**UNIVERSIDADE FEDERAL FLUMINENSE**

**INSTITUTO DE COMPUTAÇÃO**

**DEPARTAMENTO DE CIÊNCIA DE COMPUTAÇÃO**

**MARCELLO WILLIANS MESSINA RIBEIRO**

# USO DE JAVAFX PARA DESENVOLVIMENTO DE UM PROTÓTIPO BASEADO EM "CAR-FOLLOWING" PARA FINS EDUCACIONAIS

Niterói
2010

**MARCELLO WILLIANS MESSINA RIBEIRO**

# USO DE JAVAFX PARA DESENVOLVIMENTO DE UM PROTÓTIPO BASEADO EM "CAR-FOLLOWING" PARA FINS EDUCACIONAIS

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

_____

Prof. Mauricio Kischinhevsky, D.Sc. / IC - UFF
(Orientador)

_____

Prof. Marco Antonio Silva Ramos, D.Sc. / IC - UFF

_____

Prof. Esteban Walter Gonzalez Clua, D.Sc. / IC - UFF

Niterói
2010

**Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF**

R484 Ribeiro, Marcello Willians Messina
Uso de JavaFX para desenvolvimento de um protótipo baseado em "Car-Following" para fins educacionais / Marcello Willians Messina Ribeiro. – Niterói, RJ : [s.n.], 2010.
45 f.

Orientador: Mauricio Kischinhevsky.
Trabalho (Conclusão de Curso) – Departamento de Computação, Universidade Federal Fluminense, 2010.

1. Java (Linguagem de programação de computador). 2. Trânsito. 3. Educação para segurança no trânsito. 4. Veículo automotivo. 5. Tráfego urbano. I. Título.

CDD 005.133

## Agradecimentos

Aos meus pais, João Amaro Ribeiro e Vera Lucia Messina da Silva, minha avó e a toda minha família que, com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa de minha vida.

À minha namorada Danielle Ferreira, que soube compreender o tempo que eu tive que passar longe dela para concluir este trabalho e, mesmo assim, me deu todo suporte que precisei.

Ao professor e orientador Mauricio por seu apoio e inspiração no amadurecimento dos meus conhecimentos e conceitos que me levaram a execução e conclusão desta monografia.

À professora e coordenadora do Curso de Ciência da Computação, Teresa Cristina, pelo convívio, pelo apoio, pela compreensão nos momentos difíceis e pela amizade.

A todos os professores da UFF, que foram tão importantes na minha vida acadêmica e no desenvolvimento desta monografia.

Aos amigos e colegas, pelo incentivo e pelo apoio constantes.

## Resumo

Este trabalho tem por objetivo implementar um simulador de trânsito, utilizando a tecnologia JavaFX, capaz de reproduzir as principais situações vivenciadas no dia-a-dia do trânsito comum e que, ao mesmo tempo, possuísse uma interface agradável e atraente.

Esse primeiro protótipo com fins educacionais consegue ser o ponto de partida dessa idéia. Este tema é interessante, pois os acidentes de trânsito fazem parte do cotidiano das cidades, pondo em risco a segurança das pessoas [13]. Ações educativas, principalmente com alvo no público infanto-juvenil, são a melhor maneira de conscientização para que tenhamos uma trânsito cada vez mais seguro no futuro.

A tecnologia tem que ser usada em prol desse tipo de ação educacional e de cidadania. Esperamos que esse trabalho possa contribuir com tal objetivo [12].



**Palavras Chave:** JavaFX, Trânsito, Educação para segurança no trânsito, Veículo automotivo, Tráfego urbano

## Abstract

This paper aims to implement a traffic simulator, using JavaFX technology, able to reproduce the main situations experienced in the daily traffic and, at the same time, with a pleasant and attractive interface.

This first prototype for educational purposes can be the starting point of this idea. This theme is interesting, since traffic accidents are part of daily life of cities, threatening citizen's safety. Educational activities mainly targeted at children and teenagers are the best way to raise awareness so that we can have an increase in traffic safety in the future.

The technology has to promote social benefit and citizenship education. We hope this work can be a contribution to this goal.



**Keywords**: JavaFX, Traffic, Education for traffic safety, Vehicles, Urban traffic

**Sumário**

## Lista de Figuras

# 1 Introdução

O objetivo principal deste trabalho é implementar um protótipo de simulação de trânsito com fins educacionais [12] utilizando a tecnologia JavaFX. Como objetivo indireto e, por quê não, um desafio, podemos citar a obtenção de um conhecimento mais aprofundado desta linguagem de programação, que promete a criação de aplicações Web interativas e graficamente atraentes.

Definimos um conjunto de situações que podem ser observadas no dia-a-dia do trânsito e as transportamos para nosso ambiente de simulação. Dessa forma nos propusemos a dar início a este projeto com dois focos em mente:

1. Em primeiro lugar construir uma base robusta de software, utilizando modelos de fácil compreensão e expansão. Assim, com o poder que a orientação a objetos de Java propicia, tem-se um mecanismo simples para adicionar, de forma rápida e sem muito esforço, regras ainda não contempladas por este trabalho.

2. Em segundo lugar, já implementar no corpo desse ferramental, diversas situações observadas em nosso cotidiano no trânsito, servindo assim de incentivo a futuras expansões do projeto.

# 2 Escolha da tecnologia para o projeto

A escolha da linguagem Java se deu devido ao seu uso amplamente difundido e à enorme quantidade de frameworks de todos os tipos existentes, inclusive frameworks gráficos.

Inicialmente analisamos dois frameworks gratuitos em Java para a construção de jogos e animações em 2D, considerando a possibilidade de seu emprego: o PulpCore [4] e o GTGE[5]. O framework JMonkey [17], que possui uso bastante difundido na comunidade de desenvolvimento de jogos e simulações, também seria uma escolha interessante para a execução deste projeto. Contudo seu foco principal são aplicações 3D e de alta complexidade, o que foge do nosso objetivo para este projeto.

## 2.1 PulpCore

O PulpCore é um framework de código aberto (licença BSD) desenvolvido pela empresa Interactive Pulp [4] com a proposta de permitir que desenvolvedores Java criem jogos e animações para Internet. Para o princípio deste trabalho, ele foi a ferramenta utilizada. E obtivemos bons resultados. No entanto, ele se mostrou ainda muito ineficiente em termos de performance e com ausência de funções que facilitassem a execução das simulações. Sua estrutura é muito complexa e, para fazer coisas simples, era necessário utilizar muitas operações e chamadas a métodos do *framework*. Tínhamos que fazer o controle de posicionamentos dos veículos na tela, por exemplo, de forma manual.

Além disso, foram encontrados alguns *bugs* que dificultavam a evolução do trabalho e, como o projeto deste *framework* não vinha sendo atualizado de forma constante, optamos por pesquisar alternativas ao PulpCore.

## 2.2 GTGE

Posteriormente examinamos o GTGE como alternativa. Trata-se de um *framework* simples e leve, também de código aberto (LGPL), desenvolvido pela empresa Golden T

Studios [5]. A princípio, os atributos de facilidade de uso e de customização, sugeriram o seu emprego. Também chamou a atenção pela forma simples e leve que o sistema de renderização possuía. Mas, ao tentarmos utilizar funções mais avançadas (que encontrávamos no PulpCore), percebemos que o GTGE não as possuía, revelando-se ainda em estado muito inicial de desenvolvimento. Vale ressaltar ainda que o projeto do GTGE está sem novas atualizações desde 2005. Pensamos em combinar o GTGE com o PulpCore, mas devido à já citada complexidade da estrutura do segundo, o resultado foi insatisfatório em termos de performance.

## 2.3 JavaFX

Começamos, então, a vislumbrar a proposta de utilizar JavaFX. Apesar do que o nome possa aparentar, JavaFX não é um *framework* Java, ela é uma plataforma de desenvolvimento de *software.* Possui uma linguagem de programação chamada JavaFX Script que é bastante diferente, em termos de sintaxe, do Java, porém com suporte nativo aos seus tipos de dados (*Integer*, *String*, *Boolean*, etc) e inclusive capaz de herdar classes Java. Ela é basicamente uma linguagem de script desenvolvida pela Sun Microsystems especialmente para fazer animações para Internet.

Estudamos um pouco mais a fundo a linguagem e a documentação disponível e percebemos que é ainda uma linguagem em seu estado inicial de evolução. Contudo tem o suporte da SUN e previsões concretas de lançamento de novos releases. Levando isso em conta e assumindo o risco de aprender uma nova linguagem em pouco tempo para utilizá-la em um projeto dessa importância, optamos por escolher a tecnologia JavaFX para desenvolver o projeto.

# 3 Um pouco mais sobre JavaFX

JavaFX é uma plataforma-cliente para a criação de conteúdo e aplicativos para vários dispositivos. Desenvolvida para permitir a fácil criação e *deployment* de RIAs (*Rich Internet Applications*) com conteúdo e mídia interativa, a plataforma JavaFX garante que as aplicações tenham aparência e comportamento semelhantes em dispositivos e *browsers* diversos [18].

A plataforma JavaFX oferece liberdade e flexibilidade para criar conteúdo expressivo para dispositivos móveis, *desktops*, televisões e outros dispositivos de maior capacidade. Combina as melhores capacidades da plataforma Java e a funcionalidade de mídia abrangente e imersiva em um ambiente de desenvolvimento completo, abrangente e intuitivo [19].

Apresenta, também, um conjunto essencial de ferramentas e tecnologias necessário para que designers e desenvolvedores, colaborem, criem e implantem aplicativos com conteúdo expressivo para navegadores e *desktops.* [18].

A versão 1.0 foi lançada em dezembro de 2008, como foco na área dos *desktops* e web *applets*. A versão 1.1 chegou alguns meses depois, adicionado suporte ao uso em telefones. Em meados de 2009 a versão 1.2 ficou disponível, com suporte a novas ferramentas de interface com o usuário. As próximas edições prometem expandir a plataforma ainda mais, atingindo dispositivos de TV, players de *Blu-ray*, e até dispositivos de captura de vídeo, além de aprimoramentos na parte de *desktop* com suporte a controles de interface com usuário mais recentes [3].

A linguagem JavaFX Script nos permite criar interfaces com o usuário de forma simples e elegante através de uma sintaxe de programação declarativa. Ela também toma proveito de todo o poder de Java, já que permite que se instancie e use classes Java diretamente dentro do código JavaFX Script [1]. Os desenvolvedores podem usar qualquer biblioteca Java em um aplicativo JavaFX [19].

## 3.1 “HelloWorld” em JavaFX Script

A seguir, um exemplo simples de um programa “HelloWorld” em JavaFX Script:

```
import javafx.scene.Scene;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.scene.paint.Color;
import javafx.scene.text.TextAlignment;

Stage {
    title: "Hello World - JavaFX"
    width: 500
    height: 300
    scene: Scene {
        content: [
            Text {
                content: "Hello World!"
                x: 150
                y: 150
                font: Font {name: "Sans Serif" size: 36}
                fill: Color.DARKBLUE
                textAlignment: TextAlignment.CENTER
            }
        ]
    }
}
```

Para exibir qualquer objeto na tela, temos que criar uma janela. O *Stage* é o container de mais alto nível em JavaFX e é responsável por exibir a janela da aplicação, seja ela *desktop*, *mobile* ou um *applet*. Também tem a função de definir a característica da janela, como o título, posição, tamanho, etc.

Em seguida precisamos criar um objeto *Scene*, que é basicamente a área principal onde podemos colocar outros objetos que serão mostrados na tela (por exemplo, o objeto do tipo *Text* contendo o texto “Hello World!”) [20].

## 3.2 Tipos de dados e declaração de variáveis

A linguagem JavaFX Script permite que variáveis seja declaradas com seu tipo explícito. O tipo de uma variável é explicitado após seu nome e é separado deste por dois pontos ( : ). O tipo de uma variável determina quais valores poderão ser atribuídos a mesma. Por exemplo, não é permitido atribuir um valor *String* a uma variável *Integer*.

JavaFX Script também possui um sistema de inferência de tipos que permite que uma variável seja declarada sem um tipo explícito. O compilador irá inferir o tipo da variável a partir de sua inicialização ou do seu primeiro uso. Após o tipo de uma variável ser definido, este não pode ser alterado. Por exemplo, se uma variável foi inicializada com um número inteiro e, em seguida, atribuir uma *String* a essa mesma variável, será gerado um erro de compilação [1].

JavaFX Script possui várias categorias de tipos de dados: os primitivos, seqüências, funções e classes. Entre os tipos primitivos podemos citar: *Boolean, Integer, Number*, *String* e *Duration*, presentes já na versão 1.0 do JavaFX. Na versão 1.1, foram incluídos outros: *Character, Byte, Short, Long, Float* e *Double*. Estes últimos permitem que programas na linguagem JavaFX Script se integrem facilmente com bibliotecas escritas em Java [1].

Abaixo alguns exemplos de declaração de variveis:

- *var encontrouNaLista : Boolean;*
- *var quantidadeDeItens : Integer;*
- *var totalDaCompra : Float;*

Dos tipos citados, o tipo *Duration* talvez seja o único que necessite de uma explicação mais detalhada, visto que os outros já devem ser conhecidos dos programadores, tanto da linguagem Java, quanto de outras linguagens.

### 3.2.1 O tipo primitivo *Duration*

Este tipo representa uma quantidade de tempo. Valores do tipo *Duration* são usados no JavaFX em tarefas de animação. São compostos por uma parte decimal seguidos de uma unidade de tempo. O JavaFX Script suporta quatro unidades de tempo: hora (*h*), minuto (*m*), segundo (*s*) e milissegundo (*ms*). Abaixo alguns exemplos de valores válidos para o tipo *Duration* [1]:

- 0.1*h* (uma hora)
- 2*m* (dois minutos)

- 30*s* (trinta segundos)
- 1000*ms* (mil milissegundos)

### 3.2.2 Seqüências

Uma se*quence* na linguagem JavaFX Script representa uma lista ordenada de dados do mesmo tipo. O tipo de uma *sequence* é definido pelo tipo de seus elementos. A forma de declaração de uma *sequence* é semelhante a uma declaração de variável, porém adicionando um par de colchetes ( *[]* ) após a declaração do tipo do elemento [6]. A seguir alguns exemplos de declarações de variáveis:

- *var valoresValidos : Integer[];*
- *var cidadesBrasileiras : String[];*
- *var temposDaAnimacao : Duration[];*

JavaFX Script provê uma série de funções nativas para manipular *sequences:* Pode-se obter o tamanho da *sequence*; acessar um único elemento; acessar um intervalo de elemento consecutivos, ou um subconjunto de elementos não consecutivos que satisfaçam a algum critério; reverter a ordem dos elementos da *sequence;* inserir novos elementos; apagar elementos; apagar um intervalo de elementos; etc. Como podemos perceber, esse é um tipo de dado muito útil dentro do JavaFX Script.

### *3.2.3 Binding* de Dados

O sistema de *binding* de dados utilizado por JavaFX Script permite que qualquer variável seja "ligada" a uma expressão. Quando uma parte dessa expressão muda, a expressão é recalculada e o valor da variável atualizado. Esta funcionalidade é o centro da abordagem de JavaFX para desenvolvimento de interfaces gráficas, onde os atributos da interface são "ligados" ao objeto do modelo e a medida que o modelo sofre as alterações, elas são refletidas para a interface [1]. Abaixo alguns exemplos de como utilizar uma expressão *bind:*

- *var totalRecebido : Double = bind totalDinheiro + totalCheque;*

- *var p : Poimt = Point { x : bind valorDeX , y : bind valorDeY};*

No primeiro exemplo a variável "*totalRecebido"* terá seu valor recalculado toda vez que *"totalDinheiro"* ou *"totalCheque"* sofrer alguma alteração. No segundo exemplo, de forma semelhante, o objeto *"p"*, do tipo *Point*, será reavaliado sempre que *"valorDeX"* ou *"valorDeY"* for alterado. Note que, mesmo o *bind* sendo usado dentro de um construtor, ele reflete alterações no objeto, inclusive após o objeto ter sido instanciado.

### 3.2.4 Animações, *Timelines* e *Transitions*

Animar objetos com compõem a interface com o usuário é um dos pontos fortes de JavaFX, auxiliando na tarefa de oferecer uma experiência mais atraente para o público. A essência da animação consiste em alterar atributos dos objetos ao longo do tempo, sejam eles a posição na tela, a opacidade, o tamanho, a forma, etc.

A animação desses objetos envolve o uso da classe *Timeline*, localizada no pacote *javafx.animation.* Dependendo do tipo de animação desejada, podemos optar entre duas técnicas: o uso de *key frames* ou o uso de alguma subclasse de *javafx.animation.transition.Transition. Key frames* podem ser definidos como pontos de mudança em um *timelinne* [10] onde atributos do objeto a ser animado são alterados e é especificado em que intervalo de tempo essas alterações devem ser feitas.

Em algumas situações o uso de *key frames* pode se tornar muito trabalhoso, ou mesmo inviável, caso se queira fazer uma animação mais complexa. Neste caso podemos usar alguma das subclasses de *Transition.* Abaixo veremos algumas delas:

- *TranslateTransition* : Movimenta um objeto de uma posição para outra dento de um período de tempo;

- *RotateTransition* : Gira um objeto em um período de tempo;

- *PathTransition:* Movimenta um objeto ao longo de um caminho durante um período de tempo;

A transição do tipo *PathTransition* foi a mais utilizada e, sem dúvida, a mais importante para o desenvolvimento da simulação. Conforme explicado no próximo capítulo, as pistas presentes na simulação foram concebidas em um programa de editoração vetorial e importadas para o ambiente JavaFX como *paths.* A partir desse ponto, foi possível criar uma animação que, para cada veículo, realizava os movimentos de transição ao longo do *path*, criando o efeito dos veículos estarem percorrendo a pista.

# 4 Preparação do ambiente de desenvolvimento

## 4.1 Bibliotecas necessárias

Como estamos trabalhando em conjunto com a plataforma Java, necessitamos do JDK (Java Development Kit). Optamos por utilizar a versão mais recente até o momento, o JDK 6 e a versão 1.2 do JavaFX. Todas essas bibliotecas podem ser encontradas no site da Sun Microsystems.

## 4.2 A interface de desenvolvimento

O ambiente utilizado para desenvolver o projeto foi a IDE Netbeans 6.8 [16] com *plugin* nativo para JavaFX. Esta é uma ferramenta amplamente conhecida e utilizada pelo meio acadêmico e profissional, que exibe alto grau de qualidade.

Embora estejamos utilizando essa interface de desenvolvimento específica, o projeto não fica restrito à mesma. Futuras alterações podem ser feitas em qualquer IDE.

## 4.3 Versionamento do projeto

O serviço de versionamento da Google, o GoogleCode [15] foi escolhido para o controle de alterações do projeto, por utilizar o Subversion, ferramenta que atende às necessidades do projeto, além de ser uma solução gratuita. Vale ressaltar a integração existente entre o Netbeans e servidores do Subversion.

## 4.4 Editoração vetorial

Para representação das pistas e circuitos, utilizamos o software de desenho vetorial Inkscape 0.46 [14]. Com ele conseguimos representar graficamente cada faixa de um circuito (Figura 1) e, em seguida, salvar o arquivo no formato “SVG”, conforme mostra a Figura 2). Quando abrimos o arquivo salvo com esse formato em um editor de texto, conseguimos ver,

em meio às informações, uma seqüência de números e letras. Essa seqüência compõe um vetor que o JavaFX é capaz de importar e transformar em um *path* (ou caminho). É neste *path* que executamos a animação de cada veículo. Isso será explicado de forma mais detalhada nos próximos capítulos.

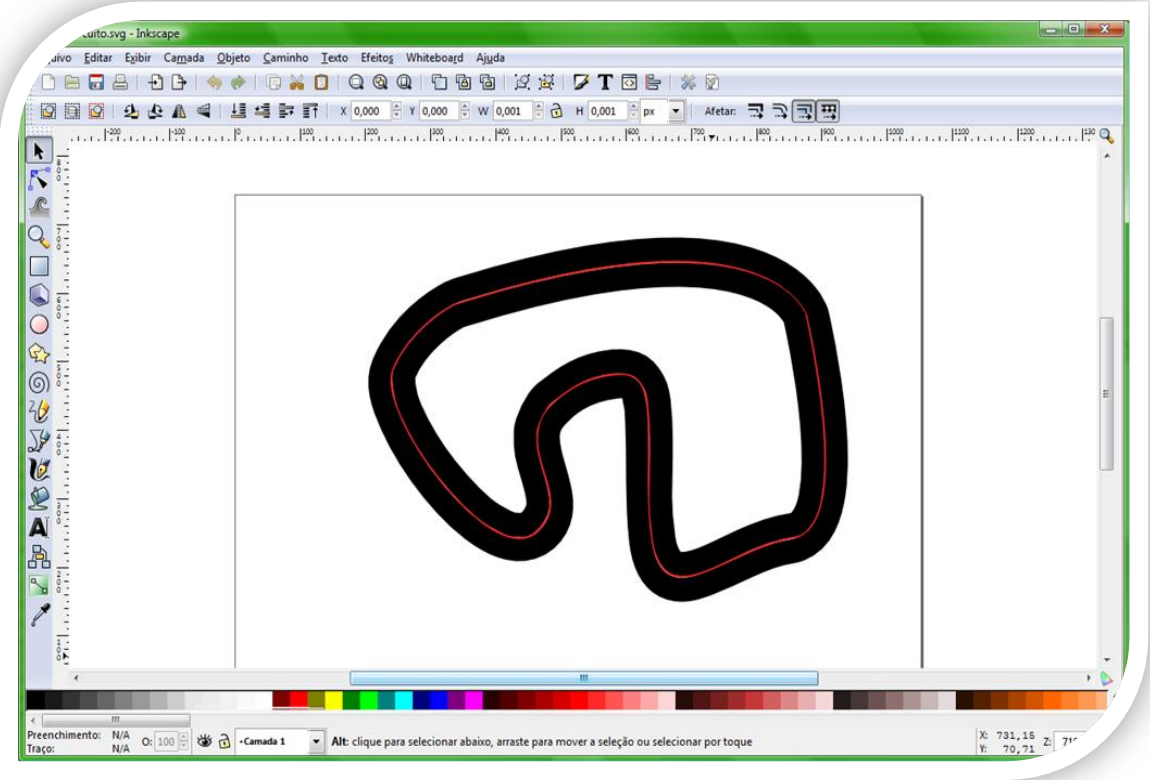


**Figura 1: Interface do *software* Inkscape com uma das pistas da simulação.**

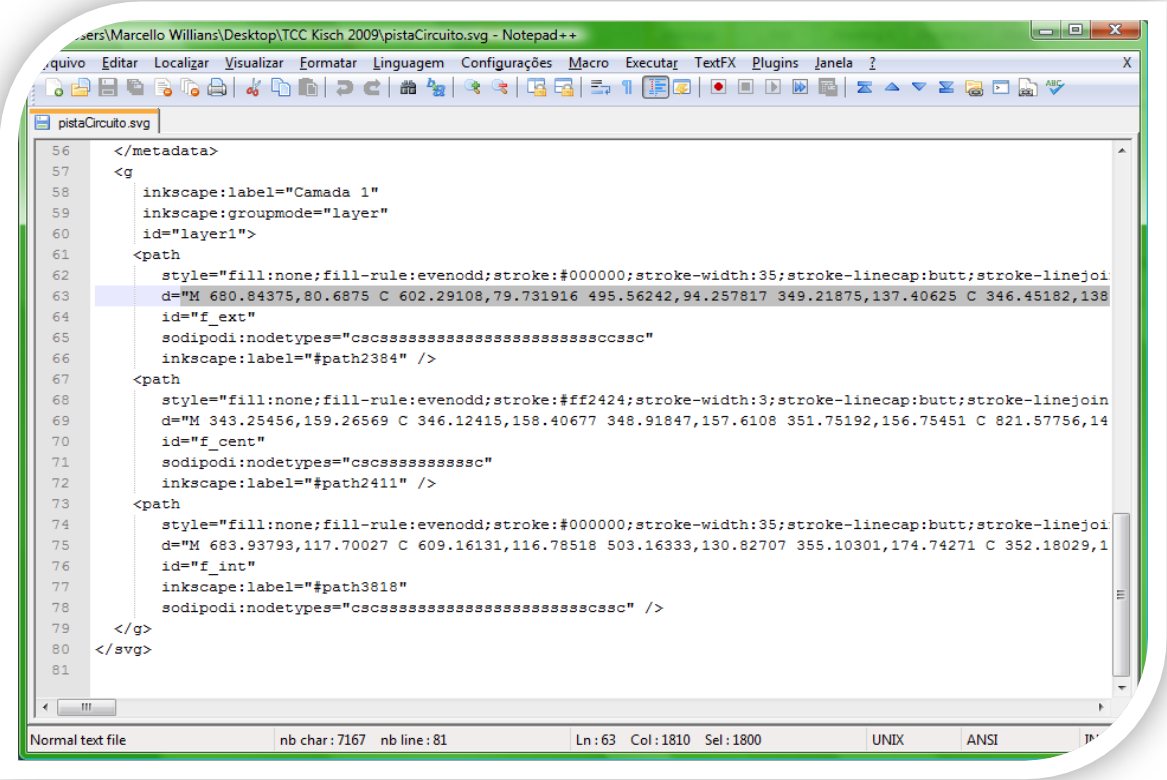


**Figura 2: Arquivo com extensão SVG da pista mostrada na Figura 1, aberto com o editor de texto Notepad++. Em destaque, um dos vetores que compõem os elementos das pistas.**

# 5 Modelagem da Simulação

A modelagem das classes que vamos utilizar é sem dúvida umas das fases mais importantes do projeto. Nesse momento temos que pensar em uma estrutura que atenda às necessidades de representação dos dados reais e, ao mesmo tempo, que não torne difícil ou ineficiente a tarefa de programar seus comportamentos. Além disso, é claro, temos que definir variáveis e atributos extras para adequação e controle por parte do framework.

Os requisitos estipulados para esta etapa inicial da simulação foram:

- A simulação deve conter duas pistas e dar suporte para a criação de outras;
- A simulação deve permitir a visualização e controle dos principais atributos de cada veículo nela contido;
- Cada pista as simulação deve conter duas faixas de rodagem e possibilitar a expansão para um número maior de faixas;
- Cada pista da simulação deve ter um conjunto de veículos próprio, independente de outra pista;
- Cada pista da simulação pode ter atributos específicos, como por exemplo uma região de subida;
- Cada veículo poderá ser do tipo “carro” ou “caminhão” e dependendo do seu tipo ele terá comportamentos distintos (velocidade máxima maior ou menor);
- Os veículos devem se manter na sua velocidade desejada. Se forem retidos por outros veículos mais lentos, ou trocam de faixa e seguem com a mesma velocidade, ou reduzem a velocidade para a mesma do veículo da frente;
- Cada veículo deve ter um componente que determine a probabilidade que ele irá trocar de faixa de rodagem quando estiver sendo retido por outro;

A seguir a Figura 3 mostra o diagrama de classes proposto para este projeto:

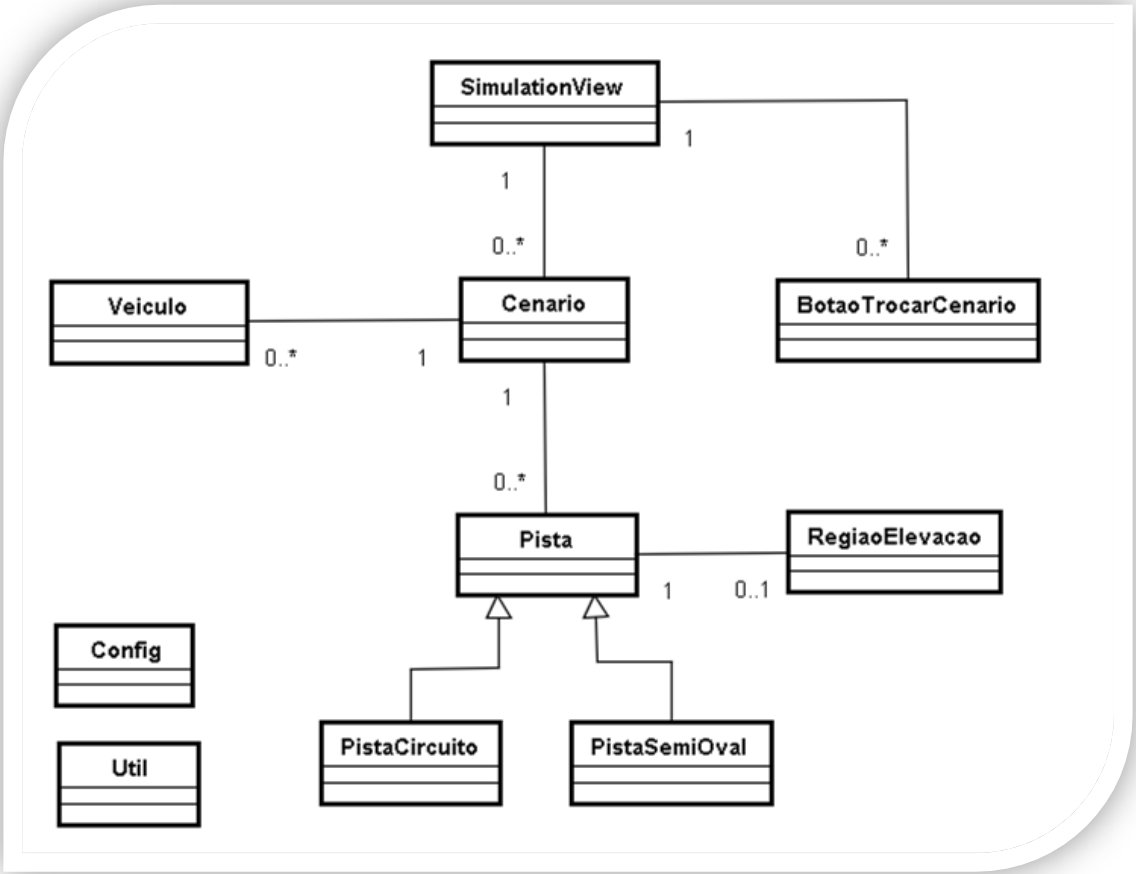


**Figura 3: Diagrama de classes do projeto.**

Agora vamos abordar com detalhes as principais classes e os atributos pertinentes a cada uma delas.

## 5.1 Modelagem das Pistas (Circuitos)

A modelagem proposta para as pistas no simulador se deu da seguinte forma: Primeiramente criamos uma super classe “Pista” com alguns atributos comuns a todos os tipos possíveis de pistas:

- *"faixasDeRodagemAnimadas"* : Uma lista que contém os "*paths*" criados no Inkscape que representa cada faixa de rodagem da pista. Em cada uma das faixas é onde ocorre a animação dos veículos.

- "*QUANTIDADE_DE_FAIXAS*" : É uma variável que deve ser preenchida pela classe que irá estender a super classe "*Pista"*, com a quantidade de faixas que a pista terá.

- *"regiaoElevacao"* : É a definição de uma área na pista que será considerada como uma região de subida. Quando os veículos passam por essa região, eles têm sua velocidade reduzida.

A seguir vamos detalhar as implementações de pistas que utilizamos na nossa simulação.

### 5.1.1 Classe "PistaSemiOval.fx"

Essa pista tem a proposta de ser uma pista simples, parecida com um circuito oval, porém com duas retas em cada lado. Ela possui poucos veículos, e seu objetivo é a adaptação do usuário ao ambiente de simulação, com as funções que ele pode desempenhar e o que esperar como retorno.

Essa pista possui uma região de elevação onde os veículos se comportam como se estivessem em uma subida. O comportamento de cada veículo é definido pelo seu tipo (tipo carro ou tipo caminhão). Por exemplo, em uma subida, os caminhões tendem a ter uma redução de velocidade mais acentuada que os carros.

### 5.1.2 Classe "PistaCircuito.fx"

Essa é a segunda pista proposta na simulação. Ela tem a característica de possuir mais curvas e não possui regiões de subida. A simulação nesta pista já se inicia com uma

quantidade maior de veículos, demonstrando melhor suas interações no decorrer da simulação.

## 5.2 Modelagem dos Veículos

### 5.2.1 Classe “Veiculo.fx”

No âmbito desta classe, optamos por realizar a separação dos seus atributos em dois grupos: o primeiro referente aos atributos pertinentes ao que se espera que um veículo deva ter (velocidade atual, velocidade máxima, dimensões do veículo, etc.); o segundo grupo de atributos é referente à necessidade de controle adicional para a realização da simulação (quantidade de colisões frontais, intenção de trocar de faixa ou não, etc.). A seguir vamos analisar cada um desses grupos.

Os atributos do primeiro grupo, ou seja, aqueles entendidos como pertinentes à representação real de um veículo, são:

- *“tipoVeiculo”* : Atributo que define se um veículo é um carro ou um caminhão;
- *“larguraVeiculo”* e *“comprimentoVeiculo”* : Atributos que correspondem às dimensões dos veículos;
- “*pista*” : Atributo que representa a pista na qual o veículo se encontra;
- “*faixaDeRodagemAtual*” : Atributo que representa o índice da faixa de rodagem, dentro da pista atual, em que o veículo está no momento;
- “*velocidadeAtual*” : É a velocidade atual do veículo na simulação. Esse valor nunca poderá ser negativo, ou seja, o veículo não poderá andar para trás;
- “*velocidadeEmPistaLivre*” : A velocidade do veículo em pista livre, ou seja, quando o veículo não está limitado por outros e pode atingir sua velocidade máxima desejada. De forma similar ao atributo “*velocidadeAtual*”, este também nunca poderá ser negativo;

- "*velocidadeEmSubida*" : A velocidade que o veículo terá caso ele esteja em uma região de subida na simulação;

- "*velocidadeAlvo*" : A velocidade "alvo" do veículo naquele momento. Baseado em sua velocidade atual, conseguimos determinar o quanto é preciso aumentar ou diminuir sua velocidade para atingir o valor desejado. Por exemplo, caso o veículo esteja atrás de outro mais lento, sua velocidade "alvo" será a mesma que a velocidade desse veículo. Se ele estiver em pista livre ou em subida será, respectivamente, uma das duas (mostradas imediatamente acima);

- "*deltaVelocidade*" : Equivale a uma aceleração/desaceleração que será imposta ao veículo para que este atinja sua velocidade alvo;

- "*distanciaDeSegurancaRelativaAVelocidadeAtual*" : Esse atributo é, na verdade, uma maneira de utilizarmos a fórmula que calcula a distância de segurança que o veículo deve manter para o da frente. A fórmula será mostrada e discutida com detalhes no tópico 0;

- *"indiceDeAgressividade"* : Valor que quantifica o quão agressivo o veículo se comportará naquele momento durante a simulação. Ou seja, o quanto ele vai tentar trocar de faixa quando estiver sendo retido por outro veículo mais lento. Esse índice varia de 0 (não tenta trocar de faixa) ao valor estipulado pela constante *"INDICE_AGRESSIVIDADE_MAXIMO"* (sempre tenta trocar de faixa);

Os atributos do segundo grupo, aqueles que se fazem necessários para controle adicional da simulação, são:

- "*idVeiculo*" : Número de identificação do veículo durante a simulação. Importante para tarefas de detecção de colisão e operações sobre um veículo selecionado pelo usuário (aumento / diminuição da velocidade, forçar troca de faixa, etc.);

- *"imagem"* : A imagem atribuída a cada veículo;

- *"animacaoDoVeiculo"* : Atributo do tipo *PathTransition* que representa a animação do veículo na simulação, ou seja, a movimentação do veículo na pista;

- *“rateRelativaVelocidadeAtual”* : Este atributo possui grande importância na simulação. Ele define a velocidade com que a animação do veículo será exibida na tela, em função da velocidade atual de cada veículo. Através desse atributo é que conseguimos dar a impressão de aceleração ou desaceleração dos veículos. Seu uso será mostrado mais a frente quando for discutido o momento de criação dos veículos. Por padrão seu valor a relação: *“velocidadeAtual”* / *“VELOCIDADE_MAXIMA_CARRO”;*

- *“quantidadeNaoColisoesFrontais”* : Quantidade de vezes seguidas que o veículo não “colidiu” frontalmente com outro. Utilizado em conjunto com a constante *“QUANTIDADE_MINIMA_NAO_COLISOES_SEGUIDAS”* para determinar com segurança que de fato o veículo não está “colidindo” **(cabe ressaltar que o termo “colisão” empregado neste trabalho, refere-se ao encontro das envoltórias dos veículos, umas com as outras, e não a uma colisão de fato de um veículo com outro);**

- *“quantidadeColisoesFrontais”* : Quantidade de vezes seguidas que o veículo “colidiu” frontalmente com outro;

- *“quantidadeColisoesLaterais”* e *“quantidadeNaoColisoesLaterais”* : Indica se o veículo está nesse momento com algum outro veículo em sua faixa lateral;

- *“quantidadeTentativasTrocarDeFaixa”* : Quantas vezes o veículo tentou trocar de faixa sem sucesso. É zerada assim que o veículo consegue trocar de faixa;

- *“trocarDeFaixaAssimQuePossivel”* : Atributo booleano que indica se o veículo deseja trocar de faixa de rodagem assim que possível;

- *“boundingBoxFrontal”* : Atributo que atua como um “bounding box” e verifica se uma colisão frontal está iminente, ou seja, se o veículo está muito próximo do veículo à sua frente. O tamanho desse “bounding box” é determinado pela fórmula de distância de segurança ou, mais especificamente, pelo atributo “*distanciaDeSegurancaRelativaAVelocidadeAtual”*;

- *“boundingBoxLateral”* : Atributo que atua como um “bounding box” e verifica se existe algum veículo na faixa de rodagem ao lado que impeça a troca de faixa;

- *“boundingBoxVeiculo”* : Atributo que atua como um “bounding box” para o veículo em si e verifica se uma colisão direta com o veículo está ocorrendo.

- *“idVeiculoSobreposto”* : Atributo utilizado para identificar um caso de erro na simulação: quando um veículo se sobrepõe a outro. Isso acontece normalmente no momento de adicionar um veículo novo à simulação. Nesse caso, o atributo armazena o *“idVeiculo”* do veículo com o qual está ocorrendo esse problema, para que sejam tomadas medidas de resolução explicadas mais a frente. O valor “-1” indica que não há sobreposição;

- *“ajustandoVelocidade”* : Este atributo define se o veículo está em processo de ajuste de velocidade (aceleração ou desaceleração);

- *“quantidadeLoopsParaFicarVisivel”* : Quantidade de *loops* para que um veículo recém adicionado apareça na simulação. Quando um veículo é adicionado ele não deve ser exibido instantaneamente, pois pode estar se “sobrepondo” a outro. Essa quantidade de *loops* “dá um tempo” para o veículo de ajustar na simulação. Por padrão seu valor é “10”;

A seguir iremos analisar as funções que compõem a classe veículo:

- *“setVelocidadeSemiRandomica(velocidade: Double)”* : Dado um valor de velocidade, essa função atribuiu um valor semi-randômico para a velocidade do veículo. Essa função é interessante, pois cada veículo terá uma velocidade diferente, porém dentro de uma faixa estipulada. A velocidade atribuída ao veículo estará no intervalo de 80% a 120% da velocidade desejada;

- *“setVelocidadeAtual(novaVelocidade : Number)”* : Função utilizada quando se deseja modificar a velocidade do veículo. Basicamente ela apenas atribui o valo passado ao atributo *“velocidadeAtual”* do veículo. Possui uma atuação diferente, entretanto, quando o nova velocidade é igual a zero: ao invés de atribuir zero a *“velocidadeAtual”*, ela atribui um valor muito pequeno a esta variável e, em seguida, pausa a simulação, causando o efeito do veículo estar parado na pista;

- *“inicializaComValoresDefault()”* : Esta função faz a inicialização padrão dos atributos do veículo;

- *“verificarTrocaDeFaixaDeRodagem()”* : Esta função verifica se o veículo deseja trocar de faixa naquele momento. Em caso positivo, tenta fazer a troca através da função *“trocarDeFaixaDeRodagem()”*. Caso a troca seja bem sucedida, reduz o *“indiceDeAgressividade”* do veículo a metade; Caso não tenha sucesso, incrementa *“indiceDeAgressividade”* do veículo, deixando-o mais propenso a tentar trocar de faixa no futuro;

- *“trocarDeFaixaDeRodagem() : Boolean”* : Esta função verifica se o veículo pode mudar de faixa naquele momento, ou seja, se ele não esta “colidindo” lateralmente com nenhum outro veículo. Caso seja possível ocorre a mudança de faixa. A animação do veículo ocorre sobre um path que esta ligado por "bind" ao valor do atributo “*faixaDeRodagemAtual*”. Alterando esse valor, o veículo passa automaticamente para a nova faixa;

- *“verificarAjusteDeVelocidade()”* : Esta função realiza o ajuste de velocidade do veículo, caso seja necessário. O decremento ou incremento necessário em sua velocidade atual depende da diferença das velocidades dele próprio e do veículo que está na frente. Esse diferença é representada por “*deltaVelocidade*”, que deve ser calculado apenas uma vez e utilizado até que se alcance a velocidade desejada;

- *“isDeltaVelocidadeValido() : Boolean”* : Verifica se o valor atual da variável “*deltaVelocidade*” é válido. Ou seja, se o carro deve desacelerar e delta tem que ser negativo; Se o carro deve acelerar, delta tem que ser positivo. Retorna *true* caso essas condições sejam satisfeitas, e *false* caso contrario;

- *“isDesacelerando()”* : Retorna *true* caso o veículo esteja desacelerando e *false* caso contrário;

- *“isAcelerando()”* : Retorna *true* caso o veículo esteja acelerando e *false* caso contrário;

- *“isAntecipandoColisao ()”* : Retorna *true* caso o veículo esteja sendo retido por outro a sua frente e *false* caso tenha pista livre;

- *“isNOTAntecipandoColisao ()”* : Retorna *true* caso *“quantidadeNaoColisoesFrontais”* seja maior que o valor definido em

"*QUANTIDADE_MINIMA_NAO_COLISOES_SEGUIDAS"*. Retorna *false* caso contrário. Esse artifício é utilizado para dar uma margem de segurança para não detecção de colisão.

## 5.3 Configurações da Simulação

### 5.3.1 Classe "Config.fx"

- "*SCREEN_WIDTH*" : Largura da tela da simulação. Por padrão seu valor é "1000" pixels;
- "*SCREEN_HEIGHT*" : Altura da tela da simulação. O valor padrão é de "800" pixels;
- "*VELOCIDADE_MAXIMA_CARRO*" : Define o limite de velocidade para os carros na simulação. Por padrão esse valor é de "80 Km/h";
- "*VELOCIDADE_MAXIMA_CAMINHAO*" : Define o limite de velocidade para os caminhões na simulação. Por padrão esse valor é de "60 Km/h";
- *"DIFERENCA_MINIMA_VELOCIDADE_AJUSTE_IMEDIATO"* : Valor mínimo da diferença entre a velocidade atual do veículo e a velocidade alvo para que o veículo atinja a velocidade alvo de forma imediata, ou seja, sem precisar reduzir/aumentar de forma gradual sua velocidade atual. Por padrão seu valor é "5Km/h";
- *"DELTA_MINIMO_PARA_PERMITIR_AJUSTE_PROGRESSIVO"* : Valor mínimo de Delta para permitir uma redução de velocidade progressiva. Valores negativos de Delta muito pequenos, menores que "-25" em geral, indicam que o veículo deve reduzir sua velocidade atual em um valor muito elevado. Não havendo espaço suficiente entre o veículo e o que se encontra na frente e, para evitar uma "colisão", adotamos uma redução imediata da velocidade atual do veículo, igualando-a com a velocidade alvo. Por padrão o valor dessa constante é "-25Km/h";

- *"QUANTIDADE_LOOPS_DESACELERACAO"* : Quantidade de *loops* da simulação que será conferida ao veículo para ajustar sua velocidade em caso de desaceleração. Essa constante busca simular o tempo que o veículo permanece em processo de frenagem até que sua velocidade atinja o nível desejado. Por padrão seu valor é "3".

- *"QUANTIDADE_LOOPS_ACELERACAO"* : Quantidade de *loops* da simulação que será conferida ao veículo para ajustar sua velocidade em caso de aceleração. Essa constante busca simular o tempo que o veículo permanece em processo de aceleração até que sua velocidade atinja o nível desejado. Por padrão seu valor é "10".

- *"EXIBIR_BOUNDING_BOX_DETECTAR_COLISAO_FRONTAL"* : Variável booleana que define se o Bounding Box de colisão frontal é exibido ou não durante a simulação. Por padrão o Bounding Box é exibido;

- *"EXIBIR_BOUNDING_BOX_DETECTAR_COLISAO_LATERAL"* : Variável booleana que define se o Bounding Box de colisão lateral (usado para troca de faixa de rodagem) é exibido ou não durante a simulação. Por padrão o Bounding Box é exibido;

- *"EXIBIR_BOUNDING_BOX_VEICULO"* : Variável booleana que define se o Bounding Box do veículo é exibido ou não durante a simulação. Por padrão o Bounding Box é exibido;

- *"QUANTIDADE_MINIMA_NAO_COLISOES_SEGUIDAS"* : Variável que estipula uma quantidade mínima de "não colisões" seguidas para determinar de fato que um veículo não esta colidindo frontalmente com outro, ou seja, que o veículo está a uma distância segura do veículo da frente. Esse artifício foi utilizado, pois percebemos que em alguns momentos a "colisão" com o veículo da frente não era detectada de forma contínua, havendo um intervalo onde o veículo achava estar sem nenhum outro a sua frente. Com essa variável conseguimos resolver o problema, estipulando que para o veículo de fato não estar "colidindo" com outro, ele tem que ficar uma quantidade de *loops* (definida por essa variável) sem ser

detectada nenhuma “colisão” frontal. Experimentalmente definimos o valor “8” como sendo adequado;

- *“QUANTIDADE_COLISOES_SEGUIDAS_PARA_REDUZIR_VELOCIDADE”* : Quantidade de “colisões” seguidas mínima para reduzir a velocidade do veículo em uma porcentagem definida, quando este encontra-se retido por outro. Experimentalmente definimos o valor “30 “como sendo adequado;

- *“PORCENTAGEM _REDUCAO_VELOCIDADE”* : Porcentagem de redução de velocidade imposta ao veículo, quando atingir o valor definido pela constante *“QUANTIDADE_COLISOES_SEGUIDAS_PARA_REDUZIR_VELOCIDADE”.* Por padrão o valor deste atributo é “90%”;

- *“QUANTIDADE_DE_TENTATIVAS_TROCA_DE_FAIXA_PARA_INCREMENTAR_INDICE_AGRESSIVIDADE”* : Quantidade de mínima de tentativas de troca de faixa que o veículo deve atingir antes de aumentar o índice de agressividade. Experimentalmente definimos o valor “30” como sendo adequado;

- *“INDICE_AGRESSIVIDADE_MAXIMO”* : Valor máximo do índice de agressividade que o veículo pode atingir. Por padrão o valor dessa constante é “10”;

## 5.4 A lógica da Simulação

A simulação possui um *loop* principal definido na classe “Main.fx”, chamado *simulationLoop.* Esse *loop* é, na verdade, uma variável do tipo *TimeLine* que executa indefinidamente [7] [8] e tem com única atribuição: chamar o método *simulationUpdate()* da classe “SimulationCore.fx”.

A função *simulationUpdate()* é a responsável por realizar todas as operações de verificação e ajuste dos valores dos atributos, tanto da simulação quanto dos veículos. Sua forma de trabalho dá-se através de um *loop* que itera sobre uma lista contendo todos os veículos da simulação. Vamos analisar cada uma das operações realizadas por esta função, na ordem que são executadas na simulação:

1. *veiculoAtual.verificarTrocarDeFaixaDeRodagem()* : A chamada a esta função (discutida anteriormente) tem o objetivo de resolver, o mais cedo possível, as trocas de faixas necessárias para os veículos. A faixa de rodagem na qual um veículo se encontra é um dado muito importante utilizado pelas funções que são executadas a seguir, logo esta verificação tem que ser a primeira;

2. *verificarColisao(veiculoAtual)* : Esta é uma função da classe "SimulationCore.fx" e é talvez a mais importante e complexa de toda a simulação. É nela que são verificadas as "colisões" [9] com o *boundingBoxFrontal* e *boundingBoxVeiculo* ("colisões" com o *boundingBoxLateral* são analisadas sob demanda somente quando o veículo tenta trocar de faixa). Além disso, é feita também a verificação se o veículo está em uma região de subida ou em pista livre. Quando uma "colisão" é detectada, medidas são tomadas para, ou reduzir a velocidade do veículo a fim de igualá-la a do veículo da frente, ou para fazer com que o veículo troque de faixa de rodagem. No caso de redução de velocidade, apenas o valor de *deltaVelocidade* é calculado. A tarefa de reduzir a velocidade de fato fica a cargo da função *verificarAjusteDeVelocidade()*, examinada a seguir;

3. *veiculoAtual.verificarAjusteDeVelocidade()* : Esta função já foi discutida anteriormente, seu papel neste momento é determinar com base na velocidade atual, velocidade "alvo" e *deltaVelocidade* (calculado na etapa anterior) se um ajuste na velocidade do veículo é necessário ou não. Em caso positivo inicia-se o processo de aceleração ou desaceleração do veículo;

4. *verificarSeVeiculoNovoPodeSerExibido(veiculoAtual)* : Esta é uma função da classe "SimulationCore.fx" e verifica se existe algum veículo que ainda não está sendo exibido e tenta exibi-lo. Entretanto algumas verificações são feitas para garantir que, por exemplo, um veículo recém adicionado não apareça "em cima" de outro que já está na simulação.

Após estas quatro operações serem executadas para todos os veículos da simulação, o *TimeLine simulationLoop* chama outra vez o método *simulationUpdate()* e o processo recomeça.

# 6 Modelo para a Distância de Segurança

O modelo para a distância de segurança surgiu da necessidade de calcularmos o valor da distância que cada veículo deveria manter para o veículo da frente. Sabemos que essa distância possui relação com a velocidade atual do veículo e que quanto mais alta for a velocidade, maior deve ser o valor desta variável. Entretanto, a distância de segurança não deve aumentar indefinidamente. Um valor considerado razoável para um veículo se movendo a 80km/h, não precisa ser muito maior que o de um veículo a 100km/h; ou que de outro a 120km/h.

A partir dessa premissa, ficou claro que precisávamos utilizar uma função que apresentasse um crescimento inicial, mas depois estabilizasse em um valor máximo. A função escolhida foi a Tangente Hiperbólica (*tanh*). A figura abaixo exibe um gráfico desta função para valores de *x* no intervalo [-10, 10].

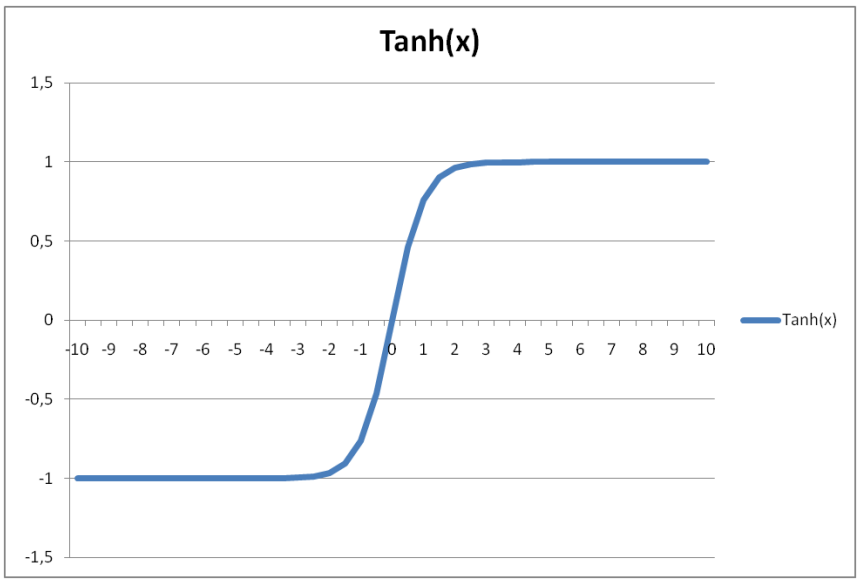


**Figura 4: Gráfico da função Tangente Hiperbólica (*tanh*).**

## 6.1 Fórmula para o cálculo da Distância de Segurança

Após análise da função Tangente Hiperbólica, chegamos à Fórmula 6.1[1], que calcula a distância de segurança em função da velocidade atual do veículo:

$$\frac{Dist.Seg.Maxima}{2} \times \left(1 + tanh\left(\left(\alpha \times \frac{velocidadeAtual}{velocidadeMaxima}\right) - \frac{\alpha}{2}\right)\right) \quad (6.1)$$

Vamos detalhar a Fórmula 6.1, examinando cada etapa de sua formação. Parte dessa fórmula, destacada abaixo em 6.2, mostra relação da *velocidadeAtual* com a *velocidadeMaxima*.

$$\frac{velocidadeAtual}{velocidadeMaxima} \quad (6.2)$$

Supondo que os veículos nunca ultrapassem a velocidade máxima permitida (80km/h), temos que a relação definida em 6.2 varia no intervalo [0,1].

Em 6.3 temos a inclusão da constante *alfa* nessa fórmula. Seu valor foi determinado de forma experimental com sendo igual a "3".

$$\left(\alpha \times \frac{velocidadeAtual}{velocidadeMaxima}\right) - \frac{\alpha}{2} \quad (6.3)$$

Desta maneira, os valores da equação representada por 6.3, irão variar no intervalo [-1.5,1.5]. Em 6.4, vemos como fica a aplicação da tangente hiperbólica na equação.

$$tanh\left(\left(\alpha \times \frac{velocidadeAtual}{velocidadeMaxima}\right) - \frac{\alpha}{2}\right) \quad (6.4)$$

[1] Veja a referência bibliográfica [11] para informações sobre como a Figura da Fórmula foi gerada para este trabalho.

A Fórmula 6.4 assumindo os valores já apresentados para *alfa* e para *velocidadeMaxima*, temos o seguinte gráfico:

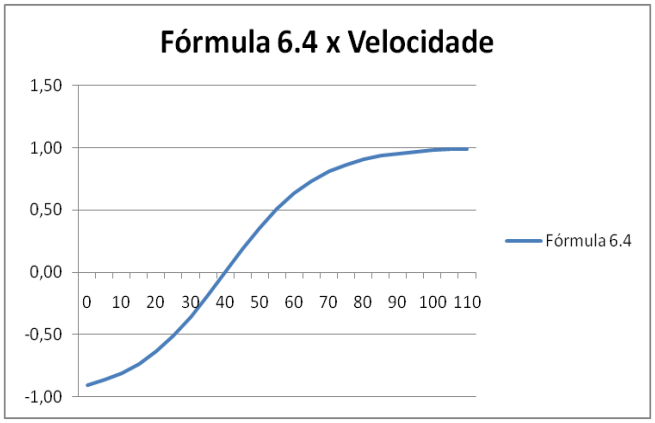


**Figura 5: Gráfico da Fórmula 6.4 para valores de velocidade entre 0km/h e 110km/h.**

Podemos perceber a diferença existente entre os gráficos da Figura 4 e da Figura 5. O gráfico da primeira apresenta um aumento da distância de segurança (eixo *y*) muito mais abrupto à medida que a velocidade aumenta (eixo *x*). O gráfico da segunda exibe uma relação muito mais suave. Isso se deve à constante *alfa* inserida na Fórmula 6.4.

Observamos, entretanto, que a função está definida no intervalo [-1,1]. Para obtermos o resultado desejado, basta adicionar "1" ao seu resultado, deslocando o gráfico para valores positivos de distância de segurança. A Fórmula 6.5 demonstra esta etapa.

$$1 + tanh\left(\left(\alpha \times \frac{velocidadeAtual}{velocidadeMaxima}\right) - \frac{\alpha}{2}\right) \quad (6.5)$$

A Figura 6 mostra como fica o gráfico após esse ajuste.

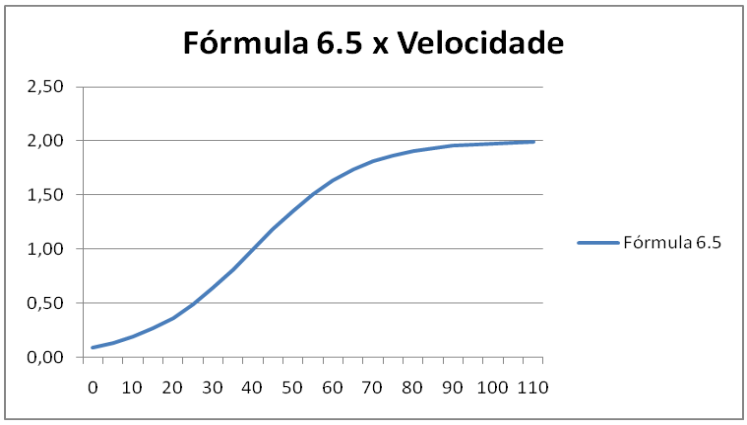


**Figura 6: Gráfico da Fórmula 6.5 para valores de velocidade entre 0km/h e 110km/h.**

Percebemos agora que o intervalo de dados obtido por esta fórmula é [0,2]. Precisamos fazer com que essa fórmula gere o intervalo [0,*Dist.Seg.Maxima*] que é de nosso interesse para a simulação. Para isso, multiplicamos a função pela metade da distância de segurança máxima (estipulada em "22m" nesta simulação). Obtemos então a Fórmula 6.1 apresentada anteriormente, que é a fórmula final da distância de segurança. Abaixo, o gráfico obtido com esta fórmula.

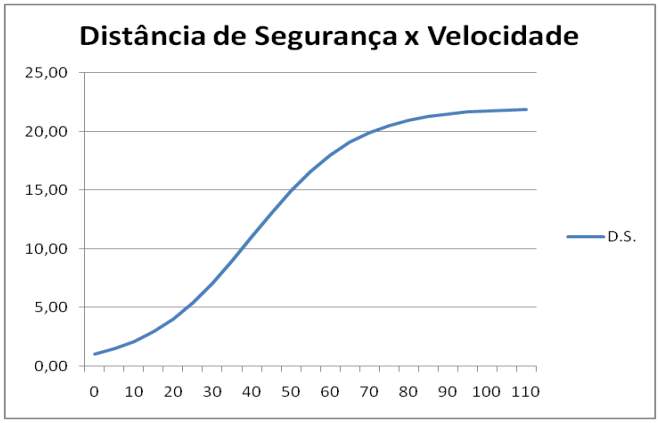


**Figura 7: Gráfico da Fórmula final da distância de segurança para valores de velocidade entre 0km/h e 110km/h.**

# 7 Ambiente do Simulador

A interface com o usuário é um ponto muito importante deste tipo de trabalho. De pouco adianta termos uma simulação com vários recursos, se o usuário não se sentir atraído a experimentá-la e, dentro de pouco tempo, não estar confortável com os comandos que tem a disposição.

Sem esquecermos a usabilidade, temos que levar em conta também a aparência da interface. Cores bem escolhidas e botões e textos dispostos de forma organizada são fundamentais para não confundir o usuário (Figura 8).


**Figura 8: Interface do Simulador.**

## 7.1 Opções de customização da simulação

A interface do simulador apresenta algumas opções de interação com o usuário:

- Forçar um veículo a trocar de faixa;
- Aumentar / Diminuir a velocidade máxima de um veículo;
- Aumentar / Diminuir o índice de agressividade de um veículo;
- Adicionar / Remover um Veículo;
- Selecionar um Veículo e visualizar seu estado atual;
- Ocultar / Exibir os *bounding boxes* que mostra as colisões laterais, frontais e no próprio veículo

A seguir vamos abordar cada uma destas opções, explicando suas funcionalidades.

A Figura 9 apresenta em detalhe os botões para escolher a pista na qual a simulação será executada. A pista escolhida fica com a borda do botão em amarelo. Atualmente possuímos apenas duas pistas.


**Figura 9: Detalhe da interface do simulador. Botões para escolha da pista.**

A Figura 10 detalha alguns botões de ação presentes na simulação.


**Figura 10: Detalhe de um conjunto de botões de ação.**

A seguir, descrevemos cada um desses botões:

- O botão “Adicionar Veículo” insere um novo veículo aleatório na simulação. O veículo inserido aparece destacado com seu *bounding box* na cor azul;

- O botão “Remover Veículo” retira da simulação o veículo que estiver selecionado (mais a frente veremos como selecionar um veículo);

- O botão “Ocultar Colisões Frontais” faz com que o *bounding box* frontal dos veículos (o triângulo que fica posicionado à frente dos veículos e que serve para detectar a aproximação de um veículo a sua frente) deixe de ser exibido. Quando clicado, esse botão troca de nome, passando a se chamar “Exibir Colisões Frontais” e tem função inversa a descrita anteriormente;

- O botão “Ocultar Colisões Laterais” faz com que o *bounding box* lateral dos veículos (o retângulo maior que fica posicionado ao redor dos veículos e que serve para detectar a presença de um veículo na faixa lateral) deixe de ser exibido. Quando clicado, esse botão troca de nome, passando a se chamar “Exibir Colisões Laterais” e tem função inversa a descrita anteriormente;

- O botão “Ocultar Bounding Boxes” faz com que o *bounding box* dos veículos (o retângulo menor que fica posicionado imediatamente redor dos veículos e que serve para detectar uma colisão direta com o veículo) deixe de ser exibido. Quando clicado, esse botão troca de nome, passando a se chamar “Exibir Bounding Boxes” e tem função inversa a descrita anteriormente;

A Figura 11 detalha a lista que exibe os veículos que estão sendo utilizados na simulação. Através desta lista podemos selecionar um veículo para executarmos algumas ações sobre o mesmo. Quando um veículo é selecionado, sua representação gráfica correspondente destaca-se com um *bounding box* na cor amarela, indicando, na pista, o veículo correspondente. A seguir vamos ver quais ações podemos executar sobre os veículos.


**Figura 11: Lista que exibe os Veículos atualmente na Simulação.**

A Figura 12 destaca um painel com botões e informações relativas ao veículo selecionado.


**Figura 12: Painel com controles direcionados a um Veículo específico.**

A seguir, detalhamos o que cada botão e informação exibida representam:

- O botão “Forçar Troca de Faixa” faz com que o veículo selecionado tente trocar de faixa de rodagem naquele instante. Caso não seja possível (devido a um veículo ocupando a faixa lateral) o *bounding box* lateral fica momentaneamente com uma cor avermelhada, indicando que houve “colisão” e não foi realizada a troca. Caso a troca seja possível, o veículo passa para a faixa ao lado e a simulação prossegue normalmente;

- Os botões “-1” e “+1” decrementam ou incrementam, respectivamente, o índice de agressividade do veículo em uma unidade. O limite inferior desse atributo é “0” e o superior é definido na classe de configuração (o padrão é “10”). O texto que fica entre os botões informa o valor atual deste atributo, ou seja, o índice de agressividade atual do veículo selecionado;

- Os botões “-5” e “+5” decrementam ou incrementam, respectivamente, o a velocidade em pista livre do veículo em cinco unidades. O limite inferior desse atributo é “0” e o superior não é estipulado. O texto que fica entre os botões informa o valor atual deste atributo, ou seja, a velocidade máxima que o veículo irá atingir quando não estiver sendo retido por outro veículo;

- O texto que aparece na parte inferior da figura informa a velocidade atual do veículo naquele momento na simulação.

A Figura 13 detalha o botão “Paly / Pause” da simulação. Quando a mesma está em execução esse botão assume o estado “Pause”, cuja função é pausar a simulação. Caso contrário assume o estado “Play”, cuja função é executar a simulação.

**Figura 13: Detalhe do botão Play / Pause: à esquerda no estado “Pause”; à direita no estado “Play”.**

# 8 Análise dos resultados

Passaremos a seguir a uma análise dos resultados do nosso projeto, tentando mostrar os nossos sucessos, as etapas que ainda necessitam de aperfeiçoamento, e encerrando com idéias que podem trazer melhoras para a simulação.

## 8.1 Objetivos alcançados

No início deste trabalho, traçamos algumas metas tanto no que tange a parte do desenvolvimento da simulação, quanto à parte do aprendizado de uma nova tecnologia (JavaFX). Podemos dizer que obtivemos um elevado grau de sucesso em abas as metas.

Quanto à etapa da simulação, obtivemos bons resultados no que nos propusemos a fazer, e temos certeza de que esse protótipo servirá de base para extensões e melhorias para softwares educacionais nessa área.

No que diz respeito ao aprendizado da linguagem JavaFX Script, o resultado também foi proveitoso. Para desenvolver a simulação, utilizamos as principais funcionalidade que JavaFX tem a oferecer. Desde as mais simples, como formas geométricas e polígonos, até mais complexas, como a importação de vetores SVG como *paths* e controle das animações. Não podemos deixar de citar a integração com classes Java e o uso de Threads. Em alguns pontos tivemos que nos aprofundar bastante na linguagem para entender seu comportamento, e descobrir como implementar as necessidades que a simulação demandava. Um bom exemplo disso foi o uso de *binds* e de *triggers*.

## 8.2 Problemas encontrados

A tecnologia JavaFX está em um estado pouco maduro e dispõe de recursos ainda limitados. A versão 1.2 (utilizada neste trabalho) trouxe muitos avanços, mas algumas opções de customização das animações, por exemplo, ainda devem ficar para as próximas versões.

A performance da simulação ainda não está no nível desejado. O uso de uma *thread* para fazer a escrita dos dados da simulação em um arquivo, melhorou muito o seu desempenho, já que é sabido que a escrita na tela ou em arquivo é uma tarefa custosa em

Java. Mesmo assim quando adicionamos muitos veículos, ou deixamos a simulação executando por muito tempo, ainda observamos problemas nesse quesito.

## 8.3 Propostas de melhoria

Podemos fazer duas abordagens para propostas de melhoria: uma referente à simulação em si, abrangendo situações que não são contempladas atualmente; outra referente à performance e questões estruturais do projeto.

### 8.3.1 Propostas de melhorias para a simulação

Elencamos a seguir algumas propostas de melhorias para a simulação:

- Criar mais pistas, com diferentes traçados e maior comprimento;
- Criar pistas com mais de duas faixas de rodagem;
- Adicionar obstáculos nas pistas como, por exemplo, interdições de faixas ou semáforos;
- Criar pistas com situações de entroncamentos e cruzamentos;
- Identificar, de forma mais precisa, a distância que cada veículo está mantendo para o da frente;

### 8.3.2 Proposta de melhoria estrutural da simulação e questões de performance

Acreditamos que a utilização de *threads* possa ajudar a melhorar o desempenho. Conforme já foi explicado, a simulação ocorre dentro de um *loop* principal onde cada veículo é avaliado (“colisões”, trocas de faixa, ajustes de velocidade, etc.) de forma seqüencial. Somente após avaliar todos os veículos pode-se representar graficamente os resultados obtidos, através da animação.

Uma alternativa é a criação de uma *thread* para cada veículo da simulação, fazendo com que cada *thread* execute as operações, as mesmas executadas pelo *loop* principal, sobre o seu respectivo veículo. Dessa forma as operações ficam descentralizadas e se obtém um alto grau de paralelismo. A expectativa é que isso acarrete em um aumento de desempenho geral da simulação.

# 9 Conclusões

Iniciamos este trabalho com o objetivo de desenvolver protótipo de simulação de trânsito voltado para fins educacionais. Portanto era primordial que o projeto possuísse uma interface gráfica atraente, mas que não comprometesse o bom entendimento do seu código fonte. Nossa meta era criar uma base de software, em uma linguagem de fácil aprendizado e robusta, que pudesse servir de ponto de partida para outros projetos.

Um ponto que merece atenção é que, a partir da escolha de JavaFX, assumimos um objetivo que não estava planejado: o aprendizado de uma nova tecnologia, visto que, até o momento do início deste projeto, nunca a havíamos utilizado. Entretanto a curva de aprendizado foi bastante rápida.

A escolha da tecnologia JavaFX cumpriu os requisitos almejados para o projeto. Garantiu uma parte visual interessante e permitiu uma integração bastante simples com Java. A linguagem JavaFX Script também preencheu bem o requisito de organização do código fonte, visto que, assim como Java, é orientada a objetos. Não podemos deixar de destacar a simplicidade que a sua sintaxe declarativa agregou ao projeto.

Além de criar toda a estrutura para execução da simulação, conseguimos implementar, ainda no protótipo, diversas situações vivenciadas no trânsito. Com isso tem-se a dimensão das possibilidades que podem ser alcançadas.

# 10 Referências Bibliográficas


[1] **WEAVER**, J. et al. *Pro JavaFX™ Platform: Script, Desktop and Mobile RIA with Java™ Technology.* Apress, 2009.

[2] **WEAVER**, J. *JavaFX Script: Dynamic Java Scripting for Rich Internet/Client-side Applications.* Apress, 2009.

[3] **MORRIS**, S. *JavaFX in Action.* Manning, 2010.

[4] Interactive Pulp. Disponível em: <http://www.interactivepulp.com/pulpcore/>. Acesso em 20 out. 2009.

[5] Golden T Studios - Golden T Game Engine (GTGE). Disponível em: <http://www.goldenstudios.or.id/products/GTGE/>. Acesso em 03 nov. 2009.

[6] JavaFX 1.2.1 API. Disponível em: <http://java.sun.com/javafx/1.2/docs/api/>. Acesso em 20 fev. 2010.

[7] JAVAFX | Programming Games in JavaFX - Part 1. Disponível em: <http://remwebdevelopment.com/dev/a41/Programming-Games-in-JavaFX-Part-1.html>. Acesso em 10 dez. 2009.

[8] JavaFX Tutorial | Develop Applications for Desktop and Mobile. Disponível em: <http://java.sun.com/javafx/1/tutorials/ui/>. Acesso em 10 dez. 2009

[9] JavaFX, rectangular collision detection. Disponível em: <http://silveiraneto.net/2008/10/30/javafx-rectangular-collision-detection/>. Acesso em 10 dez. 2009.

[10] JavaOne Technical Sessions - JavaFX Platform: Animations, Timelines, and Collision Analysis for Games – 2008. Disponível em: <http://developers.sun.com/learning/javaoneonline/j1sessn.jsp?sessn=TS-5280&yr=2009&track=media>. Acesso em 10 dez. 2009.

[11] Online LaTeX Equation Editor. Disponível em: <http://www.codecogs.com/components/equationeditor/equationeditor.php >. Acesso em 28 fev. 2010.

[12] **ASSIS**, Gilda A. et al. *EducaTrans: um Jogo Educativo para o Aprendizado do Trânsito*. CINTED/UFRGS, Porto Alegre - RS, V.4 Nº 2, Dezembro, 2006.

[13] **BALBINOT**, Amanda B.; **TIMM**, Maria I.; **ZARO**, Milton A. *Aplicação de Jogos e Simuladores como Instrumentos para Educação e Segurança no Trânsito*. CINTED/UFRGS, Porto Alegre - RS, V. 7 Nº 1, Julho, 2009.

[14] Inkscape. Draw Freely. Disponível em <http://www.inkscape.org/>. Acesso em 28 fev. 2010.

[15] Google Code. Disponível em < http://code.google.com/intl/pt-BR/>. Acesso em 28 fev. 2010.

[16] NetBeans. Disponível em <http://netbeans.org/http://www.inkscape.org/>. Acesso em 28 fev. 2010.

[17] JMonkey. Disponível em <http://www.jmonkeyengine.com/index.php>. Acesso em 08 mar. 2010

[18] JavaFX | Get Started - What is JavaFX? | Java FX. Disponível em <http://javafx.com/pt_BR/docs/gettingstarted/javafx/>. Acesso em 08 mar. 2010

[19] Visão geral do JavaFX. Disponível em <http://javafx.com/pt_BR/about/overview/>. Acesso em 08 mar. 2010.

[20] JavaFX GUI Basics. Disponível em <http://www.javapassion.com/javafx/javafx_guibasics.pdf>. Acesso em 08 mar. 2010.