UNIVERSIDADE ESTADUAL PAULISTA

Faculdade de Ciências

Departamento de Computação

***Campus* de Bauru**

TRABALHO DE CONCLUSÃO DE CURSO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

"Desenvolvimento de um *Game* para Android O.S."

Aluno: Francisco Mariani Guariba Filho

Orientador: Prof. Dr. Eduardo Morgado

Bauru

2012.

Francisco Mariani Guariba Filho

Desenvolvimento de um *Game* para Android OS

Orientador: Prof. Dr. Eduardo Morgado

Monografia apresentada junto à disciplina Projeto e Implementação de Sistemas II, do curso de Bacharelado em Ciência da Computação, Faculdade de Ciências, Unesp, *campus* de Bauru, como parte do Trabalho de Conclusão de Curso.

UNESP

2012

Francisco Mariani Guariba Filho

Desenvolvimento de um *Game* para Android OS

Monografia apresentada junto à disciplina Projeto e Implementação de Sistemas II, do curso de Bacharelado em Ciência da Computação, Faculdade de Ciências, Unesp, *campus* de Bauru, como parte do Trabalho de Conclusão de Curso.

BANCA EXAMINADORA

Prof. Dr. Eduardo Morgado
Professor Doutor
DCo – FC - UNESP – Bauru
Orientador

Profª. Drª. Simone das Graças Domingues Prado
Professora Doutora
DCo – FC - UNESP – Bauru

Prof. Dr. Antonio Carlos Sementille
Professor Doutor
DCo – FC - UNESP – Bauru

Bauru, 12 de Novembro de 2012.

Dedico esse trabalho a minha família, por me apoiar incondicionalmente em todos os momentos importantes da minha vida e da minha formação.

RESUMO

FILHO, F. M. G. Desenvolvimento de um *Game* para Android OS.
Monografia – Curso de Bacharelado em Ciência da Computação, Universidade Estadual Paulista "Júlio de Mesquita Filho". Bauru, 2012.

Com o avanço da tecnologia, os dispositivos móveis têm se tornado cada vez mais potentes e passaram a ter funções que vão além de realizar chamadas telefônicas. Novos aplicativos, que desempenham estas novas funções foram lançados. O mercado de entretenimento digital passou a ser um dos mais beneficiados com isso, já que os *games* passaram a ser um dos aplicativos mais utilizados por diversos tipos de usuários. Esta monografia apresenta o desenvolvimento de um game para a plataforma Android, levando em consideração aspectos relevantes, tais como: física, cenários e animação de personagens.

PALAVRAS-CHAVE: GAME, ANDROID OS, DISPOSITIVOS MÓVEIS, ANDENGINE.

ABSTRACT

FILHO, F. M. G. A Game Development for Android OS.
Monografia – Curso de Bacharelado em Ciência da Computação, Universidade Estadual Paulista "Júlio de Mesquita Filho". Bauru, 2012.

With the technology advancement, mobile devices have become increasingly more powerful and have started to have functions beyond of making phone calls. New applications, which perform these new functions, have been launched. The digital entertainment market has become one of the most benefited with this, once the games have become one of the most used applications for various types of users. This monograph presents the development of a game for Android OS, considering concepts like: physics, scenarios/views and character animation.

KEY-WORDS: GAME, ANDROID O.S., MOBILE DEVICES, ANDENGINE.

LISTA DE FIGURAS

LISTA DE CÓDIGOS

LISTA DE SIGLAS

ADT	*Android Development Tools*
SDK	*Software Development Kit*
IDE	*Integrated Development Environment*
API	*Application Programming Interface*
OpenGL	*Open Graphics Library*
WAP	*Wireless Application Protocol*
J2ME	*Java Plataform Micro Edition*
BREW	*Binary Runtime Enviroment For Wireless*
OHA	Open Handset Alliance
EDGE	*Enhanced Data rates for GSM Evolution*
JVM	*Java Virtual Machine*
DVM	*Dalvik Virtual Machine*
SIP	*Session Initiation Protocol*
JSR	*Java Specification Request*
ASF	*Apache Software Foundation*
UML	*Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

O avanço da tecnologia requer respostas imediatas, facilidade de interação com os novos dispositivos e uma resposta audiovisual que agrade e corresponda às expectativas das novas gerações.

A geração “Z”, as pessoas nascidas desde o início dos anos 1990 até hoje, amadureceu imersa nesta tecnologia digital e não necessita de papel e lápis para se comunicar.

Segundo Liu (2012), particularmente, os telefones móveis percorreram um longo caminho desde o seu início em 1980. Os aparelhos de hoje são uma fração de seus antepassados quanto ao tamanho, com aumento exponencial de desempenho. Os dispositivos modernos são capazes de suportar *softwares* cada vez mais complexos, interfaces inovadoras e capacidades de rede, sendo o setor de *games* o mais beneficiado com isso.

O primeiro jogo com grande repercussão no cenário mundial foi o Serpente Nokia (*Snake* Nokia) no ano de 1997. Como todos os telefones móveis da época, o jogo consistia em nada mais do que pontos e linhas, mas foi certamente o mais viciante jogo portátil desde *Tetris*. Apesar de sua abordagem minimalista, *Snake* mudou a funcionalidade do telefone celular para sempre. Uma série de jogos foi seguida por ele, contudo os programadores foram prejudicados por uma geração limitada de *hardware* fornecida na época.

O avanço da tecnologia WAP (*wireless application protocol*) teve grande impacto no desenvolvimento de jogos para dispositivos móveis. Foi possível não apenas realizar o *download* dos tradicionais *games* pela *internet*, como também foi oferecido suporte ao modo *multiplayer* (múltiplos jogadores). Os primeiros títulos WAP geralmente tomaram aventuras na forma de texto. A partir de então a tecnologia avançou rapidamente e os jogos tiveram um grande impulso no mercado global.

Serviços como J2ME (plataforma Java para embarcados), BREW (*Binary Runtime Environment for Wireless*) e séries de sistemas operacionais da Nokia como 40 e 60 tomaram corpo através do protocolo *wireless*. Essas plataformas permitiram distribuir programas mais complexos, atraindo a atenção de empresas de grande cunho na área de *games*. Na virada do milênio, os novos

aparelhos invadiram o mercado, suportando Java, Flash e outros recursos robustos *on-line*. Empresas famosas de jogos eletrônicos como *Sega, Namco*, *THQ*, *EA* e *Gameloft* deram o primeiro passo, destacaram-se pelo recurso antes inexplorado, desenvolvendo conteúdo exclusivo para dispositivos móveis.

Em 2003, a Nokia lançou o N-Gage, um dispositivo móvel, a fim de atrair os usuários do *Gameboy Advance*, um videogame portátil lançado pela Nintendo na época. Apesar de ser consideravelmente mais potente, ao oferecer comunicação e suporte *on-line*, foi um fracasso comercial. O *design* e os botões pouco agradáveis e adequados para os *games* foram alguns dos motivos. No entanto, o serviço *on-line* que a Nokia lançou como apoio ao dispositivo estava à frente do seu tempo e serviu para que a geração de *smartphones* seguinte anunciasse uma nova era de jogos para celulares.

Aparelhos como o *BlackBerry* e o N5 Nokia elevaram a geração de *smartphones* para um novo nível, suportando gráficos 3D, interfaces inovadoras e capacidades de rede extensiva. Em 2007, a Apple lançou sua primeira geração, o *iPhone*. Com a funcionalidade do sistema *touch-screen* e o suporte acelerômetro ativado, os desenvolvedores aproveitaram os controles de movimento e desenvolveram interfaces a partir do zero. Apesar disso, as setas direcionais virtuais (*virtual d-pads*) e interfaces de inclinação foram implementadas com variados graus de sucesso. Essa flexibilidade pavimentou o caminho para os desenvolvedores trazerem todos os gêneros de *games* para a plataforma.

Um dos principais fatores para o sucesso do *iPhone* como plataforma de jogos é a *App Store*. Lançada no verão de 2008, a loja digital agora abriga os jogos de todas as formas e tamanhos. O serviço facilitou aos editores de jogos distribuírem seus produtos. No entanto, não são apenas os grandes estúdios que têm colhido os frutos de especificações do *iPhone* e base de usuários. Desenvolvedores independentes descobriram um grande público para seus títulos livres de orçamento. A Rovio, desenvolvedora do jogo *Angry Birds*, é um bom exemplo de empresa que cresceu devido ao destaque do mercado iOS.

Segundo De Cássia (2012), no inicio de 2012, a desenvolvedora finlandesa de games Rovio afirmou que seu principal jogo, o *Angry Birds*, foi baixado mais de um bilhão de vezes desde o lançamento, em dezembro de

2009. Isso inclui usuários que utilizam *smartphones* e *tablets*. Andrew Stalbow, vice-presidente da companhia, voltou a público para divulgar novos números do título: mensalmente, 200 milhões de pessoas jogam *Angry Birds*.

O *iPhone* não é o único *smartphone* equipado para jogos. *Google Nexus One*, Motorola DEXT, o *Windows Phone 7* da *Microsoft* e a linha *Samsung Galaxy* são exemplos de *smartphones* que são igualmente capazes de lidar com os títulos *high-end*. A inauguração da pilha de *software* Android e o *Android Market* em 2008 abriram novas possibilidades para os programadores, oferecendo mais flexibilidade do que a loja da Apple desde que os desenvolvedores poderiam distribuir seus jogos através de qualquer plataforma que desejar. Em dezembro de 2011, o *Android Market* atingiu a marca de 200.000 aplicações.

Jogos móveis já percorreram um longo caminho desde os dias de *Snake*. O surgimento da interface *touch-screen*; um *display* ou campo que capta a presença e a localização do toque em sua superfície; o suporte de redes e os serviços *on-line* ajudaram a tornar a atividade um dos setores de maior crescimento da indústria. Consequentemente empresas de todos os tamanhos têm investido no ramo e a diferença entre as ferramentas de comunicação e as plataformas de jogos tem diminuído desde que os prestadores de serviço decidiram incluir jogos em seus celulares.

Ao considerar este avanço na tecnologia e no mercado de aplicativos para dispositivos móveis, foi amadurecida a ideia de desenvolver um jogo de plataforma bidimensional que seguisse o padrão dos jogos lançados para dispositivos móveis.

O projeto se baseia, portanto, em um jogo para o sistema operacional Android, no qual o usuário controla um personagem por cenários com inimigos, o que permite a exploração dos principais conceitos no desenvolvimento de jogos e de aplicativos para Android. Será melhor explicado nos próximos itens.

1.1.1 Objetivos do Trabalho

Conforme mencionado anteriormente pode-se dizer que devido ao forte crescimento do mercado *móbile* e à ascensão de empresas desenvolvedoras de *games*, o objetivo principal do trabalho pôde ser definido.

Explicitamente, o objetivo deste trabalho consiste em desenvolver um jogo eletrônico de plataforma bidimensional para o sistema operacional Android. Portanto, pode-se dizer mais especificamente que se trata de desenvolver um projeto a partir de uma ideia para criação de um jogo e aprender sobre as principais técnicas e ferramentas para o desenvolvimento de jogos e aplicativos a este sistema operacional.

2 FUNDAMENTAÇÃO TEÓRICA

Neste item serão apresentadas as principais características e teorias referentes ao sistema operacional Android e ao desenvolvimento de jogos em duas dimensões.

Inicia-se com uma explicação sobre o funcionamento do sistema operacional do Google e, então, explicam-se os princípios teóricos e técnicos que foram utilizados ao desenvolver o jogo eletrônico. Assim, considera-se as características da *AndEngine*, uma *engine* própria ao desenvolvimento de jogos bidimensionais para Android, que conta com uma série de extensões, dentre elas, a extensão *AndEnginePhysicsBox2D*, que também foi utilizada no desenvolvimento do jogo.

Por fim, explicam-se alguns conceitos de engenharia de *software* relacionados à modularização de sistemas e ao uso de UML que também foram utilizados no projeto.

2.1 Android O.S.

A seguir serão tratados o surgimento do sistema operacional Android, os principais recursos que ele oferece e a sua estrutura.

2.1.1 OHA

O Android surgiu da parceria da Google com a *Open Handset Alliance* (*OHA*), aliança na qual figuram os principais atores do mercado móvel mundial (SCHEMBERGER; FREITAS; VANI, 2009, p.2).

A OHA consiste em um grupo de todas as estruturas envolvidas no processo de telefonia móvel:

- Operadoras de celular: responsáveis pela conexão. Fornecem o serviço para o usuário final;
- Fabricantes de aparelhos: responsáveis pela criação do *hardware*;

- Empresas de semicondutores: criam os *chips* dos aparelhos de celulares;
- Empresas de *software*: desenvolvem os *softwares* que serão executados no Android;
- Empresas de comercialização: são as empresas responsáveis pela divulgação, *marketing* e comercialização dos produtos para o usuário. (Pereira; Da Silva; 2009 p.2).

2.1.2 O Projeto Android

O Android é uma plataforma completa para tecnologia móvel, envolvendo um pacote com programas para celulares, já com um sistema operacional, *middleware*, aplicativos e interface do usuário. É o primeiro projeto de uma plataforma *open source* para dispositivos móveis em conjunto para construir aplicações móveis inovadoras (Pereira; Da Silva; 2009 p.3).

Conforme Schemberger, Freitas e Vani (2009, p.2) demonstram, o Android conta também com um SDK que disponibiliza as ferramentas e APIs necessárias para o desenvolvimento na plataforma Android, usando a linguagem JAVA.

As principais funcionalidades da plataforma são:

- *Framework* de desenvolvimento de aplicações: reutilização de código e facilidade de acesso a recursos exclusivos e manutenção;
- Nova máquina virtual (dalvik): criada e otimizada para dispositivos móveis e as suas limitações;
- Navegador web integrado: baseado no projeto *open source webkit* – o mesmo do iPhone e Nokia série 60;
- Biblioteca de gráficos otimizada para dispositivos móveis: exclusiva biblioteca para gráficos 2D e 3D baseada na especificação *OpenGL* ES 1.0, com aceleração por *hardware* como opcional;
- *SQLite*: armazenamento de dados estruturados;

- Suporte multimídia: compatibilidade com os principais formatos existentes, entre eles mpeg4, h.264, mp3, .aac, .amr, .jpg, .png e .gif;
- Telefonia com tecnologia GSM: as aplicações podem manipular operações telefônicas, caso o fabricante permita esse acesso;
- *Bluetooth*, *EDGE*, *3G* e *WiFi*: foco nas principais tecnologias de transmissão de dados sem fio, também depende da permissão do fabricante para acesso;
- Câmera e GPS: ter o celular como uma ferramenta para interação com redes sociais, também dependente da permissão do fabricante para acesso;
- Ambiente de desenvolvimento com *plug-in* para Eclipse: inclui emulador, ferramentas para *debug* e supervisão de memória e desempenho.

2.1.3 Arquitetura

A arquitetura da plataforma Android é dividida em quatro camadas: *Kernel* GNU Linux, bibliotecas, *framework* para aplicações e as próprias aplicações – além da porção *runtime*, necessária para a execução dos aplicativos no dispositivo.

Segundo Schemberger, Freitas e Vani (2009, p.3), a camada mais baixa da arquitetura, *Kernel* Linux, é a responsável por gerenciar os processos, *threads*, arquivos e pastas, além de redes e *drivers* dos dispositivos. É responsável por gerenciar todos os processos e a memória.

A camada *Libraries* possui as bibliotecas *C/C++* que são utilizadas pelo sistema, e também bibliotecas de multimídia, funções para navegadores *web*, para gráficos e fontes, além de funções de aceleradores de *hardware*, de renderização 3D e de acesso a banco de dados *SQLite*.

A camada de aplicação é o local dos aplicativos executados sobre o sistema operacional. Enquanto a camada *Runtime* é responsável por instanciar a máquina virtual, Dalvik, criada para cada aplicação executada no Android e explicada melhor no próximo item.

A figura 1, a seguir, apresenta a organização da arquitetura Android.

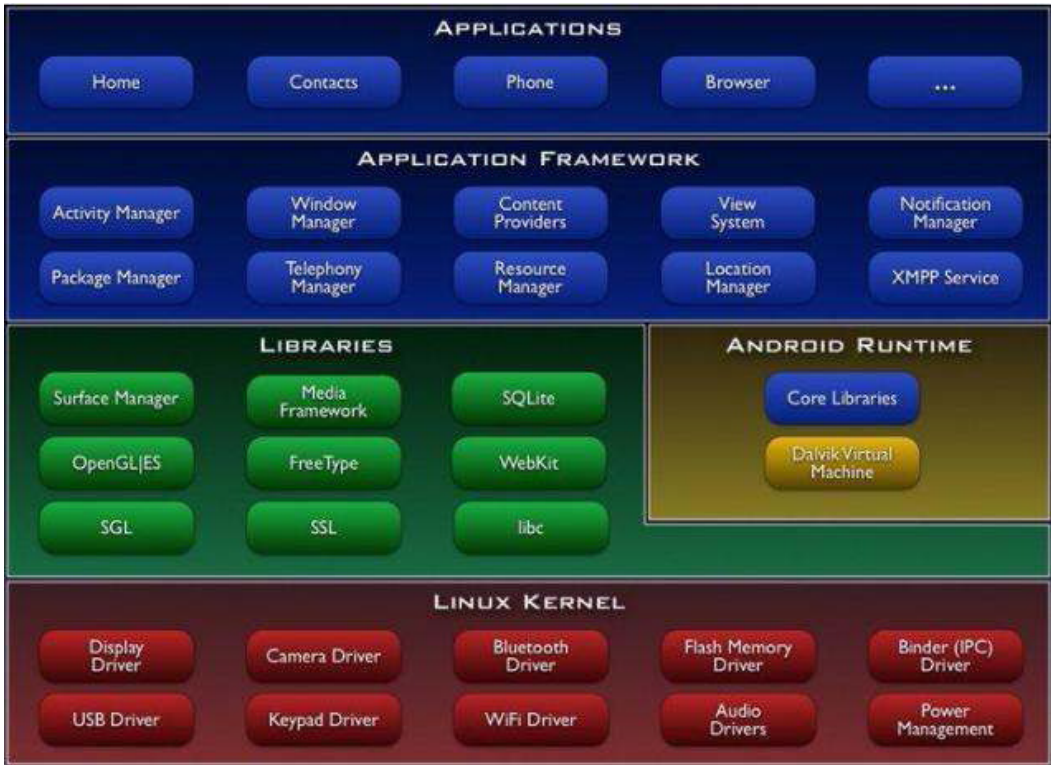

Figura 1: Arquitetura da Plataforma Android.

2.1.4 Máquina Virtual Própria

A base de desenvolvimento do Android é realizada normalmente em Java. Porém, no seu sistema operacional não existe uma Máquina Virtual Java (JVM), mas sim outra, a DVM (*Dalvik Virtual Machine*), otimizada para interpretar códigos em dispositivos móveis.

Ao desenvolver aplicações para o Android, utiliza-se a linguagem Java e todos os recursos que esta oferece. Mas, ao compilar o *bytecode* (.class), este é convertido para o formato específico da máquina virtual Dalvik, o .dex (*Dalvik Executable*), que representa a aplicação do Android compilada.

De modo análogo aos aplicativos em Java, depois da aplicação Android ter sido compilada, juntamente com outras bibliotecas e possíveis imagens utilizadas é gerado um arquivo único. Neste caso com a extensão APK (*Android Package File*), que é a aplicação pronta para ser instalada.

Se o ambiente de desenvolvimento utilizado for o Eclipse, toda essa compilação e geração do arquivo APK acontece automaticamente (SCHEMBERGER; FREITAS; VANI, 2009, p.3).

De acordo com o artigo escrito por Alípio (2011), a J2ME é limitada pelo motivo de alguns aparelhos não suportarem tecnologias como: gráfico 3D, SIP e Bluetooth. Isso se torna um incômodo para desenvolvedores da plataforma, visto que, dependendo do aplicativo, o mesmo não será compatível com alguns aparelhos. Outro fator importante que a Google destacou sobre não ter adotado o J2ME foi a especificação **JSR** (*Java Specification Request*) que é um processo bastante burocrático e necessário para que sejam inseridos novos recursos na plataforma. Em contrapartida, através da sua própria máquina virtual a Google possui o controle completo e pode impulsionar a adoção de novas funcionalidades em uma forma mais rápida e simples.

2.1.5 Código Aberto e Livre

Conforme Schemberger, Freitas e Vani explicitam (2009, p.3), o Google Android é a primeira plataforma completamente livre para aplicações móveis e de código aberto (*open-source*).

Para operadoras, fabricantes e usuários avançados isso é muito vantajoso, pois é possível utilizar gratuitamente o sistema operacional além da facilidade em personalizá-los.

A licença de uso é a *Apache Software Foundation* (ASF), que permite que alterações efetuadas no código-fonte não sejam obrigatoriamente compartilhadas.

O código fonte do Android está disponível no site do projeto: http://source.android.com.

2.1.6 Funcionamento de uma Aplicação Android

Antes de mostrar uma aplicação Android, é preciso conhecer e entender a anatomia de aplicações para esta plataforma, já que estas não seguem os formatos conhecidos de aplicações Java ou de um *Midlet* para Java ME.

A aplicação é dividida em quatro módulos descritos abaixo, conforme Schemberger, Freitas e Vani (2009, p.6) discutem:

- ***activity***: é o módulo mais comum de uma aplicação Android. Refere-se a uma tela da aplicação e é implementada como uma única classe que deriva da classe base *activity*. Essa classe deve exibir uma interface com o usuário e tratar os eventos a ela relacionados;
- ***intent receiver***: usado quando se deseja que a aplicação reaja a algum evento externo (o toque do telefone, internet disponível ou um alarme). Este módulo não exibe interface com o usuário, mas pode através da classe *Notificationmanager* avisar o usuário que tal evento ocorreu;
- ***service***: um *service* é o código que é executado durante toda a aplicação, sem a necessidade de uma interface com o usuário (tocadores de música ou *players* de vídeo). Após o usuário escolher qual música deseja ouvir ele não quer que sua música pare por causa de algum outro evento;
- ***content provider***: uma aplicação pode armazenar dados através de uma base *SQLite*. Este módulo permite que as diversas aplicações no dispositivo compartilharem informações.

Qualquer aplicação não precisa necessariamente implementar todos os módulos, mas provavelmente ela terá uma combinação entre dois ou mais deles.

2.2 O Ambiente Eclipse e o Android SDK

O Eclipse é um IDE que segue o modelo *open source* de desenvolvimento de *software*. O projeto Eclipse foi iniciado pela IBM que desenvolveu a primeira versão do produto, mas em novembro de 2001 teve seu código-fonte aberto. Atualmente, o Eclipse é controlado por uma organização sem fins lucrativos independente, chamada *Eclipse Foundation*.

A forte orientação ao desenvolvimento baseado em *plug-ins*, programas de computador geralmente pequenos e leves utilizados para adicionar funções a outros programas maiores, e o amplo suporte ao desenvolvedor com centenas de *plug-ins* que procuram atender as diferentes necessidades de

diferentes programadores, são fatores decisivos que induzem milhares de programadores a escolher esse ambiente de desenvolvimento, já que é possível a instalação do ADT, o *plug-in* do Android.

Além do Eclipse e do ADT instalados no computador, é também necessária a instalação do Android SDK, o kit de desenvolvimento necessário para desenvolver aplicações em Android. O *download* também pode ser feito gratuitamente.

Assim, com os três programas instalados no computador, pode-se iniciar o desenvolvimento de aplicativos para Android.

2.3 *AndEngine*

Criada em junho de 2012 e mantida por Nicolas Gramlich; engenheiro de *software* da Zynga (reconhecida empresa de jogos casuais), a *AndEngine* é uma *game engine*, ou seja, um programa e/ou conjunto de bibliotecas utilizado no desenvolvimento de jogos ou outras aplicações com gráficos em tempo real. No caso desta *engine*, sua função é simplificar e abstrair a criação de jogos para a plataforma Android. É totalmente gratuita e livre, e baseia-se na *OpenGL*, uma API – especificação utilizada como uma interface para diferentes componentes de *software* se comunicarem - *open-source* utilizada para suporte à computação gráfica e para desenvolvimento de aplicativos gráficos, ambientes tridimensionais, jogos, entre outros.

Esta *engine* cobre os aspectos mais importantes da biblioteca *OpenGL* quanto ao desenvolvimento de jogos bidimensionais, o que permite a criação de excelentes aplicativos ao utilizá-la, portanto não exige um prévio conhecimento sobre *OpenGL* e provê recursos para praticamente qualquer tipo de jogo a ser desenvolvido. Um exemplo disto é o jogo *Angry Birds* criado pela Rovio, já mencionado na introdução desta monografia, no qual foi utilizada a *AndEnginePhysicsBox2D* para o seu desenvolvimento, uma das extensões da *AndEngine* explicada melhor no item 2.6.

A *AndEngine* conta também com suporte à detecção de colisões, implementa diversas propriedades físicas, auxilia no desenvolvimento de

modos de multijogadores, de partículas e tem suporte a *multi touch*. Além de existir uma grande comunidade ativa que trocam constantemente informações sobre o uso desta *engine.*

A *AndEngineExamples* é outro projeto criado por Nicolas Gramlich que mostra o funcionamento de vários exemplos que utilizam os mais variados recursos da *AndEngine*, revelando assim o potencial desta *engine*. Obviamente, pode ser instalado nos dispositivos que possuem o Android como sistema operacional. E, o código pode ser visualizado, modificado e copiado conforme o desenvolvedor achar necessário.

A *AndEngine* não possui restrições quanto ao sistema operacional no qual será executada. Para utilizar esta *engine* é necessário:

- Realizar o *download* do código fonte;
- Gerar o arquivo .jar do projeto;
- Incluir a *engine* no caminho da localização do projeto;
- Estender *GameActivity* ao invés de *Activity*.

Uma desvantagem da *AndEngine* é a falta de documentação tanto no código, quanto em documentos que poderiam ser encontrados na internet ou em outro veículo de acesso à informação, como artigos ou sites explicativos. Isso dificulta inicialmente o entendimento do código por parte dos desenvolvedores. Apesar disso e devido à série de benefícios que esta *engine* oferece, a *AndEngine* foi o conjunto de bibliotecas escolhido para desenvolver o presente trabalho.

2.4 Principais Conceitos da *AndEngine*

Neste item serão explicados os conceitos fundamentais sobre a *AndEngine*, conceitos que são utilizados em qualquer tipo de jogo que seja desenvolvido por esta *engine.*

2.4.1 *Engine*

Como mencionado anteriormente, a *engine* é um programa utilizado no desenvolvimento de jogos. Tem como função permitir ao jogo prosseguir em pequenos espaços de tempo. É responsável por sincronizar periodicamente as

imagens e sons do jogo conforme se alteram, além de atualizar o cenário, que abrange todo o conteúdo sendo processado pelo o jogo.

2.4.2 Política de Resolução

A política de resolução de imagens é uma parte das opções da *engine.* É responsável por tratar os diversos tamanhos de tela de diferentes dispositivos. Pode ser definida por dois tipos distintos definidos como segue:

- Política de Resolução de Proporção: procura maximizar a visão da cena do jogo até o limite da tela do dispositivo, enquanto mantém a proporção especificada dos elementos do cenário. Isso significa que o os objetos não vão ficar distorcidos na tela, pois a visão do jogo ocupará o maior tamanho possível. Uma possível desvantagem é que a tela poderá apresentar algum espaço negro.
- Política de Resolução de Suficiência: procura preencher toda a resolução do dispositivo, sem levar em consideração a proporção dos elementos que estiverem no cenário, o que impede que espaços negros fiquem na tela. No entanto, as imagens podem ficar distorcidas.

2.4.3 Câmera

Uma câmera define o tamanho do cenário desenhado na tela, de modo que nem todo o cenário precise ficar visível o tempo todo. Pode, por exemplo, ser maior que a resolução do dispositivo, no qual o jogo será executado. Frequentemente é criada uma câmera para cada cena e existem duas subclasses com mais funções, descritas abaixo:

- BoundCamera: permite que haja mais de uma câmera no mesmo cenário;
- ZoomCamera: permite que o jogador possa aumentar ou diminuir o tamanho do cenário conforme desejar.

2.4.4 Cenas

Cada cenário é definido por uma classe denominada *Scene*, que funciona como um recipiente para todos os objetos a serem inseridos em um determinado cenário. A *Scene* contém uma quantidade específica de camadas, as quais podem conter uma quantia fixa ou dinâmica de entidades, e contém

subclasses como a HUD e a MenuScene que são desenhadas na mesma posição da cena, sem haver influência da posição da câmera do jogo. Abaixo segue uma melhor explicação sobre em quais partes do jogo estas subclasses são mais utilizadas:

- HUD: nos cenários do jogo é muito comum utilizar esta *Scene* para conter a pontuação do jogador e a quantidade de vidas de um personagem, por exemplo.
- MenuScene: é comum ser usada nas telas de menu, por exemplo, para conter uma lista de opções, as quais o jogador pode escolher.

2.4.5 Entidades

Uma entidade é um objeto que pode ser desenhado como imagens de *sprites* (explicados no próximo item), retângulos, textos ou linhas, as quais são representadas pelas classes *Sprite, Rectangle, Text e Line* respectivamente. Possui, também, diversas propriedades que podem ser previamente definidas, como posição, rotação, escala, cor e etc.

2.4.6 *Sprites*

Os *sprites* são objetos que representam praticamente todas as imagens adicionadas em um jogo e podem pertencer a três tipos de classes distintas explicadas a seguir:

- Classe *Sprite*: utilizada quando se quer adicionar uma imagem com um único estado, que não tenha nenhum tipo de animação, como por exemplo, uma pedra, uma árvore ou um cenário de fundo;
- Classe *TiledSprite*: utilizada quando se quer adicionar uma imagem que possui múltiplos estados. Um exemplo disso poderia ser um botão com dois estados: pressionado e normal;
- Classe *AnimatedSprite*: é uma subclasse da *TiledSprite* e possui, portanto, mais funções que possibilitam a realização de animação desses estados. Os exemplos seriam os personagens adicionados no jogo, os quais possuem diferentes estados como “atacando”, “andando”, “pulando” e etc.

No caso de utilizar a classe *AnimatedSprite*, deve- se adicionar uma imagem, denominada *sprite sheet,* que é representada por uma série de quadros, cada um contendo uma imagem diferente do mesmo objeto a ser adicionado. Assim, as diferentes ações desse objeto poderão ser representadas quando o programador desejar.

No caso da figura 2, representada a seguir, os quadros de 0 a 5 podem indicar o caminhar do personagem no sentido de cima para baixo, enquanto os quadros de 6 a 11 representam o caminhar da direita para a esquerda e assim por diante. Portanto, o programador pode definir os quadros mostrados conforme uma determinada ação e vale mencionar que nem todos os quadros precisam necessariamente ser utilizados.

Figura 2: Exemplo de um *sprite sheet.*

2.4.7 Texturas e Regiões de Texturas

Uma textura é a imagem armazenada na memória do *chip* gráfico do dispositivo, enquanto as regiões de texturas são recipientes para conter as texturas. Portanto, as regiões de texturas são utilizadas pelos *sprites* para permitir que o sistema entenda qual parte da textura se deseja mostrar, ou seja, qual *tile* do *tilesheet* deve ser mostrado.

2.5 *PhysicsBox2D*

Além da *AndEngine*, foram criadas diversas extensões para oferecer mais recursos no desenvolvimento de jogos, as quais dão suporte, dentre outros recursos, à realidade aumentada, à física do jogo e ao modo de multijogadores. Portanto o desenvolvedor passou a poder adicionar estas extensões conforme a sua necessidade. Dentre as extensões criadas, foi necessário utilizar no projeto *a AndEnginePhysicsBox2DExtension*, a extensão que permite a adição de física ao jogo.

Com esta extensão é possível adicionar gravidade aos cenários do jogo e criar corpos e associá-los às entidades presentes na cena.

Os corpos podem ser de três tipos distintos descritos abaixo:

- Corpo Estático: esse tipo não se move de modo algum. Seu movimento não pode ser simulado e se comporta como se tivesse uma massa infinita, ou seja, não reage a qualquer tipo de força aplicada a ele. É usado normalmente para definir o solo e, em alguns casos, paredes, dependendo do cenário.
- Corpo Cinemático: esse tipo de corpo pode ter sua velocidade simulada, mas não reage a nenhuma força aplicada a ele. Não interage com os corpos dinâmicos por exemplo. Um exemplo de uso seria uma plataforma móvel, que se movimenta constantemente em um cenário.
- Corpo Dinâmico: esse tipo tem sua movimentação completamente simulada e pode colidir com todos os tipos de corpo. Pode ser usados nos personagens do cenário e, em algumas situações, em projéteis.

As entidades poderão ter seu tipo de corpo alterado, mesmo após já estarem associadas. E após todos os corpos de um cenário estarem criados, é importante nomeá-los com um identificador, que terá grande utilidade ao verificar as colisões que ocorrerem entre eles.

2.6 Conceitos de Engenharia de *Software*

Neste item serão explicados alguns conceitos de engenharia de *software* referentes à modelagem de sistemas com o uso de diagramas de classe e o conceito de generalização, muito utilizado em linguagens orientadas a objetos.

2.6.1 Modelagem de Sistemas

Sommerville (2011, p.82) define a modelagem de sistemas como o processo de desenvolvimento de modelos abstratos de um sistema, em que cada modelo apresenta uma visão ou perspectiva, diferente do sistema. Essa modelagem geralmente representa o sistema com alguma notação gráfica, sendo que normalmente é baseada em notações de UML.

Ao desenvolver modelos de sistema, pode-se ser flexível quanto ao uso da notação gráfica. O detalhe e o rigor dependem de como se pretende usar o modelo. Os modelos gráficos podem ser comumente usados de três formas:

1. Como forma de facilitar a discussão sobre um sistema existente ou proposto;

2. Como forma de documentar um sistema já existente;

3. Como uma descrição detalhada de um sistema, que pode ser usada para gerar uma implementação do sistema.

Dentre os diversos tipos de modelagem de sistemas, os modelos estruturais de *software* exibem a organização de um sistema em termos de seus componentes e seus relacionamentos. Tem-se como exemplo disso os diagramas de classe, descritos no item 2.8.2.

2.6.2 Diagramas de Classe

Conforme Sommerville (2011) menciona, os diagramas de classe são usados no desenvolvimento de um modelo de sistema orientado a objetos para mostrar as classes de um sistema e as associações entre essas classes. Uma classe normalmente é definida da seguinte maneira:

1. O nome da classe de objeto está na seção superior;

2. Os atributos de classe estão na seção do meio. O que deve incluir os nomes dos atributos e, opcionalmente, seus tipos;
3. As operações (chamadas métodos, em Java e em outras linguagens de programação orientadas a objetos) associadas à classe de objeto estão na seção inferior do retângulo.

2.6.3 Generalização

A generalização é uma técnica normalmente usada no dia a dia. Ao invés de aprender as características detalhadas de cada entidade, colocam-se essas entidades em classes mais gerais e aprendem-se suas características. Dessa forma, é permitido inferir que os diferentes membros dessas classes têm algumas características em comum e se pode fazer afirmações genéricas que sejam aplicáveis a todos os membros da classe.

A UML tem um tipo específico de associação para denotar a generalização: representação de uma seta apontando para a classe mais geral.

As classes de nível mais baixo são subclasses que herdam os atributos e as operações de suas superclasses. Essas classes de nível mais baixo, em seguida, adicionam mais atributos e operações específicas.

3 PROPOSTA DO TRABALHO

Conforme os conceitos descritos no capítulo 2, neste trabalho é proposta a implementação de um jogo eletrônico para o sistema operacional Android. Para tanto, foram realizados estudos prévios sobre o desenvolvimento de aplicativos para este sistema e para o desenvolvimento de jogos.

Devido à escolha do Android e às facilidades já explicadas que o Eclipse oferece para o desenvolvimento de aplicações deste sistema através do ADT, foi decidido que o desenvolvimento seria realizado na linguagem de programação Java.

Também, através de pesquisas à procura de uma *engine* que fosse mais adequada ao desenvolvimento de jogos para Android, a *AndEngine* foi escolhida, devido às facilidades já mencionadas no item 2.3 desta monografia.

Ao decidir que tipo de jogo poderia ser produzido, optou-se que o jogo seria do tipo plataforma bidimensional e utilizaria conceitos de física como gravidade, velocidade linear, colisão e propriedades de corpos como massa, densidade, atrito e elasticidade. Para tanto foi utilizada uma extensão de *AndEngine*, a *AndEnginePhysicsBox2DExtension*. Tal extensão pode ser simplificadamente explicada como sendo a responsável por simular um ambiente onde são aplicadas as principais leis da física.

Com o foco dado principalmente no aprendizado, é proposta a criação de um jogo simples que utilize somente os conceitos fundamentais do desenvolvimento de um *game*, sendo que o projeto pode ser ainda continuado mesmo depois de avaliado, e com a possibilidade de se tornar um projeto ainda mais ambicioso.

O jogo terá classificação livre, porém terá como foco o público jovem. Para obter um melhor resultado final, foi definido que terá um enredo original com personagens característicos de uma determinada época, um design fiel ao enredo e que utilize bem a capacidade de processamento dos principais portáteis que utilizam o Android OS, uma jogabilidade fácil ao usuário e o uso de algum conceito de Inteligência Artificial voltada a jogos. Dessa forma, tem a finalidade de garantir um maior desafio aos jogadores e, consequentemente, um maior entretenimento.

4 MATERIAL E MÉTODOS

Neste capítulo serão descritos os materiais e os métodos utilizados para o desenvolvimento do *game*. No item 4.1 serão apresentados todos os materiais utilizados.

No item 4.2 é apresentada a metodologia, ou seja, a forma pela qual os procedimentos necessários ao desenvolvimento do projeto foram realizados, desde os estudos até a parte prática.

4.1 *Softwares* e *Hardwares* Utilizados

A seguir é apresentada a função de cada aplicativo e a descrição dos dispositivos móveis utilizados no projeto. Vale ressaltar que devido ao fato de a *AndEngine* não oferecer suporte a emuladores de dispositivos, muito comum no desenvolvimento de diversos outros aplicativos para Android, o uso de um *hardware* como material foi obrigatório (no caso um *smartphone*).

- *Samsung Galaxy SII Lite*: *smartphone* com sistema operacional Android 2.3, processador Dual Core de 1 Ghz. Tela de 4 polegadas com resolução de 800x480;
- *Samsung Galaxy SII*: *smartphone* com sistema operacional Android 2.3, processador Dual Core de 1.2 Ghz. Tela de 4.3 polegadas com resolução de 800x480;
- Eclipse: o Eclipse é a IDE mais adequada para o desenvolvimento de aplicativos para Android, pois oferece um ambiente que pode ser configurado através da instalação do Android SDK e do *plug-in* ADT, que serão explicados adiante;
- Android SDK: é o *kit* de desenvolvimento que deve ser instalado para permitir o desenvolvimento de aplicativos para o sistema Android. Vale observar que necessita ter o Java SDK previamente instalado;
- ADT: o ADT é um *plug-in* do Eclipse que facilita a criação de projetos Android e cria o *debug* das aplicações;
- *AndEngine*: foi a *game engine* escolhida. Uma *game engine* consiste em um conjunto de bibliotecas utilizado no desenvolvimento de jogos, cuja

função é simplificar e abstrair o desenvolvimento de jogos para a plataforma Android;

- *AndEnginePhysicsBox2DExtension*: extensão da *AndEngine* que trata todos os conceitos relacionados à física, como gravidade, velocidade, colisão de corpos e propriedades dos corpos como massa, elasticidade, atrito e densidade;
- *Pixillion Image Converter*: Devido ao fato de a *AndEngine* suportar apenas imagens do tipo PNG, ao pesquisar imagens relacionadas ao tema do jogo para adicionar ao projeto, necessitou-se deste conversor de imagens para converter outros tipos de imagens em PNG;
- *PhotoPad Image Editor*: este editor de imagens foi muito utilizado para adequar o tamanho das imagens, mudar as cores e deixar o plano de fundo transparente (utilizado principalmente nos *spritesheets*);
- *WavePad Sound Editor*: com a finalidade de deixar o som o mais leve possível e compatível com a *AndEngine*, foi utilizado este editor de som para converter arquivos de diferentes tipos. Assim, arquivos de som do tipo WAVE e do tipo MP3 puderam ser convertidos para OGG, o único tipo de arquivo aceito pela *AndEngine*.
- *PhysicsEditor*: editor que define a forma, a densidade, a elasticidade e o atrito dos objetos adicionados aos cenários do jogo;

Além da utilização dos programas descritos acima, na parte de estudos foram pesquisados em artigos e livros os conceitos relacionados à física e à engenharia de *software*.

4.2 Etapas do Projeto

Ao considerar que o *design* de um *game* é um dos itens fundamentais para garantir um bom produto final ao usuário de um jogo e o curso de Bacharelado em Ciência da Computação não oferece formação neste quesito, o *design* de algumas imagens foi, portanto, produzido com a ajuda de dois estudantes que cursam esta área de conhecimento. Neste tópico serão apresentadas as atividades realizadas e a divisão de tarefas executadas pelos estudantes do projeto.

4.2.1 Atividades Realizadas

Abaixo segue uma descrição resumida das atividades realizadas, a fim de revelar como o projeto foi produzido, sendo estas ordenadas abaixo conforme a evolução do projeto:

1. Pesquisa sobre o desenvolvimento de aplicativos para o sistema operacional Android;
2. Pesquisa sobre os atuais jogos para dispositivos móveis que obtiveram sucesso, a fim de promover ideias e obter modelos para o jogo desenvolvido;
3. Pesquisa sobre as principais ferramentas de desenvolvimento em *games*;
4. Seleção das ferramentas que foram utilizadas no projeto;
5. Treinamento com o objetivo de desenvolver aplicativos para Android, a fim de facilitar posteriormente o desenvolvimento do jogo;
6. Escolha do enredo, dos personagens, dos cenários, da trilha sonora e do tempo e do espaço no qual o jogo ocorrerá, com a finalidade de garantir uma identidade ao jogo;
7. Desenvolvimento de um protótipo do *game*, utilizando as ferramentas pesquisadas de desenvolvimento em jogos, no qual são priorizadas a mudança de cenários e a jogabilidade;
8. Desenvolvimento do projeto final, com a adição de inimigos, som e música;
9. Realização de testes;
10. Aplicação de arte gráfica desenvolvida por *designers* ao jogo, com a finalidade de torná-lo o mais amigável possível ao usuário e condizer com o enredo escolhido;
11. Realização de testes finais;
12. Conclusão e apresentação do projeto final.

4.2.2 Divisão de Tarefas

Ao considerar as atividades mencionadas no item anterior, as tarefas foram divididas pelos alunos participantes do projeto.

Em suma, foi definida a divisão do projeto em três partes distintas:

1) Estruturação do projeto e controle dos eventos: realizadas pelo aluno Francisco Mariani Guariba Filho;
2) Mecânica dos movimentos e testes: realizadas pelo aluno Ivo Rodrigues de Freitas Neto;
3) Arte gráfica e ilustração: realizadas pelos *designers* Henrique Fernandes Erdei e César Rosolino Pasqualinotto.

4.2.2.1 Estruturação do Projeto

A estruturação deste projeto consistiu em definir a melhor arquitetura do projeto, além do uso de alguns conceitos relacionados à geometria analítica, necessários ao seu desenvolvimento, como a disposição das imagens e dos objetos nos cenários e a implementação das telas.

O projeto de arquitetura está preocupado com a compreensão de como um sistema deve ser organizado e com a estrutura geral desse sistema. “O resultado do processo de projeto de arquitetura é um modelo de arquitetura que descreve como o sistema está organizado em um conjunto de componentes de comunicação” (SOMMERVILLE, 2011, p. 103). Para tal, foram utilizados alguns conceitos de engenharia de *software* como o diagrama de classes, com a função de mostrar as classes do sistema e a associação entre elas. A descrição se encontra nos itens 4.2.5 e 4.2.6, relativos às fases de implementação do aplicativo.

4.2.2.2 Controle de Eventos

O controle de eventos se concentra nos resultados em que cada evento do projeto produz.

Isso abrange a mudança de cenários, o tempo ideal pelo qual cada adversário do jogador aparece no primeiro cenário. Além das escolhas que jogador faz ao tocar na tela – tanto na escolha de opções do menu principal, como no uso do controle *touch screen* – ou ao tocar em algum dos dois botões do dispositivo utilizado.

4.2.3 Estudo e Definição da *Game Engine*

A fase de estudo foi a etapa que propiciou a base de conhecimento necessária para o desenvolvimento do jogo e subdividiu-se, nesta ordem, em pesquisar sobre o desenvolvimento de aplicativos para o sistema operacional Android, pesquisar sobre os atuais jogos que obtiveram sucesso para dispositivos móveis e pesquisar e selecionar as principais ferramentas de desenvolvimento de *games* para Android.

Ao pesquisar sobre os jogos lançados que obtiveram o maior número de *downloads*, pôde-se perceber que os melhores jogos baseavam-se em temas simplistas, mas que poderiam garantir um entretenimento contínuo. Não são tipos de jogos longos e com enredos complexos, como são os jogos de consoles caseiros mais potentes como *Xbox360* e *Playstation 3*, nos quais o usuário deve normalmente estar em casa e com tempo dedicado a este tipo de entretenimento. Devido a essa simplicidade, o usuário pode jogar em diversas situações do dia a dia como na rua, no ônibus ou numa sala de espera sem se cansar do jogo. Através dessa observação e do tempo de alguns meses para finalizar o projeto, optou-se por desenvolver um jogo simples que servisse para o aprendizado dos principais conceitos de jogos pelos alunos, mas ainda assim pudesse garantir um certo entretenimento.

Esta etapa não consistiu apenas na pesquisa teórica sobre o tema, como também, num estudo prático, através da implementação de pequenos aplicativos para Android, como exemplos para facilitar o entendimento sobre o tema. Primeiramente foram implementados aplicativos somente referentes ao sistema operacional, a fim de se familiarizar e compreender o funcionamento de uma aplicação. Após esse estudo, desenvolveu-se pequenos projetos que utilizam a *AndEngine*, para a compreensão do funcionamento dos métodos que

esta *engine* utiliza. Ao desenvolver esses pequenos projetos, o uso do *AndEngineExamples*, um outro projeto criado também por Nicolas Gramlich, teve grande valia na compreensão dos métodos e parâmetros que a *AndEngine* possui.

4.2.4 Tema e Enredo do Jogo

Através do que foi enunciado nos objetivos, a escolha do enredo, dos principais personagens e do tempo e espaço em que ocorrerá o desenrolar do jogo foi realizada, para conferir uma identidade ao jogo. Com isso, foi possível focar nas informações referentes ao enredo para produzir todo o *design*, o controle, os cenários e tudo o mais necessário à produção do jogo.

4.2.4.1 História

A redação de uma história tem a função de ajudar a definir o enredo e, consequentemente, o funcionamento, além de restringir os elementos que o jogo contém. Ao definir um tempo, um espaço, personagens e um enredo; é possível saber quais imagens, qual jogabilidade o jogo terá, qual o tipo de público o jogo deve agradar.

A seguir é reproduzida a história referente ao jogo, cujo tema foi escolhido através de pesquisas relacionadas à história do Japão para garantir uma verossimilhança.

Hikari no Ken

O período *Sengoku* foi uma das fases mais conturbadas e instáveis da história do Japão. Ocorreu entre a metade do século XV e o início do século XVII. Aquela região encontrava-se em uma grande instabilidade política. As disputas de poder e de terras entre os *daimiôs* (senhores feudais) ocasionaram uma onda sangrenta de guerras civis. Essas guerras contribuíram para desestruturar o poder central do Xogunato Muromachi, deixando o país à mercê da lei do mais forte e das constantes tentativas dos *daimiôs* de unificar o Japão sob seu controle, resultando sempre em fracassos.

Foi nesta época que nasceu Jinchuu. Ele fora encontrado abandonado em uma cesta na porta do Dojo Genki, onde fora acolhido pelo *sensei* Tairo, que passou a ensiná-lo o Kobudo, artes marciais samurais, e o Bushido, caminho do guerreiro. Naquela época ambos eram considerados o alicerce para um indivíduo tornar-se samurai. O Dojo Genki era o lugar onde se revelava a maior quantidade de samurais para defender o clã Musashi, um dos clãs que também estava em conflito e era seriamente ameaçado pelo clã Akuma, um dos mais temidos da região.

O medo dominava todas as classes dos feudos, desde camponeses inocentes até a corte. Os senhores feudais contavam tanto com os seus samurais, como também com mercenários e agentes secretos conhecidos como ninjas.

Em determinada noite alguns ninjas ordenados pelo clã inimigo Akuma invadiram o Dojo Genki, a fim de eliminar todos os possíveis samurais que habitavam ali. A discrição dos ninjas não foi suficiente para impedir o sensei Tairo de perceber a aproximação do bando. E, ele ordenou Jinchuu a se esconder e com a ajuda dos seus discípulos se preparou para o combate. Jinchuu, preocupado, correu para ficar atrás de uns arbustos e dali tinha uma pequena visão do que poderia acontecer.

Após uma intensa batalha Tairo, já muito ferido, sofreu um golpe fatal por Baka, o único ninja que havia sobrevivido e, que, após cumprir sua missão, incendiou o Dojo Genki, fugindo em seguida. Caso alguém aparecesse em poucos minutos, encontraria ali apenas o lar de Jinchuu destruído pelo fogo.

O pequeno Jinchuu, apavorado, correu em direção ao seu mestre que lhe disse as últimas palavras. Lembrou-lhe o quanto precisava ser forte, pois logo seria a última esperança para manter o clã Musashi vivo, para isso precisava seguir o Bushido rigorosamente. Jinchuu prometeu ser o melhor samurai de todos e preservar o clã Musashi a qualquer preço.

Além de ter a missão de proteger as pessoas do feudo do seu *daimo*, para Jinchuu, o *sensei* Tairo e seus discípulos eram considerados a sua família, algo de extrema importância a um samurai ou futuro samurai. E só a vitória sobre Akuma poderia resgatar a sua honra de volta.

4.2.5 Desenvolvimento do Protótipo

Após a fase de pesquisa, teve início o desenvolvimento do protótipo. O protótipo se baseou em fazer o personagem se movimentar em um cenário e realizar a mudança entre as principais telas do jogo.

Antes de implementar a mudança de cenários, deve-se saber que a maioria dos *smartphones* possuem sempre dois botões, descritos a seguir:

- Voltar: utilizado normalmente para retornar aos menus anteriores do sistema Android ou de aplicativos instalados e para, possivelmente, encerrar chamadas;
- *Menu*: utilizado normalmente para abrir algum menu de opções tanto do sistema, como de aplicativos instalados.

Como a preocupação inicial foi em relação às mudanças de cenários, foi criada uma tela de carregamento inicial que apresenta o jogo e dura um tempo determinado. Após isso a tela de menu principal é carregada, nela há uma MenuScene com as opções de jogar e sair. Ao tocar na opção de sair, o jogo é fechado, retornando ao sistema operacional. Ao tocar na opção de jogar é carregada a tela da primeira fase do *game* e nela foi dada a possibilidade de o jogador pausar o jogo ao tocar no botão *Menu* do dispositivo.

As telas de carregamento inicial e de *menu* principal, além dos os elementos usados nela estão representados a seguir, na figura 3 e figura 4, respectivamente:

Figura 3: Tela de Carregamento Inicial

Figura 4: Tela do *menu* principal.

Definiu-se, também, que caso o jogador esteja na tela do menu principal e pressione o botão voltar, fecha-se o jogo. Caso esteja em alguma fase, volta-se ao menu principal.

Na tela da fase inicial, o cenário foi definido por uma imagem de céu estática, uma montanha e um chão que se movem repetidamente, dando a impressão de constante movimentação através do uso de um recurso da *AndEngine*, denominado *AutoParallaxBackGround*. A imagem do controle *touchscreen* e o objeto da classe *AnimatedSprite* do personagem também foram adicionados e associados, assim foi dada a possibilidade ao jogador de controlar o personagem ao tocar no controle virtual. Depois, adicionou-se um ambiente físico ao cenário, possibilitando a definição da gravidade terrestre e a adição de dois corpos à fase: um corpo dinâmico associado à imagem do personagem e um estático associado ao solo do cenário.

A intenção foi fazer o personagem se movimentar conforme ocorria o toque na região em que o controle virtual se encontra e, assim, o personagem poderia ir para frente, para trás ou pular. Nesse último caso, o pulo do personagem foi tratado ao verificar se havia contato entre o corpo do personagem e o solo. Se houver contato, o corpo do jogador recebe uma determinada velocidade linear com sentido de baixo para cima e volta ao solo devido à gravidade terrestre simulada pela *AndEnginePhysicsBox2D*. Assim, o pulo foi simulado.

Os conceitos relacionados ao protótipo, portanto, serviram como base para prosseguir o desenvolvimento do jogo.

4.2.6 Desenvolvimento do *Game*

Ao desenvolver o *game* em si, o foco passou a ser a interação do jogador com os inimigos, as condições de vitória e derrota, a pontuação alcançada e os cuidados com mais detalhes, como mais imagens nos menus e a inclusão de sons.

Nos jogos é comum mostrar ao jogador o status de vida do personagem, ou seja, quantas vezes ainda o jogador pode sofrer danos antes que o jogo termine. Para isso, foi utilizada uma classe *HUD*, subclasse da *Scene* como explicado na fundamentação teórica. Nesta classe foram adicionados imagens de corações que representam a quantidade de vida do personagem, no caso cinco *imagens*.

Figura 5: *Sprite* de um coração para representar a vida do personagem.

Após a implementação da vida do personagem, focou-se em mostrar a pontuação alcançada (para cada acerto sobre o inimigo, a pontuação aumentaria), para isso decidiu-se colocar no cenário um texto que mostraria a pontuação. Segundo Mysliwiec (2012), é preciso inicialmente preparar a memória quando for inicializar textos nos quais ocorrerão atualizações constantemente. Para isso deve-se pensar em quais caracteres ele poderá receber, no caso do jogo, a palavra “Pontos” e os algarismos de 0 a 9. Se inicializasse apenas com “Pontos” e o texto fosse atualizado para “Pontos: 128”, por exemplo, o jogador poderia notar uma lentidão nesta mudança.

A seguir segue o código que representa tal conceito:

```
Text t = new Text(0, 0, font, "Pontos:0123456789", vbo);
```

Código 1: Criação de um objeto do tipo *Text*

O código 1 representa a criação de um objeto do tipo *Text*. Pode-se observar que esse objeto já inicia com todos os possíveis caracteres que pode receber para depois ocorrer atualizações.

Figura 6: Texto que representa a pontuação

Num primeiro momento, um inimigo foi adicionado ao cenário e a técnica de perseguição, uma técnica simples de Inteligência Artificial aplicada em jogos, foi dada a ele. Assim, ele passou a perseguir o jogador de qualquer lugar do cenário.

A seguir, no código 2, segue o algoritmo de perseguição:

```
private void perseguirJogador(Body inimigo, Body heroi) {
        if (inimigo.getPosition().x >= heroi.getPosition().x){
             Vector2 veloc = Vector2Pool.obtain(-1f, 0f);
             inimigo.setLinearVelocity(veloc);
             Vector2Pool.recycle(veloc);
        }
        if (inimigo.getPosition().x < heroi.getPosition().x){
             Vector2 veloc = Vector2Pool.obtain(1f, 0f);
             inimigo.setLinearVelocity(veloc);
             Vector2Pool.recycle(veloc);
        }
  }
```

Código 2: Algoritmo de perseguição

Em seguida, foi tratada a colisão entre o personagem e o inimigo. Quando ocorresse colisão, o personagem perderia uma vida e seria empurrado para o sentido oposto do inimigo se não estivesse atacando. Caso contrário o inimigo perderia uma vida, seria empurrado para o sentido oposto ao personagem e seria desassociado do cenário, caso seu atributo de vida fosse igual a zero. Também, se o personagem perdesse todas as suas vidas seria carregada e apresentada a tela de fim de jogo.

Após a verificação da colisão entre o personagem e um inimigo, novos inimigos passaram a ser adicionados em intervalos de tempo definidos. Então,

a colisão entre o personagem e eles foi verificada da mesma forma, assim como a técnica de perseguição ao personagem.

A seguir são mostradas as figuras 7 e 8, que representam o primeiro cenário. Lembrando que, durante a execução, novos inimigos aparecem em tempos definidos.

Figura 7: 1ª Fase do jogo.

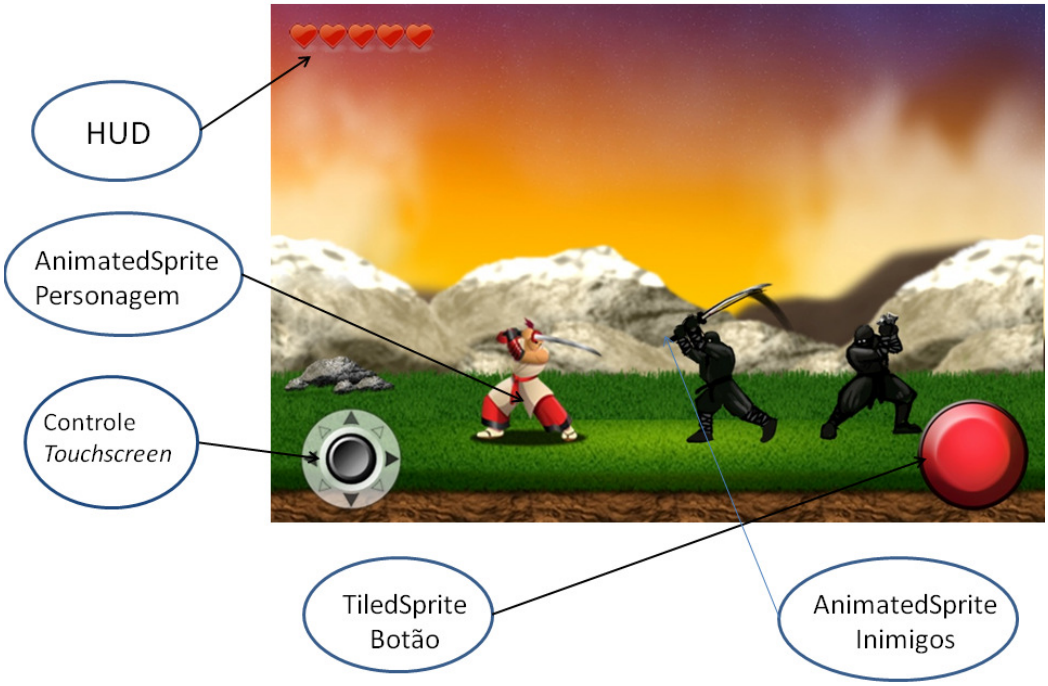

Figura 8: 1ª Fase com todos os elementos do cenário.

Caso o personagem sobreviva até um determinado tempo, ocorre a mudança para a segunda fase. Nessa fase são carregados o personagem e um inimigo final, que tem a mesma técnica de perseguição, porém uma quantidade maior de vidas e, a colisão resultaria em um maior dano ao personagem. A figura 8 ilustra o segundo cenário.

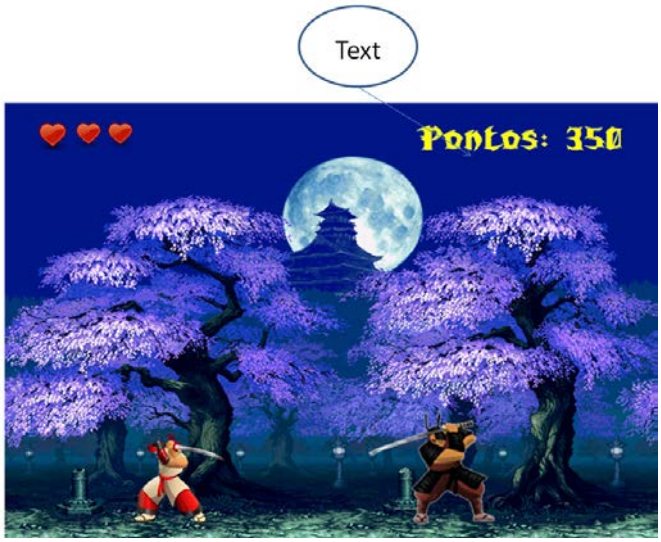

Figura 9: Segunda Fase.

Após a adição do 2º cenário, foi criada mais uma tela, que representa o fim de jogo. Ela passa a ser mostrada logo que a vida do personagem se iguala a zero em qualquer uma das fases que o jogo possui e a tela de cenário final, responsável por conter o desfecho do jogo, é carregada quando o jogador vence o inimigo do segundo cenário. A figura 10 representa esta tela.

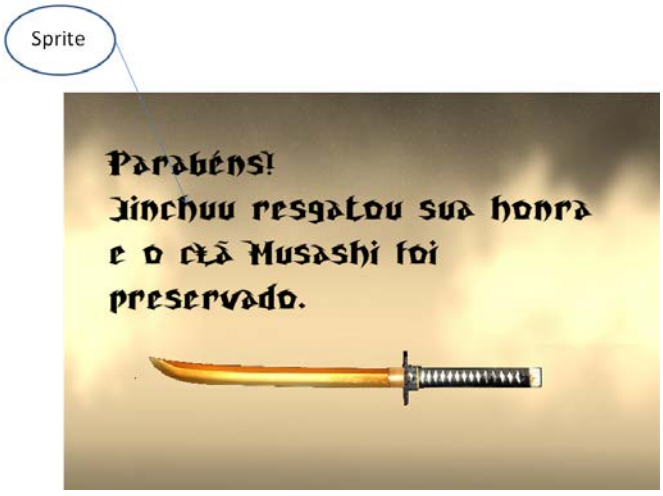

Figura 10: Tela do Cenário Final.

Finalmente, com todas as telas implementadas pode-se mostrar o esquema ilustrativo da figura 11, que contém as telas que o jogador poderá visualizar enquanto estiver executando o jogo.

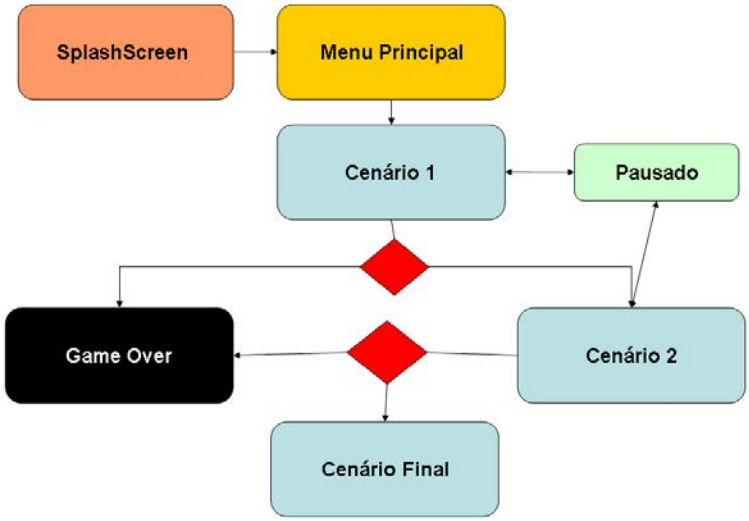

Figura 11: Fluxograma da mudança de telas.

Ao observar o fluxograma da figura 9, pode-se inferir que inicialmente é carregada a *splash screen*, ou seja, a tela inicial que é conhecida em jogos eletrônicos por mostrar o fabricante do produto ou o nome do jogo. Já implementada na etapa de desenvolvimento do protótipo, ela conduz à tela de *menu* principal. No toque da opção "jogar" dessa tela, a fase 1 é carregada. A partir de então, pode-se pausar o jogo quantas vezes quiser nessa tela e, também, na tela da 2ª fase. Há duas situações que conduzirão às telas diferentes:

- Variável de vida do jogador acabar: isso conduzirá à *game over scene*, ou tela de fim de jogo;
- O jogador sobreviver após um tempo determinado: isso conduzirá a segunda fase do jogo.

Na segunda fase, haverá novamente duas possibilidades:

- Variável de vida do jogador acabar: isso conduzirá à *game over scene*, ou tela de fim de jogo, assim como a primeira fase;

- O jogador vencer o inimigo: isso conduzirá a uma tela que apresentará o desfecho da história criada para o jogo, a tela que conterá o cenário final.

Quanto às classes utilizadas, o projeto final ficou estruturado com quatro classes: Personagem, Herói, Inimigo e *MainActivity.*

A classe *MainActivity* ficou responsável por realizar as mudanças de cenários a cada evento que ocorresse. Uma classe que realiza essa função diminui o tempo de carregamento dos elementos e facilita o acesso a objetos e diferentes partes do jogo em outras classes.

A figura 12 mostra um diagrama de classes que representa a hierarquia utilizada nas outras classes do projeto.

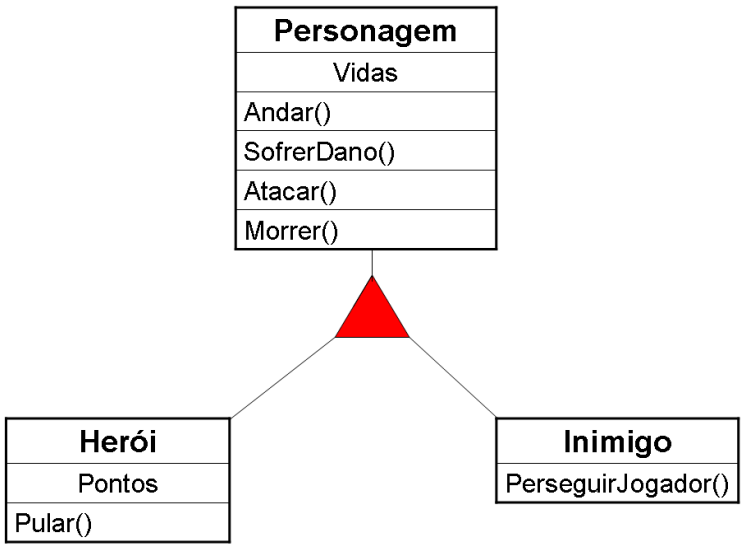

Figura 12: Hierarquia das classes dos personagens.

As classes Herói e Inimigo são subclasses da superclasse Personagem e possuem diversas animações feitas da mesma maneira, como andar, atacar, morrer e sofrer dano e um mesmo tipo de atributo que é a quantidade de vida. Elas diferem porque o jogador pode pontuar e fazer o personagem pular, enquanto os inimigos têm apenas um método de perseguir o jogador constantemente.

Após as telas estarem inseridas no projeto, foram adicionados temas musicais de fundo em cada tela e um som que representa uma espada ao ataque, sempre que o personagem a empregasse.

4.2.7 Testes Utilizados

Muitos aplicativos feitos para Android permitem que sua execução seja realizada por programas emuladores, portanto, sem a necessidade de um dispositivo portátil para analisar o programa funcionando. Entretanto, a *AndEngine* não oferece suporte a emuladores e isso implica que a observação do programa em tempo de execução seja feita obrigatoriamente em um dispositivo portátil. Dessa forma, o resultado de cada alteração realizada no projeto foi testado logo após a modificação ser concluída utilizando-se sempre um *smartphone* com a intenção de verificar se foram gerados erros no projeto, ou seja, para cada modificação feita no projeto, foi necessário reinstalar o aplicativo no Android.

Os testes de desenvolvimento do projeto se basearam nos testes de estrutura; que, segundo Sommerville (2003), consistem em uma abordagem de testes, em que estes são derivados do conhecimento da estrutura e da implementação do *software*. São aplicados às unidades de programa relativamente pequenas, como sub-rotinas, ou às operações associadas a um objeto.

Após a conclusão da etapa de desenvolvimento do *game* estar completa, foram realizados os testes para detecção de defeitos. Esses defeitos, segundo Sommerville (2003), têm a finalidade de expor defeitos latentes em um sistema de *software* antes de o sistema ser entregue. Ocorreram, portanto mais algumas modificações após a etapa de desenvolvimento.

Os problemas e os erros de desenvolvimento corrigidos pelos testes realizados serão mais bem explicados no próximo capítulo, que trata dos experimentos feitos.

5 EXPERIMENTOS

Pelo fato de haver pouca documentação no código da *AndEngine* e poucos artigos relacionados a ela, experimentos foram realizados constantemente durante a fase de estudo prático e desenvolvimento para desvendar o funcionamento da *engine* e dos métodos mais utilizados .

Os primeiros experimentos realizados foram em relação à colocação de imagens nos cenários. Como já explicado, as regiões de texturas são os recipientes das imagens, portanto devem ter tamanhos iguais ou maiores às imagens inseridas nos cenários, caso contrário o aplicativo trava assim que se tenta carregar a imagem. Foram realizados diversos experimentos nesse sentido, visando definir a menor textura possível para economizar memória.

Após as imagens estarem inseridas nos cenários, o toque no controle virtual precisou ser filtrado em direita, esquerda e cima (para o tratamento do pulo) através do seguinte código:

```
public void onControlChange(final BaseOnScreenControl pBaseOnScreenControl,
float pValueX, float pValueY)
{
	if (pValueX == -1) // esquerda
	{
		if (podePular)
			com.meuprototipo.entity.Personagem.andar(samurai);
		corpoSamurai.setLinearVelocity(-5, 0);
	}
	else if (pValueX == 1) //direita
	{
		if (podePular)
			com.meuprototipo.entity.Personagem.andar(samurai);
		corpoSamurai.setLinearVelocity(5, 0);
	}
	else if (pValueY == -1) //cima
	{
		if (podePular)
		{
			corpoSamurai.setLinearVelocity(0,-10); //pulo
			com.meuprototipo.entity.Personagem.pular(samurai);
		}
	}
}
```

Código 3: Controle do personagem.

O contato entre o jogador e o chão precisou ser também constantemente verificado, para impedir que o personagem pudesse pular enquanto já estivesse no ar.

Ao ocorrer as mudanças de cenários entre as fases 1, 2 e a tela principal, constatou-se que algumas variáveis não mantinham os valores desejados, sendo preciso, portanto, atualizá-las a cada mudança de cena. Também, foi observado que na segunda vez em diante em que a fase 1 foi acessada, a adição de novos inimigos ocorria de forma descontrolada.

O método do código 4 é responsável pela adição de inimigos a cada 10 segundos por exemplo:

```
mEngine.registerUpdateHandler(new TimerHandler( 10, true, new
ITimerCallback()
{
      public void onTimePassed(final TimerHandler pTimerHandler)
      {
            //código para adição de inimigos
      }
}));
```

Código 4: Adição de inimigos a cada 10 segundos.

Este método estava sendo executado constantemente, portanto a cada mudança de cenário, foi preciso fazer a *engine* limpá-lo através do seguinte comando:

```
mEngine.unregisterUpdateHandler(pTimerHandler);
```

Código 5: Atualização da *engine*

Quanto às animações dos personagens, conforme cada ação realizada como atacar, sofrer dano, pular e andar, foi preciso indicar quais quadros do *spritesheet* deveriam ser mostrados através do código descrito abaixo:

```
long[] duracao = {200, 200, 200, 200, 200, 200, 200};
int[] tilesAtacar = {10, 9, 7, 6, 5, 4, 3};
sprite.animate(duracao, tilesAtacar, false);
```

Código 6: Animação dos personagens

Foi necessário definir o tempo de duração de cada tile através de um vetor e quais tiles do *spritesheet* seriam mostrados por outro vetor. Então, o método *animate* recebe os vetores e um último parâmetro booleano indica se a animação deve estar em laço ou não para, enfim, simular a ação do personagem.

Descobriu-se que para não ocorrer travamentos no aplicativo, o tamanho dos dois vetores precisam, necessariamente, ser iguais e os quadros a serem mostrados devem estar em ordem crescente ou decrescente.

6 RESULTADOS

O projeto desenvolvido trabalhou com novas tecnologias na área da computação através do desenvolvimento de um jogo *mobile*. O resultado obtido consiste em um *game* para Android OS implementado através da *AndEngine*, uma *engine* nova lançada em junho de 2010 que ainda possui pouca documentação, mas um código bem produzido baseado na API *OpenGL*.

Foi, portanto, desenvolvido um simples jogo interativo com algumas características básicas, conforme descritas a seguir:

- *Menu* principal;
- Uso de algumas imagens produzidas por estudantes de *design*, principalmente referentes aos personagens;
- Personagem animado e controlável por meio de um controle virtual *touchscreen;*
- Uso de dois cenários com planos de profundidade e estruturas animadas;
- Uso de alguns conceitos de física;
- Interação do personagem com inimigos também animados;
- Uso de pontuação e vida do personagem;
- Possibilidade de o usuário pausar o jogo;
- Uso de música e som associados à temática escolhida para o jogo.

Com isso, foram aprendidos e utilizados os conceitos gerais referentes ao desenvolvimento de aplicativos para Android OS e jogos.

7 CONCLUSÃO

Com a implementação do projeto, pôde-se perceber que, para o desenvolvimento de um jogo eletrônico existe uma clara dependência entre os programadores e os *designers* gráficos para obter um bom resultado final. Isso se deve ao fato de que tanto a interação que o jogo oferecerá quanto os elementos gráficos influenciarão no entretenimento do usuário.

O produto final desenvolvido possui elementos básicos de um jogo eletrônico e procura seguir uma linha de jogos simples. Porém, que proporcione um excelente desafio aos usuários. Como é, por exemplo, a linha de *Angry Birds, Amazing Alex* e *Bad Piggies* produzidos pela já consagrada empresa Rovio, e *Fruit Ninja*, produzido pela THD.

Ainda não pode ser considerado um produto que esteja apto a entrar no mercado de *games*, concorrendo com outros títulos. Para tal, é preciso que se dê continuidade ao desenvolvimento com a adição de mais cenários, de mais interação entre o personagem e os cenários, de outros tipos de inimigos e de mais recursos de inteligência artificial voltada a jogos. Esse consistiria em um trabalho de mais tempo, já que mesmo os jogos mais simples para *móbile* são produzidos por empresas com diversos programadores e *designers.*

Apesar disso, pode-se concluir que o objetivo proposto de aprendizado sobre as técnicas e conceitos de desenvolvimento de jogos para Android OS e sobre funcionamento geral de aplicativos para este sistema operacional foi alcançado. Tal feito oferece a possibilidade aos alunos de se especializarem cada vez mais no assunto, dando continuidade a este trabalho ou amadurecendo ideias de novos *games*. Os conceitos aprendidos com este projeto podem ser aplicados continuamente em diversos tipos de jogos para Android.

7.1 Trabalhos Futuros

Devido ao aprendizado inicial sobre alguns recursos da *AndEngine* e alguns conceitos sobre a criação de jogos, há a possibilidade de desenvolver novos trabalhos.

A *AndEngine* contém diversos recursos próprios para o desenvolvimento de um jogo, porém, mesmo assim, possui nove extensões. Cada uma com características próprias que possibilitam a criação de aplicativos que utilizam alguns conceitos de realidade aumentada, outros que permitem a criação de planos de fundo animados, modos multijogadores, cenários que são divididos em quadros para economia de memória, entre outros recursos. Os nomes dessas extensões estão mencionados a seguir:

- *AndEngineAugmentedRealityExtension;*
- *AndEngineLiveWallPaperExtension;*
- *AndEngineMODPlayerExtension;*
- *AndEngineMultiPlayerExtension;*
- *AndEnginePhysicsBox2DExtension;*
- *AndEngineScriptingExtension;*
- *AndEngineSVGTextureRegionExtension;*
- *AndEngineTexturePackerExtension;*
- *AndEngineTMXTiledMapExtension.*

No caso do projeto foi utilizada apenas uma extensão, a *AndEnginePhysicsBox2D*. Há, portanto, um estímulo para utilizar mais extensões em novos projetos com outros enredos, temas e jogabilidade.

8 REFERÊNCIAS BIBLIOGRÁFICAS

ALÍPIO, Douglas: Máquina Virtual Dalvik. Mai. 2011. Disponível em: <http://www.euandroid.com.br/iniciante-android/2011/05/maquina-virtual-dalvik/>. Acesso em: 08 abr. 2012.

BURNETTE, Ed. Eclipse IDE: Pocket Guide – first edition. O´Reilly Media. 2005.

DE CASSIA, R. 200 milhões de pessoas jogam 'Angry Birds' por mês: Número ainda é inferior ao da rival Zynga, com 306 milhões por mês. VEJA On-line, São Paulo, 10 out. 2012. Disponível em: <http://veja.abril.com.br/noticia/vida-digital/200-milhoes-de-pessoas-jogam-angry-birds-por-mes>. Acesso em: 16/10/2012.

GRAMLICH, N. AndEngine Free Android 2D OpenGL Game Engine. Schriesheim, Alemanha: 2010. Disponível em: <http://www.andengine.org/blog>. Blog pertencente ao criador da *AndEngine*, que possui fóruns de discussões, postagens de trabalhos e notícias sobre a *AndEngine*. Acesso em 15/10/2012.

LIU, P. Avanços recentes do telefone celular. EZ9. São Paulo, abr. 2012. Disponível em: <http://pt.ez9articles.appspot.com/article/recent-cell-phone-advancements>. Acesso em: 05 jun. 2012.

MYSLIWIEC, M. Android Game Development Tutorial. Inglaterra: Matim Development 2011. Apresenta um tutorial que explica diversos conceitos relacionados à *AndEngine*. Disponível em: <https://sites.google.com/site/matimdevelopment/tutorials>. Acesso em: 01/07/2012

PEREIRA, Lúcio; DA SILVA Michel. Android Para Desenvolvedores. Rio de Janeiro: Brasport, 2009.

SCHEMBERGER, Elder; FREITAS, Ivonei; VANI, Ramiro. Plataforma Android.UNIOESTE- Universidade Estadual do Oeste do Paraná – Cascavel-PR. UNIVEL- União Educacional de Cascavel – Cascavel- PR. 2009.

SCOCCO Daniel; CAVALCANTE Krysamon; ZANON Bruno. Blog do Android. São Paulo, 2009. Apresenta as últimas notícias sobre Android O.S. e seus aplicativos. Disponível em: <http://www.blogdoandroid.com/2012/03/os-10-melhores-jogos-para-android-gratis/>. Acesso em: 16/10/2012

SOMMERVILLE, I. Engenharia de Software - 6ª Edição. Tradução de André Maurício de Andrade. São Paulo: Addison Weasley, 2003.

SOMMERVILLE, I. Engenharia de Software - 9ª Edição. Tradução de Ivan Bosnic e Kalinka G. de O. Gonçalves; revisão técnica de Kechi Hirama. São Paulo: Pearson Prentice Hall, 2011.