarda de trânsito para escrever as declarações dos envolvidos.



DECLARAÇÃO DOS ENVOLVIDOS NO ACIDENTE

NOME:

RG:

ENDEREÇO:

ASSINATURA DO DECLARANTE

TESTEMUNHA Nº

NOME ESCOLARIDADE

IDADE SEXO ☐M ☐F IDENTIDADE ÓRGÃO UF

END. (RUA, Nº, APTO., BAIRRO CEP)

TELEFONE MUNICÍPIO UF PROFISSÃO

Figura 47 – Formulário das informações referentes a declaração dos envolvidos

ANEXO B – Croqui

A figura 48 apresenta o desenho de um croqui feito manualmente pela guarda de trânsito. Na parte superior direita é possível observar o número do croqui.

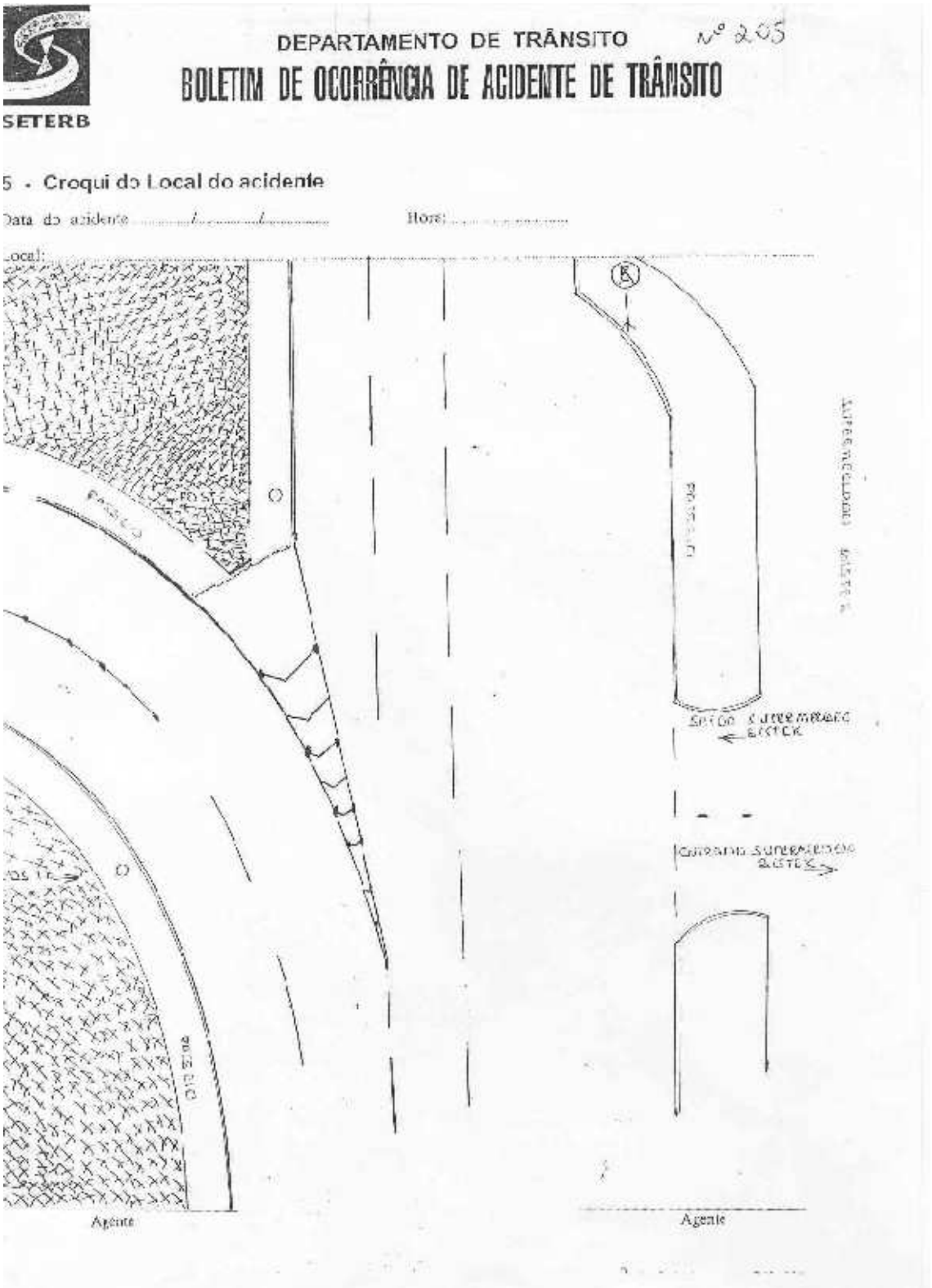


Figura 48 – Croqui desenvolvido manualmente pela guarda de trânsito

ANEXO C – Tabela de busca dos croquis

A figura 49 apresenta a tabela utilizada pelos agentes para localizar os croquis nas pastas onde estão armazenados.

RELAÇÃO DE CROQUI		
AGROLÂNDIA	MANOEL P. DA SILVA NETO	402
ALAMEDA DUQUE DE CAXIAS	BEC	179
ALAMEDA DUQUE DE CAXIAS	CEARÁ	112
ALAMEDA DUQUE DE CAXIAS	CELESC	475
ALAMEDA RIO BRANCO	AMAPÁ	95
ALAMEDA RIO BRANCO	CEL VIDAL RAMOS	49
ALAMEDA RIO BRANCO	DR. LUIZ DE FREITAS MELRO	30
ALAMEDA RIO BRANCO	DR.FREDERICO G.BUSCH	248
ALAMEDA RIO BRANCO	ENGº RODOLFO FERRAZ	101
ALAMEDA RIO BRANCO	FARMACEUTICO JOÃO MEDEIROS	53
ALAMEDA RIO BRANCO	FREDERICO BUSCH	193
ALAMEDA RIO BRANCO	LAURO MUELLER	99
ALAMEDA RIO BRANCO	PANDIÁ CALÓGERAS	318
ALAMEDA RIO BRANCO	PASTOR STUTZER	204
ALAMEDA RIO BRANCO	SEBASTIÃO CRUZ	62
ALBERTO STEIN	PROEB	245
ALBERTO STEIN	ZENAIDE DOS SANTOS	264
ALBERTO STEIN	ALMIRANTE TAMANDARÉ	40
ALBERTO STEIN	ATALIBA	114
ALMIRANTE BARROSO	FREI GABRIEL ZIMMER	299
ALMIRANTE BARROSO	GUSTAVO SALINGER	161
ALMIRANTE BARROSO	THEODORO HOLTRUP	162
ALMIRANTE BARROSO	FELIPE CAMARÃO	327
ALMIRANTE TAMANDARE	SETE SETEMBRO	328
ALMIRANTE TAMANDARE	TOBIAS BARRETO	329
ALMIRANTE TAMANDARÉ	10.º BPM	350
ALMIRANTE TAMANDARÉ	JOSÉ BOITEUX	382
ALMIRANTE TAMANDARÉ	MARECHAL DEODORO	196
ALMIRANTE TAMANDARÉ	MARIANA BRONNEMANN	329
ALMIRANTE TAMANDARÉ	PROX. PRAÇA ESTUDANTE	364
ALMIRANTE TAMANDARÉ	BENJAMIN CONSTANT	57
ALMIRANTE TAMANDARÉ	CURITIBANOS	426
ALMIRANTE TAMANDERE	ALBERTO STEIN	427
ALWIN SCHRADER	CEÁRA	376
ALWIN SCHRADER	XV DE NOVEMBRO	39
AMADEU DA LUZ	GETÚLIO VARGAS	92
AMAZONAS	23.º BI	207
AMAZONAS	23.º BI	218
AMAZONAS	ACRE	307
AMAZONAS	ADOLFO FREYGANG	311
AMAZONAS	ANGELONI	321
AMAZONAS	ANITA GARIBALDE	397
AMAZONAS	BISTEK	205
AMAZONAS	CARL. HEINZ BUECHLER	273
AMAZONAS	CENTENÁRIO	305
AMAZONAS	DA GLORIA	17
AMAZONAS	ENTRADA SUB TENENTE	72
AMAZONAS	GERTRUDE METZGER	235

Figura 49 – Tabela de busca dos croquis