Curso de Sistemas de Informação
Universidade Estadual de Mato Grosso do Sul

Implementação de um Game Interativo Utilizando a Biblioteca *OpenCV*

Antonio Ronaldo Nugoli

Prof. Msc. Diogo Fernando Trevisan(Orientador)

Dourados -MS
Setembro de 2016

Curso de Sistemas de Informação
Universidade Estadual de Mato Grosso do Sul

Implementação de um Game Interativo Utilizando a Biblioteca *OpenCV*

Antonio Ronaldo Nugoli

Setembro de 2016

Banca Examinadora:

Prof. Msc. Diogo Fernando Trevisan (Orientador)
Área de Computação - UEMS

Prof[a]. Dr[a]. Mercedes Rocío Gonzales Márquez
Área de Computação - UEMS

Prof[a]. Msc. Jéssica Bassani de Oliveira
Área de Computação - UEMS

Implementação de um Game Interativo Utilizando a Biblioteca *OpenCV*

Antonio Ronaldo Nugoli

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso II devidamente corrigida e defendida por Antonio Ronaldo Nugoli e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Dourados, 31 de outubro de 2016.

Prof. Msc. Diogo Fernando Trevisan
(Orientador)

Dedicatória

A minha esposa *Elisangela*, aos meus pais, filhos e minha irmã, com amor, admiração e gratidão pelo apoio incansável ao longo do período de elaboração deste trabalho.

Agradecimentos

Agradeço primeiramente a Deus por me dar saúde e força para superar as dificuldades, ao meu orientador professor Diogo, que com paciência e sabedoria me ajudou a trilhar o caminho para conclusão deste trabalho.

Em especial a minha família, que sempre me deu o apoio necessário para que não desistisse.

E por fim, aos meus amigos que conquistei durante os anos de graduação os quais sempre terei grande estima.

Resumo

O objetivo deste projeto foi implementar um *game* que utilizasse processamento de imagens e visão computacional como forma de obter mais interatividade.

A maioria dos *games* utilizam como dispositivos de entrada teclados, *mouse* ou *joysticks*. Neste projeto foi utilizada uma *webcam* para rastreamento de objetos físicos que podem servir como meios de interação com o *game*. Para isso, foi desenvolvido um pequeno *game* de corrida onde o jogador pode utilizar objetos para fazer com que o carro vire, como se fossem o volante do mesmo.

Este jogo foi desenvolvido utilizando a linguagem de programação *Java* e a biblioteca de visão computacional *OpenCV*, a qual auxiliou na captura e rastreamento dos objetos que servem para controlar o veículo.

Palavras-chave: *Jogos Interativos*, *Visão Computacional*, *OpenCV*, *Rastreamento*, *Processamento de Imagem*.

Abstract

The objective of this project was to implement a game that used image processing and computer vision as a way to get more interactivity.

Most games use as input keyboards, mouse or joysticks. In this project we used a webcam for tracking physical objects that can serve as a means of interaction with the game. For this, we developed a small racing game where the player can use objects to make the car turn, like the steering wheel of the same.

This game was developed using the Java programming language and OpenCV computer vision library, which helped capture and tracking of objects that serve to control the car.

Keywords: *Interactive Games*, *Computer Vision*, *OpenCV*, *Tracking*, *Image Processing*.

Sumário

Lista de Siglas

OpenCV *Open Source Computer Vision Library*

HCO *Open Handset Alliance*

IDE *Integrated Development Environment*

FPS *Frames por segundos*

SSD *Solid State Drive*

SVH *Sistema de Visão Humano*

CCD *Charge Coupled Device*

CMOS *Complementary Metal Oxide Semiconductor*

API *Application Programming Interfaces*

BSD *Berkeley Software Distribution*

IPL *Image Processing Library*

CPU *Central Processing Unit*

MIT *Massachusetts Institute of Technology*

RAM *Random Access Memory*

Lista de Figuras

Capítulo 1

Introdução

Este capítulo tem como objetivo introduzir o tema da pesquisa. Para tanto, será explanado de forma simples o que incitou a pesquisa e por qual razão desenvolvê-la, bem como os objetivos gerais e específicos deste trabalho.

1.1 Justificativa e motivação

Com a constante evolução computacional é comum termos acesso a computadores cada vez mais potentes, e isso não é tudo. Seus periféricos também evoluíram. Pode-se citar como exemplo a captura de vídeo, hoje podemos realizar vídeo conferências, tirar fotos em alta definição, tudo isso utilizando o próprio computador, celulares ou *tablets*.

Todo esse progresso, abre uma gama enorme de possibilidades para criação de novos aplicativos, inclusive na área de visão computacional onde é possível utilizar todos esses recursos de hardware para disponibilizar ao usuário aplicativos, que, através de sua câmera possam extrair informações do ambiente externo que lhe ajudem de diversas maneiras, sendo uma desta o foco referência para este projeto – a implementação de jogos. A união de todo este potencial, somado ao fascínio pelos *games*, proporciona uma imersão cada vez maior aos jogadores.

1.2 Objetivos

O objetivo deste projeto foi implementar um *game* no estilo corrida de carros, fazendo uso dos recursos da biblioteca de visão computacional *OpenCV*, para proporcionar ao jogador maior interatividade, dispensando o uso de dispositivos físicos como teclados, *mouse* ou *joysticks* para controlar a movimentação do seu veículo.

1.2.1 Objetivos Específicos

Os objetivos específicos são:

- Utilizar o processamento de imagens para criar um *game* de corrida de carros interativo;
- Permitir ao jogador controlar o carro sem uso de controles físicos como *mouse* teclado ou *joystick*[1];
- Capturar a imagem do jogador em tempo real e rastrear objetos com formato geométrico quadrilátero;
- Utilizar informações de rastreamento do objeto obtidas com a biblioteca *OpenCV* para guiar o carro durante todo trajeto especificado na corrida;
- Ser compatível com a maioria das *webcams*[2] disponíveis no mercado;

1.3 Metodologia

Para o desenvolvimento desse trabalho foi feita uma pesquisa bibliográfica, a fim de levantar os conceitos e as características da linguagem e biblioteca utilizada no desenvolvimento do projeto.

O projeto foi desenvolvido para ser compatível com as versões mais usuais do Sistema Operacional *Windows*.

A ferramenta utilizada no desenvolvimento foi o *Eclipse*, por se tratar de uma *Integrated Development Environment* (IDE) *Open Source* com fácil configuração e utilização.

Para a codificação do jogo foi utilizada a linguagem de programação *Java* e também a biblioteca *OpenCV*.

1.4 Organização dos Capítulos

O texto do projeto está organizado em um único volume, contendo listas de figuras e códigos. Além deste Capítulo, o volume é organizado em outros 3 capítulos e a conclusão, cujos conteúdos são apresentados a seguir.

No **Capítulo 2**, é apresentado um conceito básico sobre assuntos relevantes na elaboração do presente projeto. No **capítulo 3**, é explanado sobre o desenvolvimento do jogo. O **Capítulo 4** trata sobre os testes realizados. O **Capitulo 5** é a conclusão.

[1]Dispositivo de entrada composto por alavancas e botões que quando pressionados executam certar ações. São utilizados com maior frequência em consoles de vídeo games ou em jogos de computadores.

[2]Dispositivo de baixo custo para captura de imagens em computadores portáteis ou de mesa

Capítulo 2

Revisão Bibliográfica

Nesse capítulo serão apresentados o histórico e conceitos de imagens, visão computacional, modelo de cores, filtros, a linguagem *Java* e a biblioteca de visão computacional *OpenCV* que estarão presentes no desenvolvimento do projeto.

2.1 Imagens

2.1.1 Histórico

Imagem é a representação visual do que os olhos podem ver. Até o século VIX ela podia estar somente em nossas lembranças ou representada através da pintura de um artista, mas em 1826 um francês chamado *Joseph Nicéphore Niépce*, por meio de um sistema rudimentar, conseguiu capturar em um *negativo*[1], uma imagem fixa, considerada como a primeira fotografia, exibida na Figura 2.1.

[1]Placa de estanho coberta com derivado de petróleo fotossensível.

Figura 2.1: "View from the Window at Le Gras"("Vista da janela no Le Gras"), de *Joseph Nicéphore Niepce*, de 1826, imagem capturada em Saint-Loup-de-Varennes (França)
Fonte: www.incinerrante.com (2015).

Segundo pointdaarte.webnode.com.br (2011), esta invenção significaria o início para novas áreas de estudos, que com sua evolução permitiu registrar momentos históricos e passou de uma simples imagem quase ilegível tirada da janela do quarto de *Joseph* para incríveis efeitos alcançados hoje em dia com os avanços no processamento de imagens digitais.

Porém, essa evolução não ocorreu assim tão rápido. Durante quase um século as pesquisa foram voltadas unicamente para a melhoria na qualidade das imagens capturadas. Novas técnicas foram criadas como a fotografia colorida e a sua própria popularização como objeto de consumo. Então, na década de 20, surgiu a necessidade para que meios de comunicação como os grandes jornais da época separados por distâncias continentais pudessem trocar as imagens entre si de maneira mais rápida.

Segundo BARBOZA (2004) a forma encontrada para tanto foi o sistema *Bartlane* de transmissão criado por *Harry G. Bartholomew e Maynard D. McFarlane* que consistia em um sistema adaptado de *radiotelegrafia*[2], onde uma imagem era codificada na origem e por meio de cartões perfurados eram transmitidas ao destino em cabos submarinos onde se realizava a recepção e decodificação, através de fitas telegráficas simuladas por carácteres onde foram utilizadas apenas cinco tons de brilho que posteriormente após nove anos foi significativamente aumentado para quinze tons, aumentando assim a qualidade da imagem recebida, como pode ser visto na Figura 2.2.

[2]Método de telegrafia sem fio pela qual são transmitidas mensagens em código Morse, por meio de ondas eletromagnéticas

Figura 2.2: Imagem dos generais Pershing e Foch,transmitida em 1929 a partir de Londres para Nova Iorque com 15 níveis de brilho distintos
Fonte: GONZALEZ (2002).

De acordo com MARQUES FILHO (1999) o grande avanço para os estudo na captura de imagens ocorreu nas décadas de sessenta, com o advento dos computadores de grande porte e o programa espacial norte americano, com os estudos realizados para as correções e distorções de imagens recebidas pela sonda *Ranger*[3] e enviar imagens para terra através de aparelhos televisores.

Esse aumento na pesquisa do aprimoramento no processamento de imagens trouxe inúmeros benefícios para diversas áreas de pesquisa como medicina, biologia, astrologia, geoprocessamento e robótica.

2.1.2 Aquisição de Imagens

De acordo com MARQUES FILHO (1999) o processo de capturar uma cena real tridimensional para uma imagem eletrônica se dá através de uma redução da dimensionalidade, convertendo esta cena 3D em uma representação 2D sendo assim denominado como processo de aquisição.

Ainda segundo MARQUES FILHO (1999), os dispositivos eletrônicos mais utilizados para esta aquisição são câmeras com tecnologia *Charge Coupled Device* (CCD) ou atualmente também o *Complementary Metal Oxide Semiconductor* (CMOS), sendo estes sensores de captura de imagem, onde através de lentes e conjuntos de prismas e filtros de cor tem a finalidade de conseguir decompor a luz recebida em sinais elétricos, e, posteriormente gerar uma imagem combinada para um padrão de cor correspondente ao utilizado pelo dispositivo de saída.

Na Figura 2.3 pode-se ter uma ideia simplificada do funcionamento na etapa de aquisição de imagens em uma câmera com sensor *CCD*.

[3]O Programa *Ranger*, foi constituído por uma série de missões espaciais não tripuladas, conduzidas pelos Estados Unidos, na década de 60, com o objetivo de obter imagens ”em close”da superfície Lunar.

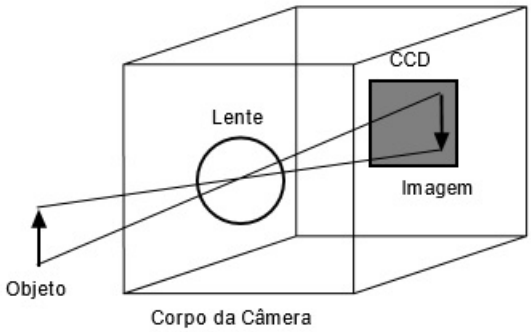

Figura 2.3: Aquisição de imagens em câmeras *CCD*.
Fonte: MARQUES FILHO (1999).

2.1.3 Armazenamento

Uma grande dificuldade no estudo do processamento de imagens digitais está no seu armazenamento em razão a grande quantidade de *bytes*[4] que são necessários para seu manuseio e arquivamento.

Se for levado em consideração apenas a manipulação destas em computadores mesmo que de uso doméstico, isso pode parecer algo irrelevante tendo como base o atual nível tecnológico em que a humanidade se encontra, com computadores cada vez mais bem equipados, portadores de capacidades significantes para armazenamento e processamento. Porém, quando se trata de dispositivos móveis, essa premissa ainda é relevante. Quando se trabalha com desenvolvimento para tais dispositivos é preciso levar em consideração que grande parte dos aparelhos disponíveis para os usuários dispõem de poucos recursos.

De acordo com MARQUES FILHO (1999), há três categorias em que se pode classificar a forma de trabalhar com armazenamento de imagens:

1. Armazenamento de curta duração;
2. Armazenamento de massa;
3. Arquivamento de imagens.

No primeiro caso a imagem fica em memória apenas durante o tempo de processamento. Neste caso é aconselhado o uso da memória RAM, ou de placas especificas conhecidas como *Frame Buffers*[5]. Estas são memórias específicas para tratamento de imagens, presentes nas placas gráficas atuais e podem conter centenas ou milhares de megabytes. Operações como *zoom* e rolagem, realizadas nestas circunstâncias são feitas instantaneamente devido ao tempo rápido de resposta propiciado por esse tipo de memória.

No segundo caso, se encaixam operações de recuperação relativamente rápidos. Isto é feito através do armazenamento em unidades físicas como discos magnéticos, ópticos ou *Solid State Drive (SSD).*

Um exemplo de cálculo para o custo de armazenamento é feito de maneira simples, calculando o número de *pixels*[6] na horizontal e multiplicado pelo número de *pixels* na vertical, vezes o número de *bits* necessários para escala de cinza dividido por oito. Isso, considerando imagens monocromáticas que são representadas apenas por uma matriz contendo os valores dos tons de cinza para cada pixel, o que não ocorre com as imagens coloridas que demandam de um cálculo muito mais complexo, pois, na prática, esses valores são substituídos por referências às paletas de cores específicas da imagem analisada.

E, por fim, tem-se o arquivamento ou armazenamento das imagens, onde são considerados outros valores que ficam registrados em um cabeçalho no arquivo, sendo estas

[4] O *byte* é uma unidade de medida computacional que equivalem a oito *bits* sendo essa a menor unidade de medida computacional

[5] Memória de acesso rápido em alta velocidade para armazenamento de imagens

[6] O pixel é a menor unidade de uma imagem digital

informações sobre o tamanho da imagem, números de cores ou tons de cinza, dentre outras que podem variar conforme seu formato, sendo os mais comuns o BMP, PCX, TIFF, JPEG, GIF e PNG.

2.2 Visão Computacional

Segundo FORSYTH and PONCE (2012), visão computacional é a ciência que utiliza das técnicas de processamento digital de imagens para, através de métodos estatísticos construídos em modelos baseados em dados geométricos, físicos e técnicas de inteligência artificial, obter um solido conhecimento do processo de formação da imagem.

A partir das informações de valores individuais de cada *pixel*, combinados com as informações disponíveis em múltiplas imagens, e como um todo, impor alguma ordem a grupos distintos de *pixels*, desta forma dando a possibilidade de reconhecer e interpretar objetos .

Ainda segundo FORSYTH and PONCE (2012), A visão computacional tem uma ampla área de aplicação como robótica, inspeção industrial e inteligência militar, análise de imagens nas áreas médicas, entre outras.

2.3 Processamento de Imagens

Diferente do *Sistema de Visão Humano (SVH)*, que nos permite reconhecer padrões em imagens mas que são facilmente influenciadas por fatores externos como fadiga, iluminação entre outros, a visão computacional, através do processamento digital, tem a capacidade de processar um gigantesco volume de informações contidas em uma única imagem.

Após esse processamento segundo GONZALEZ (2002), é possível empregar diversas técnicas para que a interpretação destes dados alcance os resultados almejados para os fins que se destinam.

Segundo SILVA (2011) é possível considerar o processamento digital de imagem, como técnicas de manipulação por computador de forma que a saída de uma imagem processada seja um produto que atenda os requisitos esperados tanto na melhora do seu aspecto visual quanto na interpretação e extração de dados contidos na imagem original.

Para alcançar tais resultados na manipulação e edição das imagens, várias técnicas podem ser aplicáveis através de inúmeros filtros, sendo estes transformações *pixel* a *pixel* em uma imagem, que levam em consideração os níveis de cinza dos *pixels* vizinhos.

2.4 JAVA

De acordo com DEITEL and DEITEL (2010) a linguagem *Java* nasceu da visão empreendedora de uma empresa chamada *Sun Microsystems*, que, ao ver o grande impacto que os

microprocessadores estavam tendo em dispositivos eletrônicos de consumo popular em 1991 financiou projetos internos visando a exploração deste mercado.

Tendo como base a linguagem *C++* chegou à criação de uma nova linguagem orientada a objetos denominada *JAVA*.

Sua popularidade se tornou ainda mais crescente quando em 1993 a *Sun Microsystems*, novamente com sua visão empreendedora, lançou o *Java* na *WEB* como ferramenta para criação de conteúdo dinâmico para páginas da *internet*.

O ciclo de desenvolvimento do programa em *Java* pode ser facilmente percebido tomando como base a Figura 2.4.

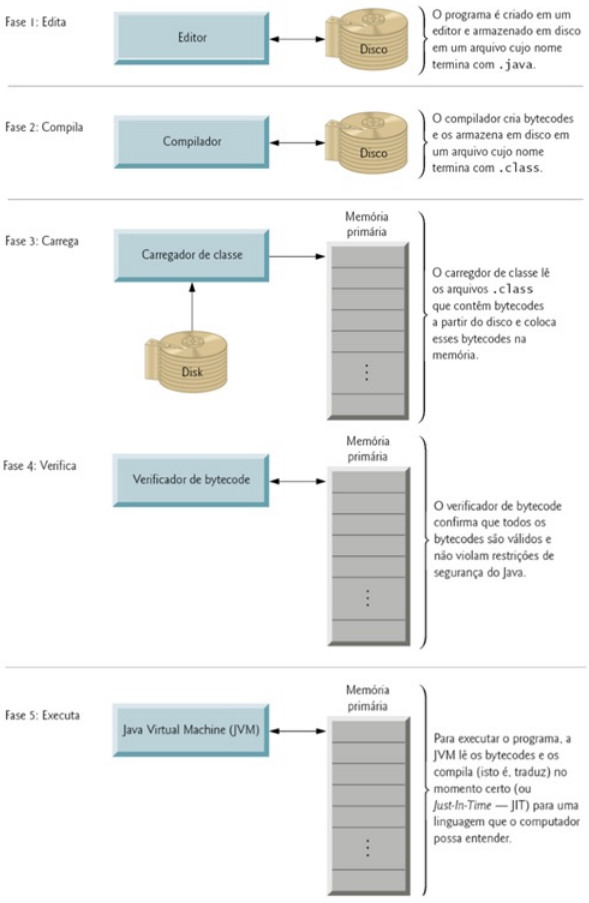

Figura 2.4: Fases do desenvolvimento Java
Fonte: DEITEL and DEITEL (2010), pag.9.

Basicamente, programas em *Java* são constituídos por classes, que são compostas por métodos, que, por sua vez tem a finalidade de executar tarefas e retornar informação das tarefas concluídas. Possibilitando desta forma uma rica coleção de bibliotecas também conhecidas por *Java Application Programming Interfaces (API)* , que com sua vasta documentação facilitam a vida dos programadores.

2.5 Biblioteca OpenCV

Open Source Computer Vision Library (OpenCV) é uma biblioteca de uso livre, sob licença *BSD*[7] Intel , criada inicialmente pela Intel no ano de 2000 com o propósito de avançar nos estudo de aplicações para uso intensivo de *Central Processing Unit* (CPU). Foi posteriormente distribuída para universidades como o *Massachusetts Institute of Technology* (MIT) para apoiar os acadêmicos no estudo de visão computacional.

A biblioteca possui mais de 500 funções BRADSKI (2008), é escrita em *C/C++* e está disponível para sistemas operacionais *MAC*, *Windows*, *Linux* e para sistemas *mobile* na plataforma *Android*.

Uma das vantagens da utilização da biblioteca *OpenCV* é o fato de já na sua execução ser detectado o tipo de processador e posteriormente ajustar suas funções, otimizando o foco em tempo de processamento. Seu funcionamento depende parcialmente da biblioteca *Image Processing Library (IPL)* BRADSKI (2008).

Juntamente com a biblioteca também é disponibilizado um pacote de exemplos que ajuda aos iniciantes terem prévio conhecimento das funções básicas.

A biblioteca *OpenCV* pode ser dividida em cinco grupos básicos de funções:

1. Processamento de imagens;
2. Análise estrutural;
3. Análise de movimento e rastreamento de objetos;
4. Reconhecimento de padrões e Calibração de câmera;
5. Reconstrução 3D.

A biblioteca está atualmente na versão 3.0 e possui, além da interface de desenvolvimento para linguagem *C/C++*, portabilidade também para linguagens *Python* , *Java*, *Visual Basic* e *Android*.

2.6 Detecção de Contornos com *OpenCV*

A detecção de contornos tem como objetivo evidenciar áreas de um objeto em uma imagem. Este procedimento tem grande aplicabilidade na área de processamento de imagens, principalmente quando o objetivo é extrair informações de determinados pontos de interesse.

Segundo BRADSKI (2008) as funções utilizadas na biblioteca *OpenCV* para rastreamento de contorno o representa como lista de pontos (*pixels*), que representam de uma maneira ou de outra, uma curva em uma imagem. Esta representação varia de acordo com as circunstâncias, já que há muitas maneiras de representar uma curva.

[7] BSD é uma licença de código aberto inicialmente utilizada nos sistemas operacionais do tipo *Berkeley Software Distribution*.

Ainda segundo BRADSKI (2008), uma das maneiras de se extrair informações sobre esta sequência de pontos é binarização, através da conversão de uma imagem com níveis de cinza para uma imagem com representação binária, ou seja com apenas dois tons. A biblioteca, através de suas funções especificas, separa em duas cores (preto ou branco) através de um ponto de corte conhecido como limiarização (*threshold*), evidenciando assim pontos de interesse em uma imagem, como pode ser visto na Figura 2.5.

Após a binarização e a evidenciação dos pontos de interesse, a biblioteca*OpenCV* oferece funções que localizam e gerenciam informações sobre os pontos de contorno presentes na imagem.

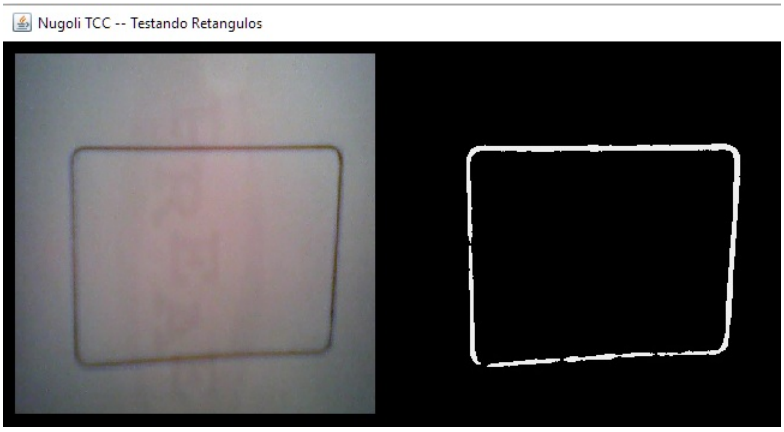

Figura 2.5: Exemplo de binarização realizada pelo método *adaptiveThreshold* da biblioteca *OpenCV* , A esquerda imagem original e a direita imagem binarizada.
Fonte: Autoria Própria

Capítulo 3

Desenvolvimento

Neste capítulo é apresentada a descrição, conceito e diagramas que facilitam e auxiliam na compreensão do projeto desenvolvido, assim como o detalhamento das classes implementadas e a apresentação das telas do jogo e sua codificação.

Para criação dos diagramas apresentados foram utilizados *plugins* existentes na própria plataforma de desenvolvimento *Eclipse*.

3.1 Descrição do Jogo

O *game* implementado neste projeto foi intitulado *Nugoli Racing Web Cam*. Este é um jogo de corrida onde o principal objetivo é percorrer a maior distância possível. Para isso o jogador deve portar em suas mãos algum objeto que contenha a forma quadrilateral, como por exemplo um retângulo desenhado sobre uma folha em branco, que é identificada pelo jogo e usada para indicar em qual direção seu carro deve seguir, desviando assim dos outros carros presentes na pista.

Com o passar do tempo o número de veículos a serem desviados aumentam, exigindo movimentos mais ágeis do jogador para que ele possa chegar sempre mais longe.

Na tela inicial são apresentados ao jogador três botões pelos quais ele pode optar em iniciar uma partida, alterar a resolução de vídeo (disponíveis três resoluções diferentes) e, por fim, um botão que apresenta uma tela com informações sobre o jogo.

Ao clicar em iniciar, o jogador é levado à pista de corrida onde o jogo terá início assim que algum objeto for detectado. No centro superior da tela o jogador verá a imagem de sua *webcam*, onde um marcador mostra ao jogador o objeto detectado juntamente com linhas delimitadores informando os cantos do objeto e seu ponto central, que servem de referência para indicar a intenção de movimentação do veículo.

O jogador deve evitar que seu carro colida com os demais carros, evitando assim o fim da partida. Na parte superior esquerda da tela um contador mostra para o jogador a distância percorrida.

Ao colidir com qualquer veículo adversário a partida é encerrada e uma mensagem

aparece informando a distância percorrida e o recorde atual. Caso jogador atinja uma distância maior que recorde, uma mensagem o parabenizando é mostrada.

Por fim, a tela inicial surge novamente dando a opção de encerrar o jogo ou iniciar uma nova partida.

3.2 Requisitos

Os requisitos utilizados no desenvolvimento do jogo seguem abaixo:

3.2.1 Requisitos Funcionais

- O jogo deve apresentar uma tela inicial com opções para iniciar o jogo;
- No início da partida deve ser apresentadas instruções ao jogador;
- O jogo deve mostrar a distância que foi percorrida;
- O jogo deve informar o fim da partida e a distância total percorrida;
- O jogo deve arquivar e comparar as distâncias percorridas a fim de estabelecer recordes dos jogadores;
- O jogo deve permitir a detecção de movimento do jogador através de uma *webcam*.

3.2.2 Requisitos Não Funcionais

- O jogo deve ser implementado em linguagem Java e a biblioteca de visão computacional *OpenCV*;
- O jogo deve ser desenvolvido para plataforma *desktop* com Sistema Operacional *Windows*;
- O jogador deverá ter uma *webcam* conectada ao computador.

3.2.3 Regras de Negócio

- O jogo termina sempre que houver uma colisão com outro carro na pista;
- O jogador será sempre informado se quebrou um recorde e terá opção de iniciar uma nova partida.

3.3 Modelagem

Na seguinte seção é apresentada uma breve descrição dos diagramas com finalidade de melhor exemplificar o conceito deste projeto.

3.3.1 Diagramas de Caso de Uso

O diagrama caso de uso na Figura 3.1 demonstra de que forma o jogo deve responder às entradas de vídeo realizadas pela *webcam*.

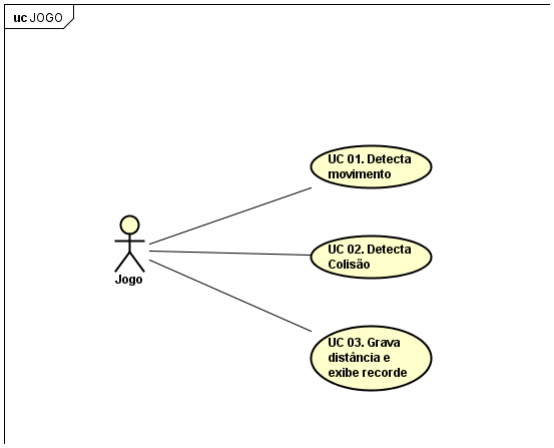

Figura 3.1: Diagrama de Caso de Uso do Jogo.
Fonte: Autoria Própria

Na Figura 3.2 são mostradas as ações que podem ser executadas pelo jogador.

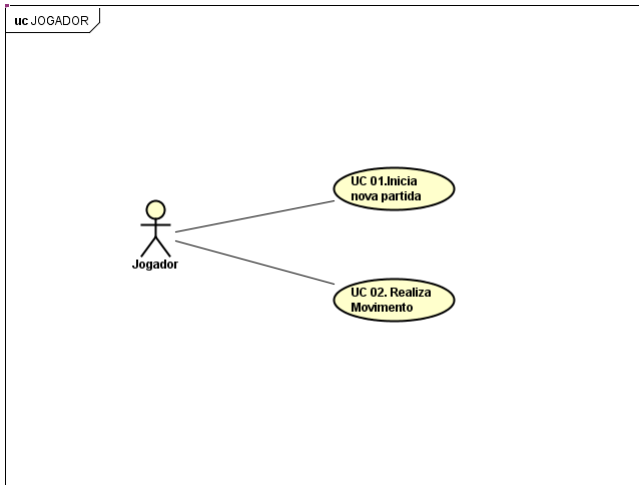

Figura 3.2: Diagrama de Caso de Uso do Jogador.
Fonte: Autoria Própria

3.3.2 Diagrama de Classe

A modelagem de classes, apresentada no diagrama da Figura 3.3, foi feita pensando para dividir as tarefas em pacotes separados, visando facilitar a integração das diferentes ferramentas.

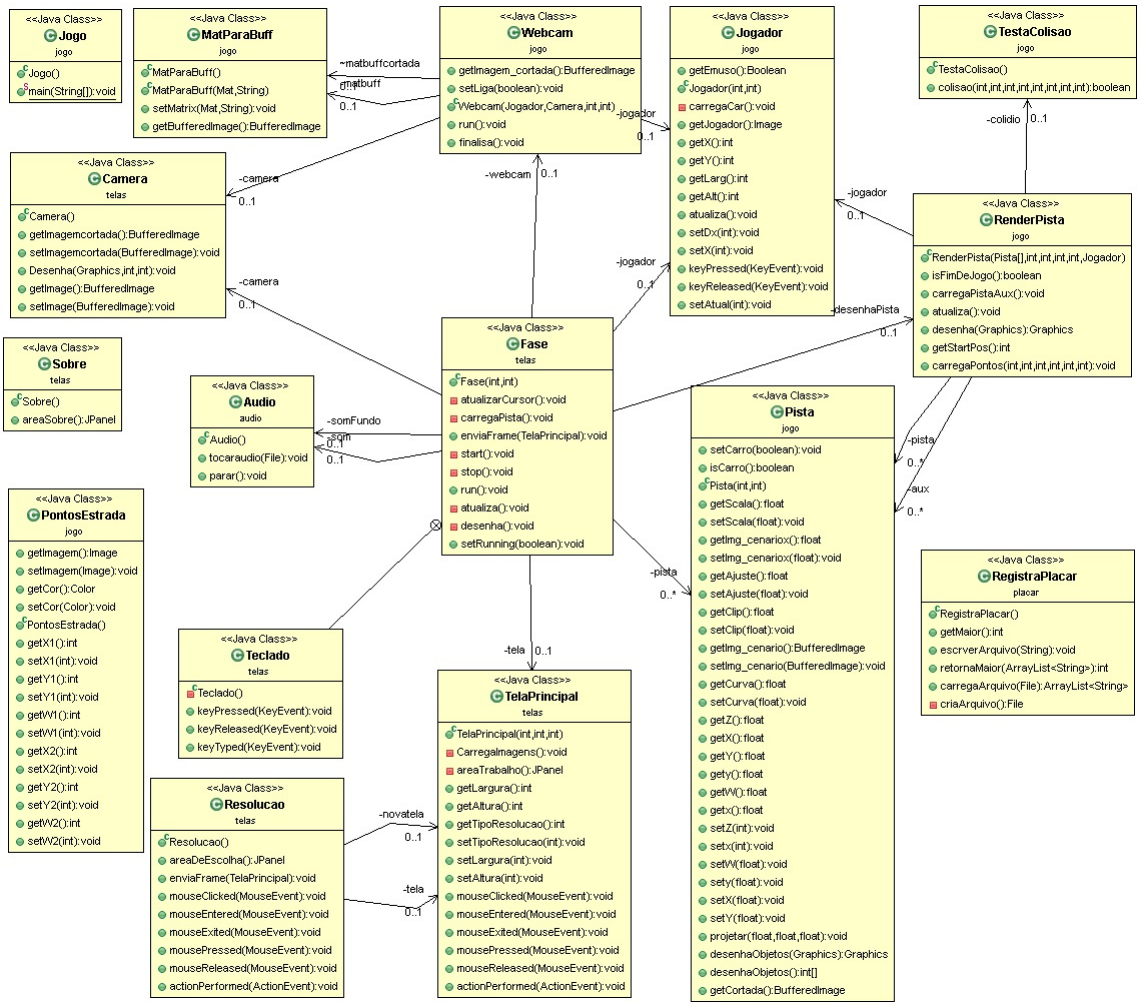

Figura 3.3: Diagrama de Classe.
Fonte: Autoria Própria

Como pode ser visualizado no diagrama, o projeto foi dividido em quatro pacotes distintos, sendo o principal denominado *jogo*, responsável pela inicialização e desenrolar da partida. Este pacote contém todos os métodos de processamento da fase e gerenciamento dos dados obtidos pela *webcam*. O segundo pacote, *telas*, fica responsável pelo gerenciamento da parte visual e os dois últimos *audio* e *placar* ficam responsáveis pelos efeitos sonoros e gerenciamento da pontuação obtida pelo jogador.

O detalhamento das principais classes será melhor explanado nas próximas seções.

3.4 Telas e Codificação

A Figura 3.4, mostra a tela inicial do jogo. Ao ser executado pela primeira vez, a classe *Jogo* instancia um novo objeto da classe *TelaPrincipal* responsável por carregar os botões e imagens para que o jogador possa escolher entre iniciar uma nova partida, alterar a resolução

ou obter informações sobre o jogo.

Figura 3.4: Tela inicial do jogo.
Fonte: Autoria Própria

Ao clicar no botão *sobre* é instanciado um novo objeto da classe *sobre* e é apresentada ao jogador a tela mostrada na Figura 3.5.

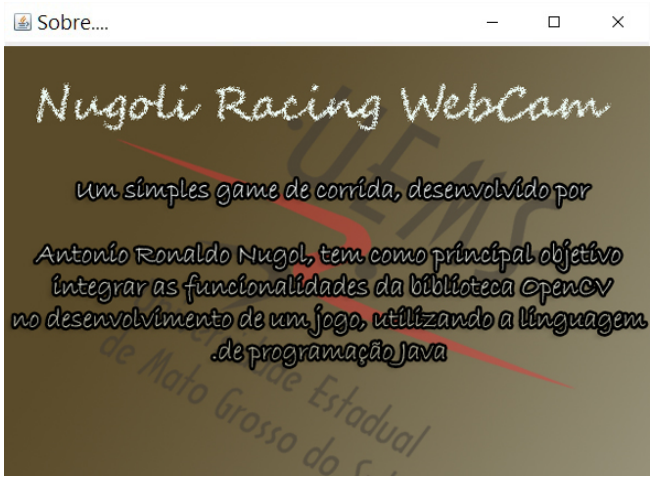

Figura 3.5: Tela sobre
Fonte: Autoria Própria

O botão *Alterar Resolução* instancia um objeto da classe *Resolucao* e apresenta a tela da Figura 3.6. O jogador pode escolher entre três resoluções distintas sendo 800x600 a resolução padrão. Esta configuração de tela serve como parâmetro e vai determinar a base para calcular as escalas de toda parte visual durante a execução da partida.

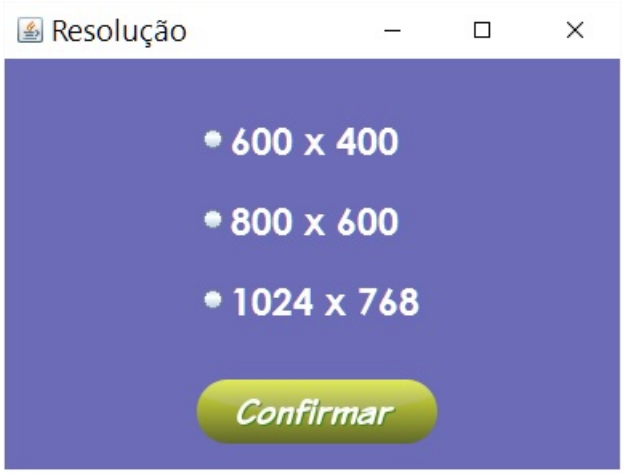

Figura 3.6: Tela para escolha de resolução.
Fonte: Autoria Própria

Ao clicar no botão *iniciar*, uma nova partida começa com a apresentação de uma mensagem ao jogador com intuito de lhe fornecer dicas sobre o jogo, como pode ser visto na Figura 3.7.

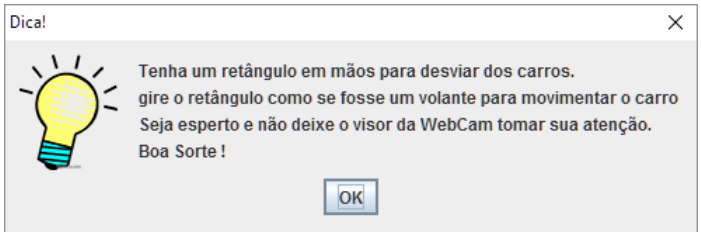

Figura 3.7: Tela de dica.
Fonte: Autoria Própria

A Figura 3.8 mostra uma partida em execução. Esta é iniciada através da instância de um novo objeto da classe *Fase*, o qual irá executar em *loop* as ações que darão andamento ao jogo, sendo elas descritas na seção 3.4.1.

Figura 3.8: Partida em execução.
Fonte: Autoria Própria

3.4.1 Loop do jogo

O *loop* do jogo, é responsável por manter todos os eventos que ocorrem durante a execução de uma partida atualizados, tratar as entradas do usuário, a inteligência artificial dos inimigos e execução de sons, tentando mostrar tudo isso de forma atualizada para jogador.

Segundo ANDREW (2005), o grande problema quando se trata das atualizações dentro de um *loop* é manter uma taxa de quadros por segundo aceitável para o jogador ter a impressão que o jogo corra fluidamente, sem estar travando ou rápido demais. Esta variação ocorre, pois, o tempo de execução de uma máquina pode ser diferente de outra. Para contornar esta situação ANDREW (2005) propõe controlar os tempos de execução entre os ciclos

do *loop*, controlando o tempo inicial e final de execução, mantendo sua velocidade constante independentemente da plataforma executante.

O código 3.1 demostra o método *run()* implementado no jogo. Neste, é feito o controle do tempo de execução em nanosegundos a fim de se obter uma melhor resolução de tempo. A cada ciclo é calculado o diferencial dos tempos de execução para que se obtenha uma taxa fixa de atualização próxima a 60 quadros por segundo.

```
1
2     public void run() {
3         long tempoanterior = System.nanoTime();
4         final double fps = 60.0;
5         double nanosegundos = 1000000000/fps;
6         double controle = 0;
7         int updates =0;
8         int frames = 0;
9         long timer = System.currentTimeMillis();
10
11        while(running){
12            long agora = System.nanoTime();
13
14            if(controle >=1){
15                atualiza();
16                updates++;
17                controle--;
18            }
19
20            desenha();
21            frames++;
22            controle+= (agora-tempoanterior)/nanosegundos;
23            tempoanterior=agora;
24
25            if(System.currentTimeMillis()-timer>1000){
26                timer +=1000;
27                System.out.println(updates+" Quadros, Fps "+ frames);
28                updates = 0;
29                frames=0;
30            }
31        }
32
33        stop();
34    }
```

Código 3.1: Código para execução do *Loop*.

Dentro do *loop*, a cada iteração, são chamados o método *atualiza()* (linha 15 do código) 3.1 , responsável por executar procedimentos que vão calcular todas as ações do jogo como entrada de vídeo, posicionamento dos veículos e itens do cenário, e o método *desenha()* (linha 20 do código) 3.1 que tem o objetivo de transformar toda essa informação que foi processada em algo visual para o jogador.

3.4.2 Composição do Cenário

A composição de fundo do cenário é feita através da sobreposição de imagens estáticas que compõem o céu, as nuvens, montanhas e florestas.

Durante a execução da partida a posição destas imagens na tela são deslocadas para criar a ilusão de movimento no ângulo de visão da câmera. Esta movimentação ocorre quando existirem curvas na pista para assim dar mais vida ao cenário.

Também é apresentado ao jogador um pequeno volante no canto inferior esquerdo da tela, o qual tem a função de girar quando detectada a intenção do jogador através da movimentação do quadrilátero, sendo assim apenas mais uma forma visual de mostrar para o jogador que a detecção está ocorrendo corretamente.

A Pista

O desenho da pista, foi criado baseando-se em um efeito utilizado em alguns jogos da segunda geração de consoles, onde a representação da pista em um plano de duas dimensões dá ao jogador a impressão de uma falsa terceira dimensão, acrescentando profundidade à pista. Segundo GORENFELD (2013) isso é possível utilizando matemática para calcular diferentes graus de projeção dos itens do cenário sobre a pista.

Na Figura 3.9 pode-se verificar como são apresentados ao jogador os segmentos de um trecho de pista, onde cada seção de um segmento está ligado à sua próxima seção e a escala entre um segmento e outro dá a impressão de profundidade.

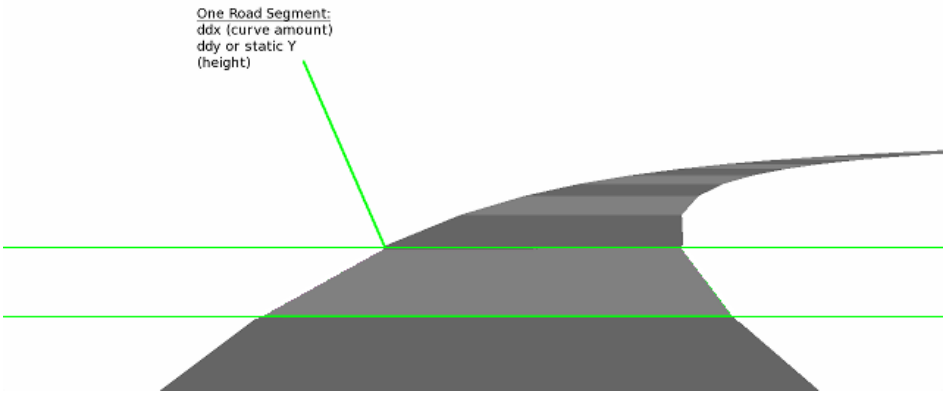

Figura 3.9: Composição da pista na tela.
Fonte GORENFELD (2013).

Ainda segundo GORENFELD (2013), esta técnica foi bastante difundida por ter sido usada em jogos clássicos como *Road Rash* e *Test Drive II: The Duel*, onde a pista é feita de segmentos poligonais. No entanto, em vez de mover no espaço *3D* completo, apenas se move em direção à câmera. Para as curvas, existe apenas uma distorção para esquerda ou direita não havendo assim uma rotação real da câmera.

O Jogador

A classe *Jogador* fica responsável por carregar e gerenciar as imagens que serão apresentadas do jogador, assim como atualizar as coordenadas a serem exibidas na tela. No Código 3.2 é mostrado o construtor da classe *Jogador* onde são carregados os *sprites*[1] do carro e há um ajuste do seu tamanho em relação à dimensão de tela utilizada no jogo.

```
1
2       public Jogador(int largura,int altura){
3           /* Carrega sprite do carro */
4           carregaCar();
5           this.largura=largura;
6           /* coloca as dimensoes do carro em escala proporcional
7            * a tela */
8           x=largura/2-carJogador[1].getWidth(null)/2+50;
9           y=altura-altura*45/carJogador[1].getHeight(null)-30;
10          larg=largura*55/carJogador[1].getWidth(null);
11          alt=altura*50/carJogador[1].getHeight(null);
12          /* Indica se o jogador esta movimentado o carro
13           * para direita ou esquerda */
14          emuso=false;
15      }
```

Código 3.2: Construtor da classe Jogador.

Tela da Câmera

A classe *Camera*, tem como objetivo delimitar na tela de exibição do jogo uma área para o jogador visualizar as imagens obtidas pela *webcam* e desenhá-las na tela juntamente com os delimitadores de área, os quais mostram ao jogador o objeto quadrilátero rastreado. Este é apresentado ao jogador sobreposto por uma área de mesmo tamanho na cor azul, como forma de orientar o jogador sobre a correta detecção do objeto.

Na Figura 3.10 é mostrada a imagem do jogador sobreposta pelos marcadores que indicam o rastreamento do objeto e os delimitadores de posições para informar que o carro deve virar para a direita ou esquerda da estrada, conforme o ângulo de inclinação do quadrilátero.

[1]Objeto gráfico bidimensional que se move na tela.

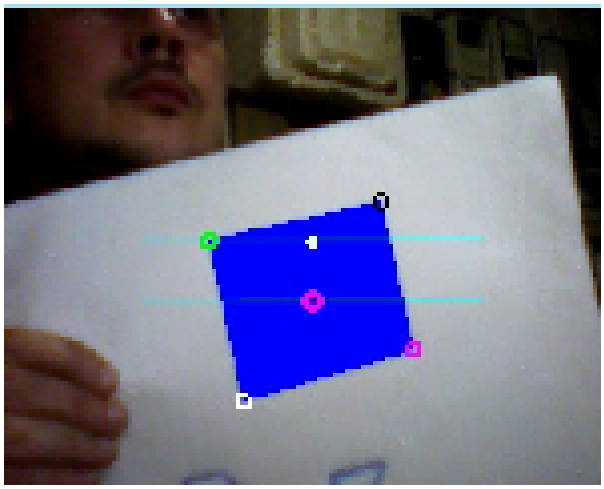

Figura 3.10: Tela da Câmera.
Fonte: Autoria Própria

3.4.3 Webcam

O Código 3.3 pertencente ao construtor da classe *Webcam* e é responsável por se comunicar com a *câmera* e gerar as imagens que são mostradas na classe *Camera* 3.4.2.

Esta classe trabalha em conjunto com os recursos da biblioteca *OpenCv* para acesso à câmera e para rastreamento do objeto em posse do jogador.

```
1
2           // Tenta se comunicar e iniciar a webcam.
3           capture= new VideoCapture(0);
4
5           if(capture.isOpened()){
6           // Caso consiga configura a altura e largura de cada quadro
                capturado.
7           capture.set(Highgui.CV_CAP_PROP_FRAME_WIDTH,300);
8           capture.set(Highgui.CV_CAP_PROP_FRAME_HEIGHT,300);
9           }
```

Código 3.3: Construtor da classe Webcam.

Após a conexão estabelecida com a câmera cada quadro é capturado e tratado individualmente através de quatro métodos da biblioteca *OpenCV*, que no jogo terá o objetivo de rastrear em tempo real objetos quadriláteros que estão na mão do jogador, sendo estes métodos relacionados abaixo:

- *Imgproc.cvtColor*
- *Imgproc.adaptiveThreshold*
- *Imgproc.findContours*

- *Imgproc.approxPolyDP*

Conforme documentação disponível no site OPENCV (2015) O primeiro método tem a função de converter a escala de cores do quadro capturado, neste caso a imagem é convertida para uma escala de cinza.

O próximo passo é a binarização da imagem, como mostrado na seção 2.6. Este método possui parâmetros que são ajustados para um maior o menor valor de corte na binarização da imagem.

Após a binarização com as bordas evidenciadas, o método *findContours* cria uma lista contendo informações do *pixels* de cada contorno.

O último método *approxPolyDP*, analisa a lista de contornos e é feita sua simplificação. Segundo OPENCV (2015), o método se baseia na utilização do algoritmo de *Douglas-Peucker* de aproximação de polígonos, facilitando a busca de vértices que formam ângulos de ligação entre si, criando assim uma lista de regiões candidatas a conter o formato de interesse. Pode-se então verificar se em um contorno há quatro ligações que formam um quadrilátero.

A partir desta análise é possível extrair diversas informações do objeto rastreado, como as coordenadas x,y de cada canto do quadrilátero, sendo assim possível calcular o seu ponto central, e, através da variação da posição dos cantos direito e esquerdo superiores do quadrilátero em relação à coordenada y do ponto central, é possível verificar se o mesmo está sendo girado e em qual direção.

A última tarefa desta classe é desenhar os delimitadores e um quadrilátero evidenciando o que está sendo rastreado, diretamente sobre a imagem que será exibida na classe *Camera* 3.4.2. Por último, é necessário informar a classe jogador (descrita na seção 3.4.2) se haverá incremento ou decremento em relação à sua posição na tela.

3.4.4 Áudio

Para Fan and Ries (1996), o som é uma parte integrante para uma experiência multimídia completa. Na linguagem *Java* há uma facilidade na inserção de sons, pois podem ser representados como objetos *AudioClip*, uma interface definida no pacote *java.applet*.

Há duas etapas envolvidas na utilização de um *AudioClip*: carregar um *AudioClip* e tocar o *AudioClip* Fan and Ries (1996).

A classe *Audio*, apresentada no Código 3.4, é responsável por tocar o áudio através do método *tocaraudio* (linha 4 Código 3.4), que recebe como parâmetro o caminho para o arquivo de áudio a ser tocado. O método *parar* (linha 13 Código 3.4) interrompe a execução do som quando solicitado.

```
1
2       public class Audio {
3           public Clip clip;
4           public void tocaraudio(File Sound){
5               try {
6                   clip = AudioSystem.getClip();
```

```
7                   clip.open(AudioSystem.getAudioInputStream(Sound));
8                   clip.start();
9               } catch (Exception e) {
10                  e.printStackTrace();
11              }
12          }
13          public void parar(){
14                clip.stop();
15          }
16      }
```

Código 3.4: Classe para execução de áudios.

O áudio é instanciado pela classe *Fase*, ou seja, o som é inicializado com a partida e prossegue em *loop* até o término da mesma. Durante a colisão, o som de fundo para, ativando um som de batida, informando assim o fim da partida.

3.4.5 Fim de Jogo

A Figura 3.11 indica ao jogador o fim da partida. Esta é mostrada sempre que for detectada a colisão entre os carros que estão distribuídos aleatoriamente no arranjo do objeto *Pista* e o carro do jogador.

Figura 3.11: Tela indicando o fim da partida.
Fonte: Autoria Própria

A colisão é tratada através da classe *TestaColisao*, usando a técnica conhecida como *Bounding box* descrita por http://devmag.org.z (2009), que trata os *sprites* como se estivessem dentro de uma caixa limitadora que é testada constantemente para ver se houve intersecção com outros *sprites* do jogo.

No Código 3.5, o método *colisao* usa um simples algoritmo para testar se há intersecção entre as coordenadas dos objetos – neste caso o *sprite* do jogador, com o *sprite* do adversário – recebendo como parâmetro as coordenadas de localização x, y, largura e altura dos objetos a serem comparados, retornando uma variável *booleana* indicando se houve ou não a colisão;

```
1
2       public class TestaColisao {
3           public boolean colisao(int obj1X, int obj1Y, int obj1W, int obj1H,int
                obj2X, int obj2Y, int obj2W, int obj2H){
4               if ((obj1X >= obj2X && obj1X <= obj2X + obj2W) && (obj1Y >= obj2Y
                    && obj1Y <= obj2Y + obj2H)){
5                   return true;
6               } else if ((obj1X + obj1W >= obj2X && obj1X + obj1W <= obj2X +
                   obj2W)&&(obj1Y >= obj2Y && obj1Y <= obj2Y + obj2H)){
7                   return true;
8               } else if ((obj1X >= obj2X && obj1X <= obj2X + obj2W)&& (obj1Y +
                   obj1H >= obj2Y && obj1Y + obj1H <= obj2Y + obj2H)){
9                   return true;
10              } else if ((obj1X + obj1W >= obj2X && obj1X + obj1W <= obj2X +
                   obj2W)&&(obj1Y + obj1H >= obj2Y && obj1Y + obj1H <= obj2Y +
                   obj2H)){
11                  return true;
12              } else {
13                  return false;
14              }
15          }
16      }
```

Código 3.5: Classe para detectar colisão.

Com a detecção da colisão, uma tela final é exibida. Nesta tela estão presentes informações sobre o placar e o maior recorde ja registrado 3.4.6. Eventualmente, caso o jogador atinja um recorde maior, uma frase nesta mesma tela o parabeniza pelo feito. Clicando no botão *Ok* desta mesma tela o jogador é levado para tela inicial, tendo a possibilidade de iniciar uma nova partida ou encerrar o *game*.

3.4.6 O Placar

O placar é apurado pela distância total, determinada através da quantidade de seguimentos de pista que o jogador conseguiu percorrer até o momento em que é detectada uma colisão. Quando esta ocorre, um objeto da classe *RegistraPlacar* é criado, como pode-se ver no construtor do Código 3.6. Um arquivo de texto contendo os placares das jogadas anteriores é carregado, e, o método *retornaMaior* (linha 6 do Código 3.6) faz uma busca do melhor placar a fim de comparar com o valor atual e informar ao jogador se um novo recorde foi quebrado.

Por fim, o placar do jogador é inserido no final do arquivo de placares.

```
1
2       public RegistraPlacar(){
3           ArrayList<String> resultado= new ArrayList<String>();
4           arquivo=criaArquivo();
5           resultado = carregaArquivo(arquivo);
6           maior = retornaMaior(resultado);
7
8       }
```

Código 3.6: Classe para registro do placar.

Capítulo 4

Testes

Após a conclusão do jogo foram realizados diversos testes utilizando uma *webcam* com resolução de imagem de 640 x 480 *pixels*. Estes testes foram de suma importância para correção de *bugs* e refinamento no processamento de imagem, podendo assim obter um rastreamento do item desejado com maior precisão.

Em todos os testes o jogo não apresentou quedas significativas de *frames*, assim, o jogo fluiu corretamente durante toda a partida mantendo uma taxa de 60 *Frames por segundos (FPS)*.

Em relação ao rastreamento do objeto, ficou claro que deve haver uma padronização para que ocorra a identificação do quadrilátero, tendo em vista que ao processar a imagem para sua posterior binarização, o valor de corte utilizado pelo método *Imgproc.adaptiveThreshold* é fixo não havendo ainda possibilidade de alterá-lo em tempo de execução. Desta forma, fica impossível que o jogador utilize qualquer tipo de objeto pois suas cores, incidência de luz, manchas sobre ele ou até mesmo pelo simples fato do jogador colocar os dedos sobre o objeto impede que haja o rastreamento efetivo do mesmo.

Para contornar esse problema optou-se em desenhar retângulos com cores escuras sobre uma folha de papel mais claro, como pode ser observado na Figura 4.1. Desta forma, foi possível ajustar previamente o valor *Threshold*, tornando possível jogar uma partida inteira do *game* sem perder o foco abruptamente do quadrilátero, salvo exceção de movimentos muito rápidos, distanciamento ou aproximação excessiva da câmera durante a execução da partida.

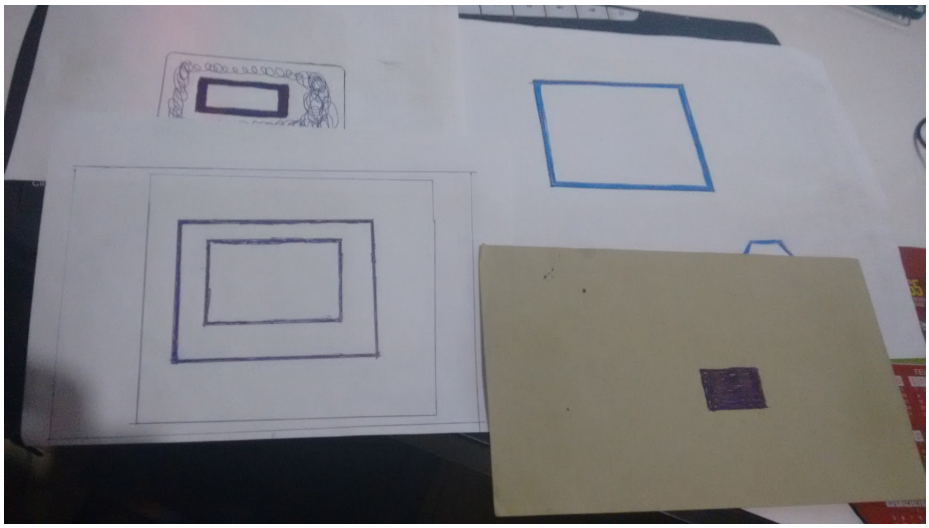

Figura 4.1: Diversos desenhos de retângulos utilizados para rastreamento durante a partida.
Fonte: Autoria Própria

Capítulo 5

Conclusão

Atualmente, jogos eletrônicos são cada vez mais comuns e não perdem em nada para outras plataformas de entretenimento. Estes também podem ser usados como ferramenta para estimulo à educação e é extremamente gratificante conseguir conciliar o desenvolvimento de *games* com técnicas de visão computacional, para simplesmente dar maior liberdade e imersão ao jogador, ou podendo até mesmo servir como apoio no de desenvolvimento motor e cognitivo em casos de educação especial.

No desenvolvimento deste projeto foi possível verificar que ainda há uma falta de recursos literários muito grande, principalmente se tratando do desenvolvimentos de jogos utilizando visão computacional quando se deseja focar em uma linguagem mais especifica.

As ferramentas escolhidas para desenvolvimento deste projeto, como a linguagem *Java* utilizando a plataforma *Eclipse* é bem completa e compatível com a biblioteca *OpenCV*, porém, a falta de documentação do uso das funções mais complexas disponíveis na biblioteca para esta linguagem de programação tornam sua compreensão e adaptação mais complexas.

Apesar destas dificuldades e limitações apresentadas em capítulos anteriores, os objetivos traçados para este projeto foram alcançados de maneira satisfatória, porem este permanece aberto para estudo e possíveis atualizações, como criação de novas fases, inclusão de sons, novos cenários, aprimoramento na técnica utilizada para rastreamento dos objetos, criação de placar personalizado com armazenamento de informações em banco de dados e reconhecimento facial do jogador.

Referências Bibliográficas

ANDREW, D. (2005). *Killer Game Programming in Java*. O'Reilly Media.

BARBOZA, J. L. L. (2004). Processamento de imagens aplicado na monitoração de processos.

BRADSKI, G. . K. (2008). *Learning OpenCV*. O'Reilly.

DEITEL, H. M. and DEITEL, P. J. (2010). ***Java - Como Programar***. Pearson Education Do, 8 edition. 1114 P.

Fan, JOEL; Tenitchi, C. and Ries, E. (1996). *Black Art of Java Game Programming*. Waite Group Press.

FORSYTH, D. A. and PONCE, J. (2012). *Compter Vision a Modern Approach, Second edition*. Pearon.

GONZALEZ, Rafael; WOODS, R. E. (2002). *Digital Image Processing (2nd Edition)*. Prentice Hall.

GORENFELD, L. (2013). Lou's pseudo 3d page. `http://www.extentofthejam.com/pseudo/`, acesso em janeiro de 2016.

http://devmag.org.z (2009). Basic collision detection in 2d. `http://devmag.org.za/2009/04/13/basic-collision-detection-in-2d-part-1/`, acesso em Janeiro de 2016.

MARQUES FILHO, Ogê; VIEIRA NETO, H. (1999). *Processamento Digital de Imagens*. Brasport.

OPENCV (2015). Opencv 2.4.2. `http://docs.opencv.org/java/2.4.2/`, acesso em Agosto de 2015.

pointdaarte.webnode.com.br (2011). História da fotografia. `http://pointdaarte.webnode.com.br/news/a-historia-da-fotografia/`, acesso em Julho de 2015.

SILVA, A. M. e. (2011). Curso processamento digital de imagens de satélite. `http://www.cartografia.org.br`, acesso em Julho de 2015.

www.incinerrante.com (2015). Vista da janela em le gras. http://www.incinerrante.com/textos/vista-da-janela-em-le-gras-1826-7-de-joseph-nicephore-niepce#axzz3p3XZr4hE, acesso em Julho de 2015.