UNIVERSIDADE TECNOLOGICA FEDERAL DO PARANÁ

ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

MARCELO KORJENIOSKI

DESENVOLVIMENTO DE JOGOS 2D COM ANDROID

Curitiba

2011

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

MARCELO KORJENIOSKI

DESENVOLVIMENTO DE JOGOS 2D COM ANDROID

Monografia apresentada como requisito parcial para conclusão do Curso de Especialização em Tecnologia Java - UTFPR.

Orientador: Prof. João Alberto Fabro

Curitiba

2011

Aprendemos através dos nossos erros.

Sumário

Sumário

Resumo

A procura de jogos e aplicativos para a plataforma portáteis usando iOS/Android esta em alta (FARAGO 2011) e a venda crescente de dispositivos com Android em 2011 favorecem o desenvolvimento para a plataforma Android (CANALYS, 2011). Como a exigência para desenvolvimento de jogos para o mercado ainda é pouco complexo foi criada a oportunidade para pequenos empreendedores pelo menos do ponto de vista do desenvolvimento do produto, e a publicação via Android Market é um bom caminho para a distribuição.

Com este cenário o objetivo deste trabalho é mostrar os primeiros passos para criar um jogo para Android com foco na distribuição internacional e comentando as técnicas básicas para criação de um jogo 2D. Foi criado um jogo chamado “Bats on Fire” que demonstra na prática, o uso de cada uma das tecnologias e os seus resultados obtidos.

1 INTRODUÇÃO

1.1 Mercado de Jogos

O aumento da procura de games para dispositivos portáteis usando iOS/Android liberada pela Flurry (FARAGO 2011), mostra que, do período de 2009 para 2010, este segmento teve um crescimento de 5% para 8%, avançando na fatia dos jogos para consoles e portáteis de empresas já consagradas, como por exemplo Nintendo, Sony e Microsoft.

Com o intuito de escolher a melhor plataforma para a criação de jogos para celulares utilizando Java, foi realizado um estudo de viabilidade. A plataforma a ser selecionada deve possuir como características: consumir pouco tempo no desenvolvimento, baixo investimento financeiro nos recursos para programação e que também garanta uma distribuição abrangente com pouca burocracia para venda e distribuição.

A proposta inicial era criar jogos para JME (Java Micro Edition). Entretanto a distribuição e venda de aplicativos consome muito tempo, já que não existe uma Store (loja) unificada e sim varias lojas separadas. A Google oferece a Android Market como ponto central de distribuição dos softwares facilitando o gerenciamento das vendas e suporte do produto. Desta forma, foi selecionada a plataforma Android para o desenvolvimento deste trabalho.

1.2 Objetivos

Este projeto objetiva mostrar como desenvolver um jogo bidimensional para a plataforma Android de Smartphones. No início da pesquisa sobre a biblioteca gráfica OpenGL ES para Android, foi encontrada uma *game engine* (conjunto de bibliotecas para desenvolvimento de jogos) 2D (duas dimensões) chamada AndEngine licenciada como LGPL (*Lesser General Public License*) que usa como base o OpenGL ES 1.0.

Este conjunto de bibliotecas tem vários recursos disponíveis que vão desde o tratamento da simulação física do jogo até suporte a jogos com múltiplos jogadores em rede (*Network Multiplayer*). Desta maneira o esforço para criar um framework para o desenvolvimento do jogo foi poupado; a engine proporcionou quase tudo o que era necessário para concluir o projeto (ANDENGINE, 2011).

O protótipo desenvolvido é baseado em jogos de movimento lateral (*Side Scrolling*), muito comuns em consoles mais antigos pertencentes à terceira geração de consoles mais conhecida como era 8-bit (HISTORY, 2011). O jogo desenvolvido chama-se "*Bats on Fire*" e possui temática medieval com vampiros, zumbis e alguns monstros da mitologia grega. É ambientado na Transilvânia do século XV onde a crueldade de Vlad Tepes aterrorizava a população (VAMPIRETOOLS, 2011). Para uma melhor ambientação, foi realizado um estudo no tocante a nomes Húngaros da época, que servissem aos personagens. O jogo está dividido entre seis telas de atividades. Um melhor detalhamento destas técnicas pode ser visto no capítulo 3.

1.3 Organização do Trabalho

No capítulo 2 apresenta-se um referencial teórico sobre os assuntos abordados nesta monografia. O capítulo 3 contém a descrição da metodologia aplicada no projeto e no capítulo 4 estão os resultados obtidos. Por fim, o capítulo 5 trata das considerações finais sugerindo novas implementações.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentadas as principais tecnologias utilizadas no desenvolvimento deste projeto, bem como uma breve apresentação do funcionamento do jogo.

2.1 A Tecnologia Android

O Android é um sistema operacional de código aberto (*open-source*) para dispositivos móveis que foi inicialmente desenvolvido pela Android Inc. em 2003. Esta empresa foi fundada por Andy Rubin e Rich Miner em Palo Alto, Califórnia, USA em Outubro de 2003.

O sistema operacional Android para dispositivos móveis utiliza Java, que é uma linguagem de programação orientada a objetos, independente de plataforma, desenvolvida pela Sun MicrosystemsInc. A linguagem Java é tanto compilada como interpretada: o compilador transforma o programa em bytecodes, que consistem em um tipo de código de máquina específico da linguagem Java; o interpretador, disponível na JVM (*Java Virtual Machine*) que pode ser instalada em qualquer plataforma, transforma os bytecodes em linguagem de máquina para execução, sem que seja necessário compilar o programa novamente. As aplicações de Android não tem um único ponto de entrada como o método main(), elas tem quatro tipos de componentes essenciais que o sistema pode instanciar. São eles: *Activity*, *Service*, *Broadcast receivers* e *Content providers*.

Activity (Atividade): Responsável pela criação da tela do aplicativo Android. Um aplicativo pode ter várias *Activities* para compor o sistema.

Service (Serviço): não tem interface visual, sendo responsável por executar tarefas em paralelo a uma *Activity* como, por exemplo, acesso a rede.

Broadcast receivers **(Tratadores de mensagens)**: estes componentes funcionam como gatilhos, e ficam aguardando algum evento ocorrer com, por exemplo, a chegada de uma mensagem de texto. Ao ser notificado da ocorrência do evento, o *broadcast receiver* é ativado para tratá-lo, como por exemplo, através da execução de uma Atividade, para iniciar uma aplicação que trate a mensagem.

Content providers **(Provedores de conteúdo)**: este componente é responsável por compartilhar dados entre as aplicações.

O código Java compilado, juntamente com todos os dados e arquivos necessários pela aplicação, deve ser empacotado em um arquivo com a extensão ".apk" (Android Package), através da ferramenta "Aapt Tool". Este arquivo é usado para a distribuição da aplicação (ANDROID,2011).

O Android usa sua própria maquina virtual chamada Dalvik que é baseada em registradores e não em pilhas como a Java Virtual Machine o que melhora seu desempenho (SHI ,2011). A Dalvik executa arquivos ".dex" (Dalvik Executable) que são compilados a partir de arquivos Java previamente compilados e projetados para otimizar o uso de memoria e compartilhamento de dados (EHRINGER ,2011).

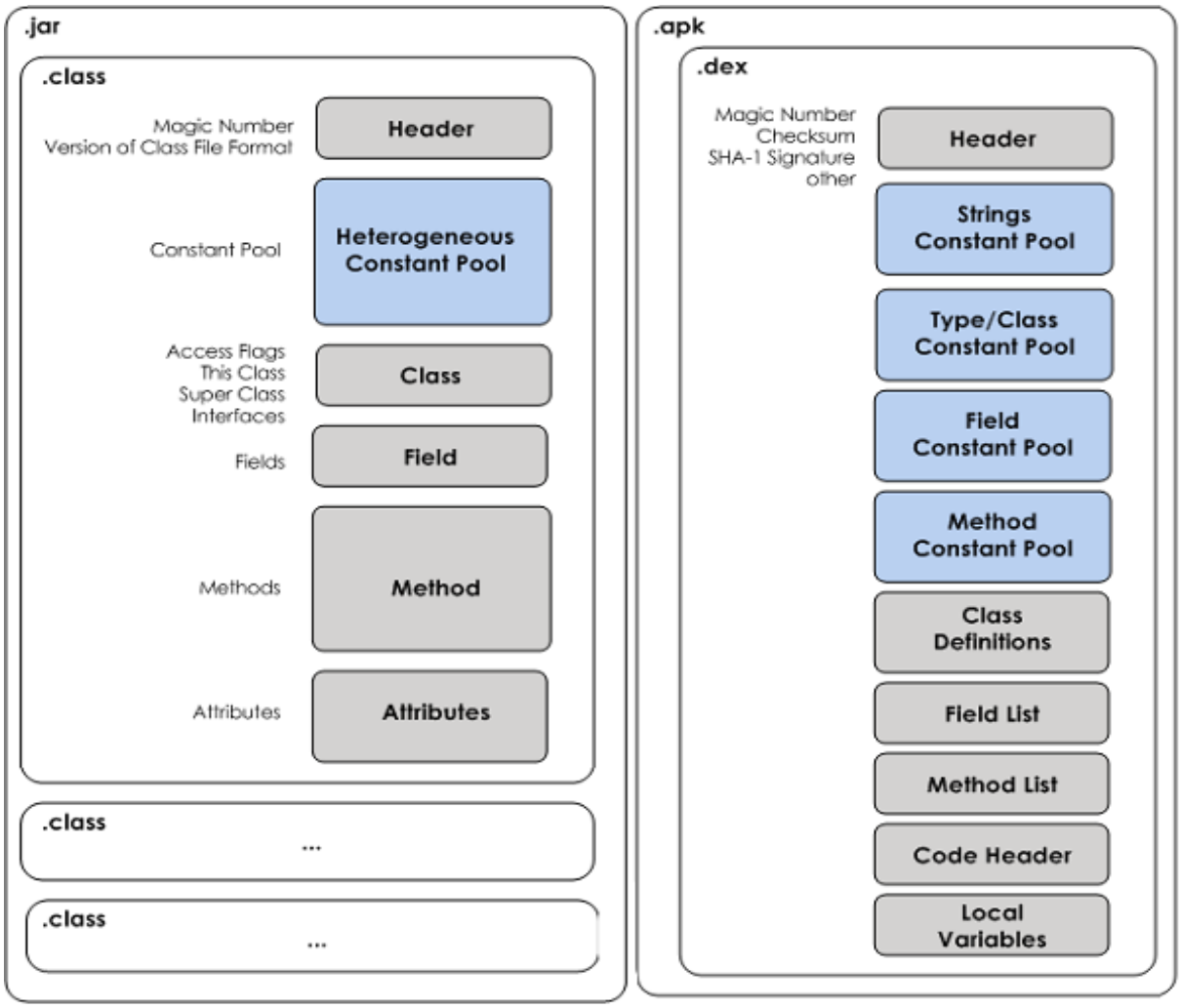

Figura 2.1 – Comparação do formato .class da JVM com o .dex usado pela Dalvik VM.

2.2 Linha do Tempo do Sistema Android

O sistema operacional para dispositivos móveis Android teve o início de seu desenvolvimento na empresa Android Inc. Em agosto de 2005 a Google Corp. comprou a Android Inc, transformando-a em sua subsidiária. Os principais profissionais da empresa, como Andy Rubin, Rich Miner e Chris White, foram mantidos após a compra. Dentro da Google, Rubin deu continuidade ao desenvolvimento do sistema, que é baseado no núcleo (*kernel*) do sistema operacional Linux.

No dia 5 de Novembro de 2007 é criada a *Open Handset Alliance*(OHA), um consórcio de várias empresas. A meta da OHA foi criar padrões abertos para dispositivos móveis. No mesmo dia, ela apresentou seu primeiro produto: o Android, um sistema operacional para dispositivos móveis construído sobre o Kernel do Linux versão 2.6.

O SDK(*Software Development Kit*, Kit de desenvolvimento) do Android foi apresentado pela primeira vez aos desenvolvedores no dia 12 de Novembro de 2007. A Google criou um campeonato de desenvolvimento de aplicativos para Android que começou dia 02 de Janeiro de 2008 e durou até 17 de Abril de 2008 e que oferecia um prêmio total de 5 milhões de dólares para os melhores e mais inovadores aplicativos.

O Android Market foi anunciado no dia 28 de Agosto de 2008, facilitando a distribuição dos aplicativos (apps), games e utilitários para seus dispositivos licenciados. A SDK 1.0 é lançada para os desenvolvedores no dia 23 de Setembro de 2008, o que permitiu o início do desenvolvimento.

A Google liberou todo o código fonte sobre a licença “Apache” (APACHE, 2011). Mas, mesmo o código sendo aberto para usar a marca Android da Google, é necessário que a empresa certifique o dispositivo de acordo com o documento de definição de compatibilidade. Depois do equipamento ser certificado ele terá acesso às aplicações fechadas, que inclui o Android Market (ANDROID, 2011).

2.3 Android Market

O Android Market permite que desenvolvedores distribuam seus aplicativos de forma direta para os usuários de smartphones Android. O Android Market é aberto para

todos os desenvolvedores de aplicativos Android. É necessário fazer um cadastro, pagar uma taxa de US$ 25.00, aceitar os termos de distribuição e preparar seu aplicativo conforme indicado na documentação de publicação (MARKET, 2011).

Feito o cadastro o desenvolvedor têm o controle completo sobre quando e como tornar suas aplicações disponíveis para os usuários, pode também gerenciar facilmente o seu portfólio de aplicativos, ver informações sobre downloads, classificações e comentários. Quando a aplicação é publicada é gerada uma assinatura digital que será usada para identificação do seu aplicativo que permitirá publicar atualizações e novas versões.

2.4 Versões disponíveis

Atualmente existem sete versões do Android disponíveis, e saber qual é a versão mais utilizada é a chave do sucesso para conseguir alcançar o maior número de usuários para sua aplicação. No site oficial do desenvolvedor do Android (http://developer.android.com/) existe um gráfico que mostra as versões disponíveis e qual a porcentagem de usuários. Estes dados são coletados dos aparelhos quando acessam o Android Market. Como apenas aparelhos licenciados pela Google tem acesso ao Android Market, celulares genéricos estão fora desta pesquisa.

2.5 Tecnologia OpenGL

A biblioteca OpenGL (*Open Graphics Library*) é uma especificação de padrão para desenvolver aplicações gráficas em 2D e 3D criado em 1992 por um conselho formado pelas empresas 3DLabs, ATI, Dell, Evans&Sutherland, HP, IBM, Intel, Matrox, NVIDIA, Sun e Silicon Graphics. As instruções para execução do OpenGL pode ser via hardware ou software que emula estas funcionalidades, desta maneira é possível atualizar API sem dificuldades. O conjunto de funções contidas na API pode praticamente acessar todos os recursos do hardware de vídeo. Usando as funções da API é possível alterar cor, transparência, cálculos de iluminação, efeitos de neblina dos objetos usados. Uma das características da OpenGL é que ela trabalha a medida de ângulos em graus e não em radianos, os componentes de cor por padrão usam pontos flutuantes que varia de 0 até 1, as linhas de coordenadas gráficas seguem o eixo cartesiano, assim o número da linha é decrescente (KHRONOS, 2011).

2.5.1 OpenGL ES

Esta API é a mais enxuta da versão original do OpenGL, livre de *royalties* e tem funcionalidade para gráficos 2D e 3D. Ele está disponível nas principais plataformas como iOS, Android, Symbian e Windows Mobile. Ela foi projetada para sistemas embarcados como celulares, PDAs, vídeo games, tablets e etc. Esta versão é mantida pelo Khronos Group (KHRONOS, 2011).

O Android possuiu suporte 3D de alto desempenho através da API OpenGL ES (*OpenGL for Embedded Systems*) a partir da versão 2.0.

Existem várias versões da API OpenGL ES. A versão 1.0 do OpenGL ES foi criada com base na versão 1.3 do OpenGL. A versão 1.1 é baseada na versão 1.5 e a versão 2.0 está usando a versão 2.0 da biblioteca OpenGL. Existe um software para testar a capacidade de processamento do seu equipamento e que serve para mostrar todos os recursos disponíveis em cada versão do OpenGL ES. Ele é chamado GLBenchmark (GLBENCHMARK, 2011) e mostra de forma prática os melhores equipamentos do mercado usando a pontuação obtida nos testes.

- OpenGL ES 1.0 é a biblioteca de gráficos 3D padrão do sistema operacional Symbian OS e Android.
- OpenGL ES 1.1 é utilizada como a biblioteca de gráficos 3D do iPhone e Android.

- OpenGL ES 2.0 será a biblioteca de gráficos 3D do console Pandora, iPhone e Android. Esta versão também foi escolhida para o WebGL (OpenGL para navegadores internet - browsers).

2.6 Tecnologias Gráficas

A computação gráfica é destinada para a geração de imagens em geral e que foi criada para suprir a necessidade humana para visualizar dados. Os jogos são os que mais usam a computação gráfica e por este motivo são os que mais contribuíram para seu desenvolvimento.

2.6.1 Gráficos em 2D

Na computação gráfica 2D, os objetos gráficos são visualizados em duas dimensões (largura e comprimento) (2D, 2011).

2.6.2 Sprite

Esta técnica começou a ser usada nos arcades por volta de 1974 quando surgiu hardware necessário para efetuar a animação. Antes de ser usado o Sprite existia apenas uma camada para criar os gráficos do jogo. Como uso do Sprite é criado uma camada acima do fundo onde as imagens são posicionadas. A empresa japonesa Taito usou no jogo “Basketball” uma imagem Sprite que representava os 4 jogadores do jogo sobre o cenário de fundo. A figura 2.1 mostra a técnica em uso no jogo “Basketball” (SPRITE, 2011).

Figura 2.1 – Sprites dos jogadores do jogo Basketball criado pela Taito em 1974.

2.6.3 Parallax Scrolling

A técnica de computação gráfica Parallax Scrolling (PARALLAX, 2011) é uma técnica de deslizamento que usa camadas que se movem em velocidades diferentes para formar o cenário do jogo. Com esta técnica o cenário da à impressão de profundidade criando um pseudo-3D por ter mais de um ponto de referência. Ela ficou muito popular nos anos de 1982 com o jogo para Arcade chamado Moon Patrol.

Há dois jogos muito conhecidos que usam esta técnica. São eles: Super Mario World para SNES e Sonic - The Hedgehog para Mega Drive. Esta técnica tem como princípio colocar as camadas em diferentes velocidades. A figura 2.2 mostra as camadas de um cenário.

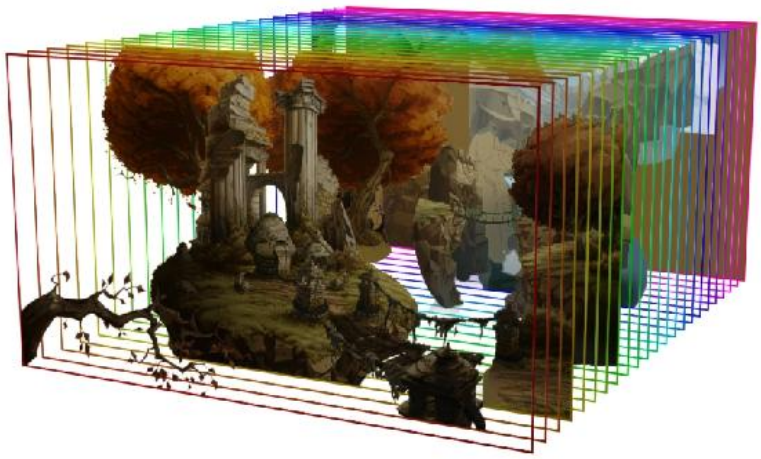

Figura 2.2 - Camadas de um parallax scrolling no jogo The Whispered World.

2.7 Game Engine

A Game Engine é um software com IDE (***Integrated Development Environment)*** e scripts e/ou um conjunto de bibliotecas que reúne varias funcionalidades que simplificam e facilitam a criação de jogos e poupam o programador de ter que "reinventar a roda". Na sua grande maioria as Engines são distribuídas como API (***Application Programming Interface)*** . As vantagens de se usar uma engine é que

já estão disponíveis funções de detecção de colisão, animação, sons, inteligência artificial, simulador de física, networking, gerenciamento de memoria e etc. Antes da game engine os jogos eram criados do zero com e o aproveitamento dos códigos era mínimo. O termo game engine surgiu por volta dos anos 90 e tem relação com os jogos de tiro em primeira pessoa. Seu uso ficou popular com os jogos Doom e Quake que disponibilizavam parte do código do jogo para que programadores criassem seus próprios gráficos, fases, personagens e armas. Hoje existem varias empresas que são especialistas em criar game engine. Alguns exemplos de game engines são **Unity, Corona, Trinigy e TorqueX**

2.8 Padrões de Projeto

"*Cada padrão descreve um problema no nosso ambiente e o cerne de sua solução, de tal forma que você possa usar essa solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira"* Christopher Alexander.

O movimento ao redor de padrões de projeto ganhou força com o livro "Design Patterns: Elements of Reusable Object-Oriented Software", publicado em 1995 por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, conhecidos como a "Gangue dos Quatro" (Gang of Four) ou simplesmente "GoF". Eles não criaram as 23 soluções apresentadas no livro, mas organizaram o trabalho feito pela comunidade (GAMMA, 2000).

Os padrões de projeto apresentados pelo GOF são divididos em três categorias: Estrutural, Comportamental e de Criação. Cada categoria trata de um tipo de solução específica.

- Padrões de Criação – Abstraem o processo de instanciação de objetos. Eles ajudam a tornar um sistema independente de como seus objetos são criados, compostos e representados e também, dão muita flexibilidade para o que é criado, quem cria, como e quando cria.
- Padrões de Estrutura – Preocupam-se com a forma como as classes e objetos são compostos para formar estruturas maiores. Neste padrão as classes utilizam a herança para compor interfaces ou implementações.
- Padrões de Comportamento – Preocupam-se com algoritmos e a

atribuição de responsabilidades entre os objetos, dando um grande foco na comunicação entre eles.

Os padrões utilizados neste projeto são *Observer*, *Mediator* e *Factory Method* (*Object pool*)

Observer - O processo consiste em um objeto (observado) disparar eventos de notificação para os objetos observadores poderem agir de acordo com essa mudança. Um exemplo nos jogos é a colisão entre elementos e também a reação de um inimigo a partir de um movimento do jogador.

Mediator - Este padrão é usado para evitar que os objetos se refiram uns aos outros explicitamente e permite variar suas interações, então ele fornece um acoplamento fraco entre os objetos. Como exemplo, pode ser citado os jogos de RPG: Ao invés de cada classe de personagem/inimigo ter que criar um método para calcular ataque, defesa e vida, é possível que a classe personagem solicite que uma classe intermediária Mediator faça o cálculo do ataque, sem precisar se comunicar diretamente com o objeto do inimigo.

Factory Method (Object pool) - Os objetos são inicializados e guardados em um pool prontos para o uso. Quando um cliente solicita para o pool um objeto após ele ser usado volta a ser guardado. Esta técnica aumenta muito o desempenho já que a criação de objetos tem um grande custo e em jogos é muito usada para diminuir lentidões durante a execução do programa. Um exemplo é a chamada de um inimigo na tela: Se a cada vez que você adicioná-lo no jogo tiver que criar o objeto que irá carregar textura, alocar memória e, quando for morto, tiver que destruir o objeto (e fazer isso novamente para cada chamada) o custo será enorme e o consumo de memória irá variar bastante, deixando o jogo instável. Então, utilizando um pool de objetos quando o objeto "inimigo" for chamado para o jogo, ele estará pronto para uso e, quando este morrer, bastará reiniciar os seus atributos.

2.9 Conclusão

Este projeto visa implantar as tecnologias, padrões e serviços citados para a criação de um jogo 2D para Android para mostrar na prática o seu uso.

3 METODOLOGIA E DESENVOLVIMENTO

O projeto mostra de forma simplificada o desenvolvimento de jogos 2D para Android 2.1, discorrendo sobre os passos iniciais para a criação. Além disso, mostra os padrões de projeto e técnicas de computação gráfica.

Para o desenvolvimento foi usado a IDE (*Integrated Development Environment* – Ambiente de Desenvolvimento Integrado) Eclipse (ECLIPSE, 2011) juntamente com Android SDK (*Software Development Kit*) (DEVELOPER. 2011) e a biblioteca AndEngine (ANDENGINE, 2011).

Trabalhar com jogos 2D é a melhor opção para quem tem poucos recursos gráficos para desenvolvimento, já que hoje em dia é mais fácil (e barato) encontrar artistas que possam criar modelos em 2D, em relação aos modelados em 3D. As imagens do jogo foram feitas usando o editor de imagens "GIMP" (GIMP, 2011) no formato PNG (*Portable Network Graphics*) (PNG, 2011).

Para a animação dos elementos na tela (inimigos, fireballs e personagens), foram usados Sprites, que são imagens bidimensionais que são adicionadas na tela sem deixar traços dos seus movimentos.

A parte sonora, como efeitos especiais, foi criada usando o editor de áudio "Audacity" (AUDACITY, 2011). O uso de MIDI (*Musical Instrument Digital Interface*) (MIDI, 2011) para a trilha sonora é uma forma de economizar recursos de armazenamento uma vez que este formato ocupa pouco espaço e é possível criar músicas que lembrem facilmente a época dos consoles de 16Bit. Para os efeitos sonoros será usado o formato WAV (*Waveform Audio File*) (WAVE, 2011) para conseguir uma qualidade maior na execução do som.

3.1 Requisitos Funcionais e Não-Funcionais

O usuário tem acesso ao download do software por meio da Android Market criando uma conta. O software é gratuito e pode ser encontrado pelo nome "Bats on Fire".

3.1.1Requisitos Não-Funcionais

O requisito básico para que o jogo seja executado é utilizar um dispositivo com sistema operacional Android superior a versão 2.1 que possua os sensores de Acelerômetro e Touch Screen e com no mínimo 5 Megabytes de espaço para a instalação. O sistema irá permitir que o usuário navegue em todas as telas usando o *Touch screen* do dispositivo.

3.1.2 Requisitos Funcionais

- O jogo deve ter suporte à internacionalização e a escolha do idioma irá ficar disponível na tela de Opções(Options) antes de jogo começar.
- O jogo deve permitir que o jogador escolha os níveis de dificuldade “fácil”, “normal” e “difícil” através da tela de Opções antes de jogo começar.
- O jogo disponibiliza a escolha “ativar” ou “desativar” o som através da tela de Opções antes de começar.
- É permitido que o jogador calibre a posição inicial da tela do jogo através da tela de Opções, antes do início.
- O jogo possui uma tela de Créditos (Credits) que mostra o nome de todos os responsáveis pelo desenvolvimento.
- O jogo é composto de três cenários e usa a técnica parallax para criar o cenário de fundo usando o recurso oferecido pela biblioteca Andengine.
- Todas as imagens usadas no jogo estão no formato PNG.
- Todas as músicas do jogo estão no formato MIDI.
- Todos os efeitos sonoros estão no formato WAV.
- São utilizados Sprites para a animação de objetos como personagem principal, inimigos, botões e etc.
- O jogo possui três fases que são divididas em duas partes: hordas com inimigos e chefe (boss). Para terminar cada fase o jogador terá que sobreviver às hordas de inimigos e no final derrotar o chefe, assim ele irá avançar para a próxima fase.

- O jogo tem uma música para cada cenário. Os efeitos sonoros para disparos e colisões utilizam os recursos de mídia oferecidos pela biblioteca Andengine.
- O jogo tem um sistema de estatísticas do jogador que guarda informações sobre suas ações durante a partida atual, como quantas vidas perdeu, quantos tiros errou e etc.
- O jogo permite que o jogador faça uma pausa, e que o jogo retorne ao mesmo estado quando voltar a jogar. Durante a pausa são mostradas as estatísticas do jogo.
- O jogador movimenta seu personagem pelo cenário usando o acelerômetro do aparelho.
- O jogo usa o sistema de detecção de colisão oferecido pela biblioteca Andengine.
- O jogador interage com os inimigos durante o jogo através de dois ataques: a “Fireball” e o “Psycho Power”.
- O comportamento da “Fireball” no jogo é o de um projétil lançado pelo personagem principal que percorre um caminho retilíneo a partir da coordenada da parte frontal do personagem. O dano ocorre quando existe a colisão entre a “Fireball” e o inimigo.
- O “Psycho Power” dá dano ao inimigo quando o jogador usa o touch screen na posição onde o inimigo se encontra.
- Quando o jogador perde ou finaliza o jogo é mostrada a tela de Game Over com as estatísticas da partida.

3.2 Diagrama de Casos de Uso

A figura 3.1 apresenta o diagrama de caso de uso do Jogo. Apresenta as ações que o jogador pode executar neste aplicativo.

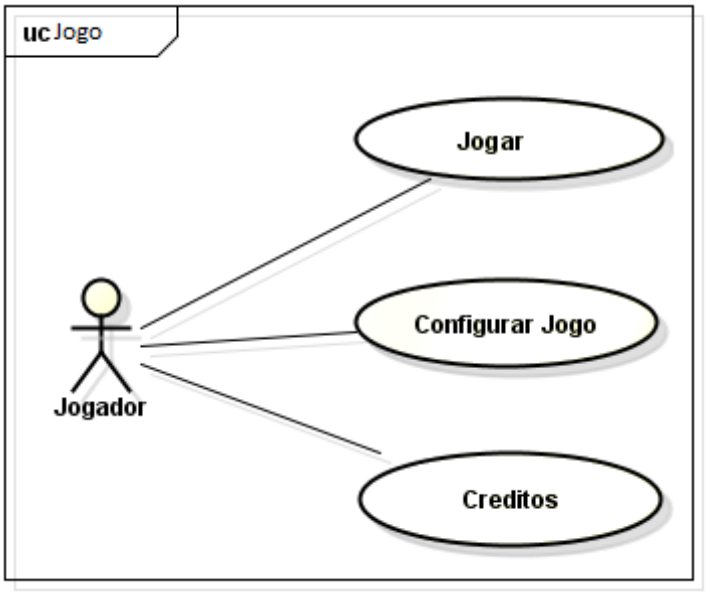

Figura 3.1 – Diagrama de caso de uso do Jogo.

3.2.1 Caso de Uso Jogar

Nome do caso de Uso	Jogar
Ator Principal	Jogador
Resumos	Esse caso de uso descreve as etapas percorridas por um jogador durante uma partida do jogo.
Pré-condições	Sistema iniciado e nenhuma existência de um jogo em andamento.
Pós-condições	Jogo Terminado.
Ações do ator	**Ações do sistema**
1. O caso de uso inicia quando o jogador seleciona a opção “Start Game” na interface do sistema.	2. Carrega os valores *default* dos parâmetros dos inimigos, jogador e cenário.
	3. Constrói o cenário do jogo onde constam, os inimigos e o personagem do jogador.
	4. Controla a interface entre o jogador e as fases do jogo.

3.2.2 Caso de Uso Configurar Jogo

Nome do caso de Uso	Configurar Jogo
Ator Principal	Jogador
Resumos	Esse caso de uso descreve as etapas percorridas por um jogador durante a configuração do jogo.
Pré-condições	Sistema iniciado e nenhuma existência de configuração em andamento.
Pós-condições	Configuração alterada.
Ações do ator	**Ações do sistema**
1. O caso de uso inicia quando o jogador seleciona a opção “Options” na interface do sistema.	2. Carrega os valores da configuração atual do jogo.
	3. Constrói tela com as informações carregadas, das opções disponíveis.
4. Seleciona a configuração.	
	5. Regista a configuração selecionada.

3.2.3 Caso de Uso Créditos

Nome do caso de Uso	Créditos
Ator Principal	Jogador
Resumos	Esse caso de uso descreve as etapas percorridas por um jogador para visualizar a tela de créditos.
Pré-condições	Sistema iniciado e nenhuma existência de tela de créditos em andamento.
Pós-condições	Tela de créditos iniciada
Ações do ator	**Ações do sistema**
1. O caso de uso inicia quando o jogador seleciona a opção “Credits” na interface do sistema.	2. Carrega os valores padrões da tela de "Credits".
	3. Constrói a tela de "Credits".

3.3 Diagramas

O sistema foi dividido em seis pacotes: batsonfire, utils, configuration, entity, model e activity. Esta estrutura foi sendo definida durante o desenvolvimento do jogo conforme a necessidade organizacional e funcional que foi surgindo. O pacote utils foi criado para armazenar as classes utilitárias do sistema. O pacote de configuration contem a classe config que é usada para definir os parâmetros de inicialização do jogo. Com o pacote activity a organização das Activity em um único pacote facilita a manutenção e organização funcional do projeto. A criação do pacote model é para abrigar as classes que contem dados compartilhados por todo o sistema. Ver figura 3.2.

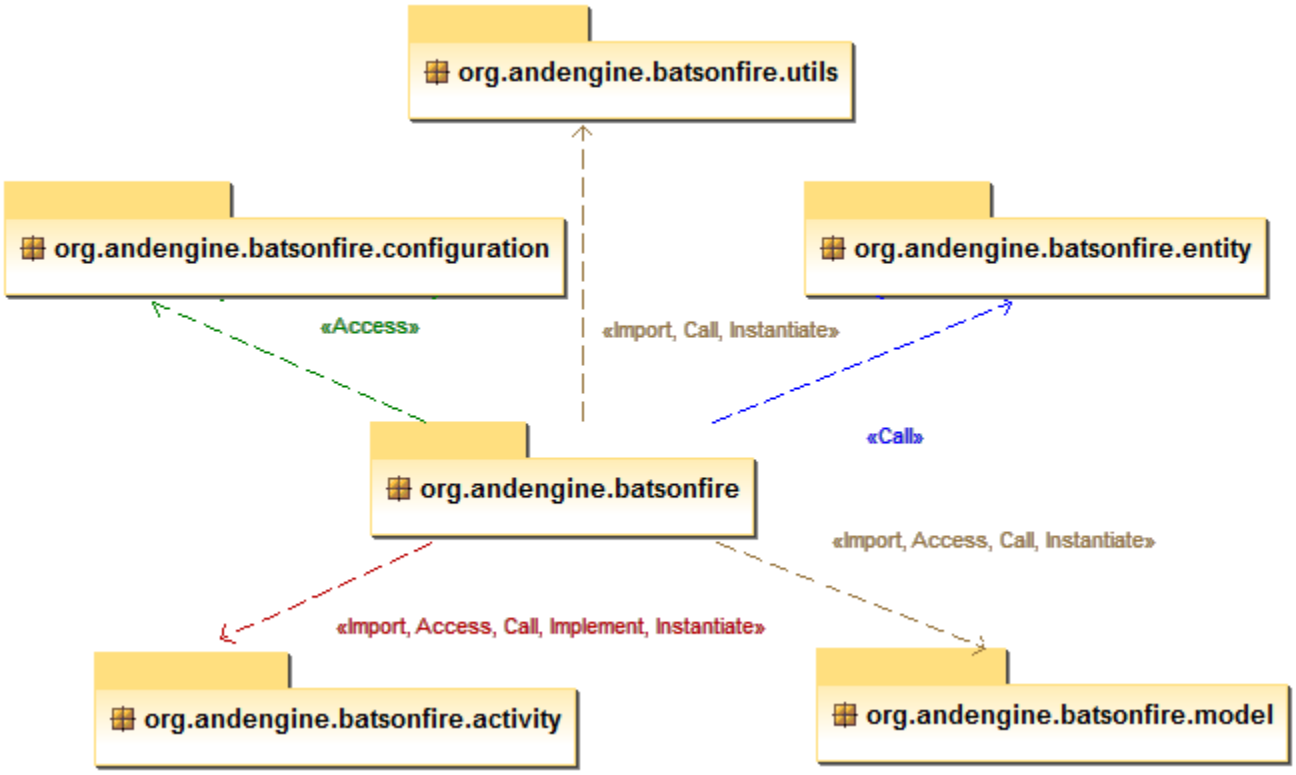

Figura 3.2 – Diagrama dos pacotes do jogo.

O pacote *Activity* contém as classes *SceneOptions*, *SceneGame*, *SceneGameOver*, *CameraSceneStatistics*, *SceneMenu*, *SceneCredits*, *CameraScenePause* e *SceneCutscene*, que são classes do tipo *Activity* responsáveis pela renderização(desenho) das telas do jogo e controle dos eventos. Todas as telas que o jogo possui são definidas e controladas neste pacote. Ver figuras 3.3, 3.4 e 3.5.

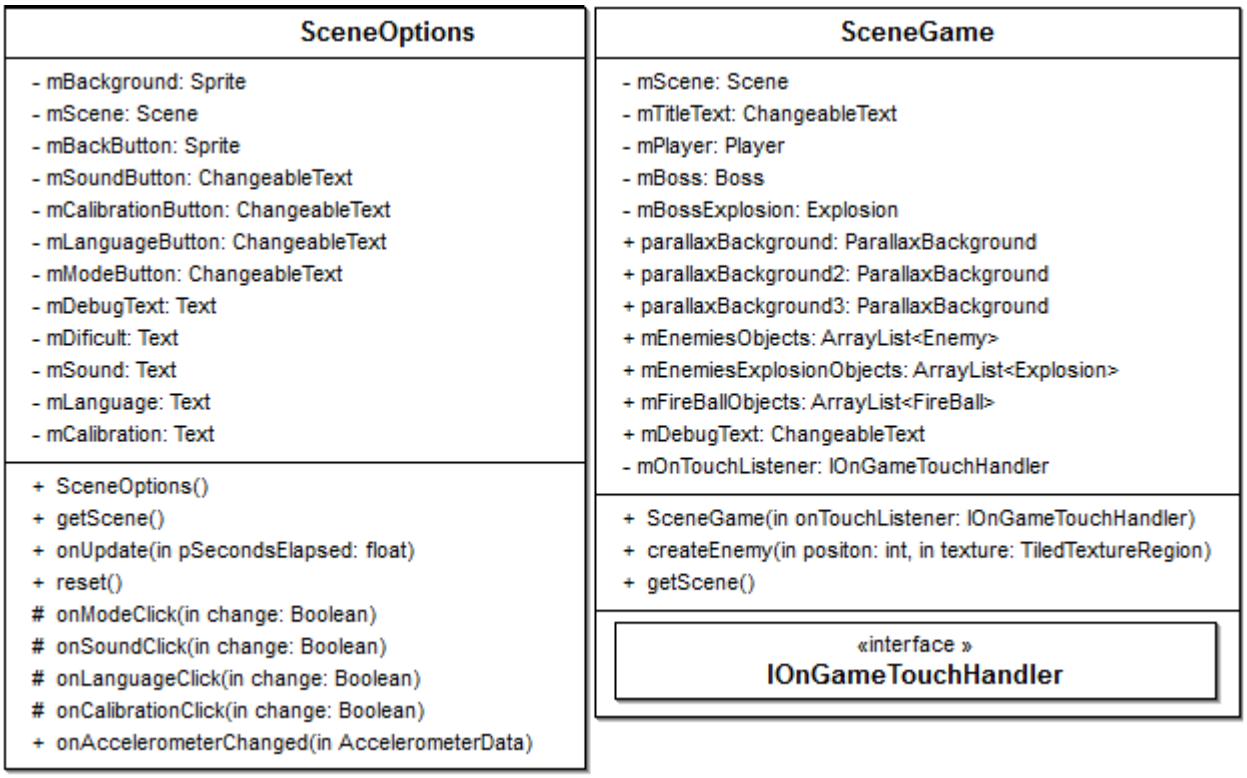

Figura 3.3 – Diagrama de Classes do pacote Activity parte 1.

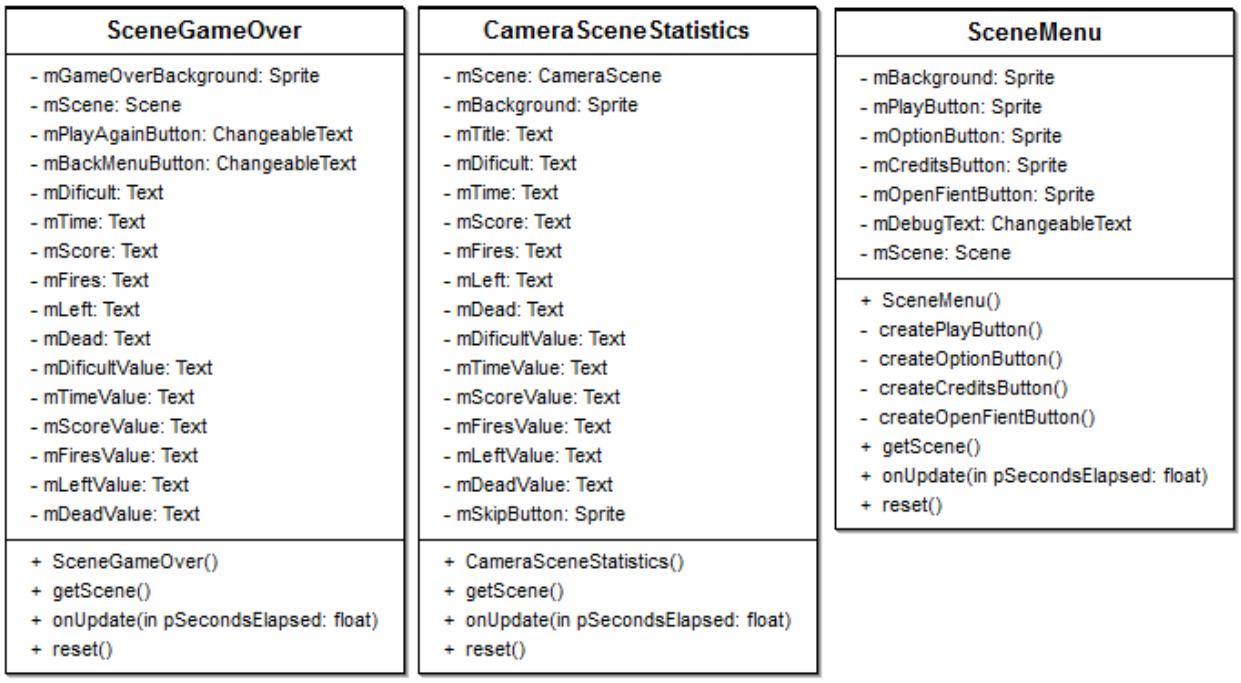

Figura 3.4 – Diagrama de Classes do pacote Activity parte 2.

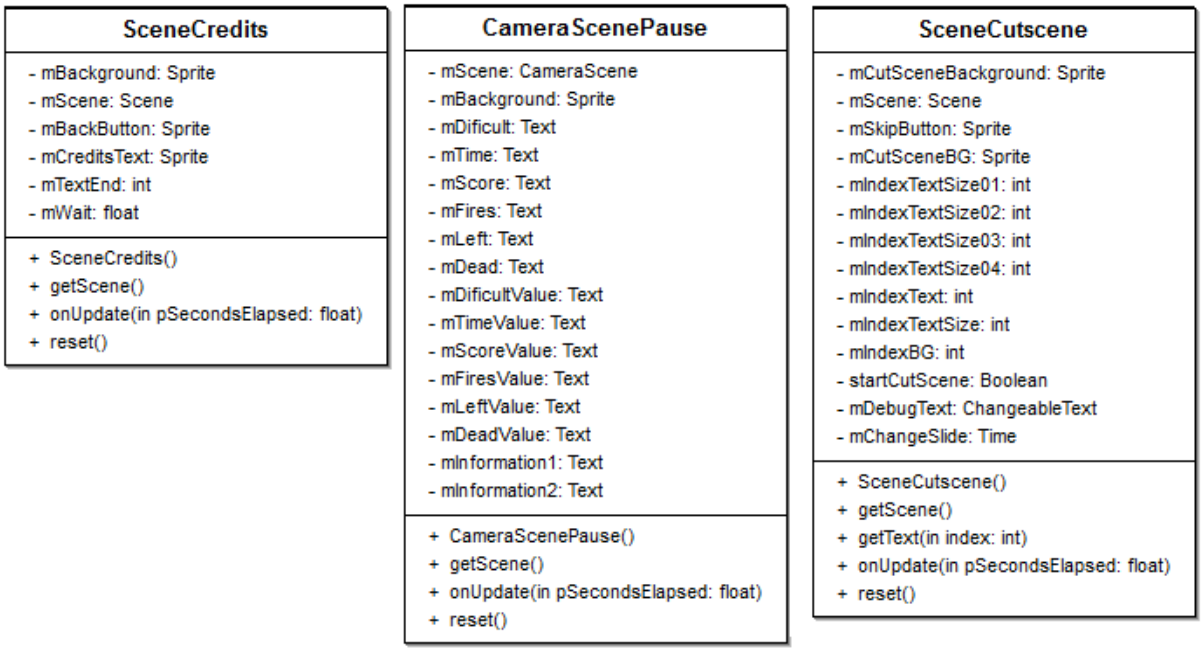

Figura 3.5 – Diagrama de Classes do pacote Activity parte 3.

Dentro do pacote *Model* estão as classes *Assets* e *GameData*. A classe *Assests* é responsável por carregar as texturas e os sons do jogo. Quando um objeto do pacote *Entity* é criado, é feita a referência para a textura carregada pela classe *Assets*.

Esta classe (*Gamedata*) contém outros atributos estáticos que são usados de forma global entre as classes do tipo *Activity* para a troca de informação como, por exemplo, dados estatísticos de pontuação do jogo e a fase atual no qual o jogador se encontra e é onde todas as classes do tipo *Activity* são instanciadas durante o carregamento inicial do jogo. Ver figura 3.6.

Figura 3.6 – Diagrama de Classes do pacote model.

O pacote *Entity* contém as classes *Enemy*, *Player*, *FireBall*, *Explosion* e *Boss*

que estendem a classe *Entity*, que é responsável pela criação de um objeto do tipo *Sprite* com a finalidade de renderizar imagens animadas durante o jogo. São estes elementos que o jogador irá interagir dentro do jogo como: os tiros disparados, os inimigos na tela, o chefe de cada fase e as explosões quando os inimigos ou o jogador morrer. Ver figura 3.7.

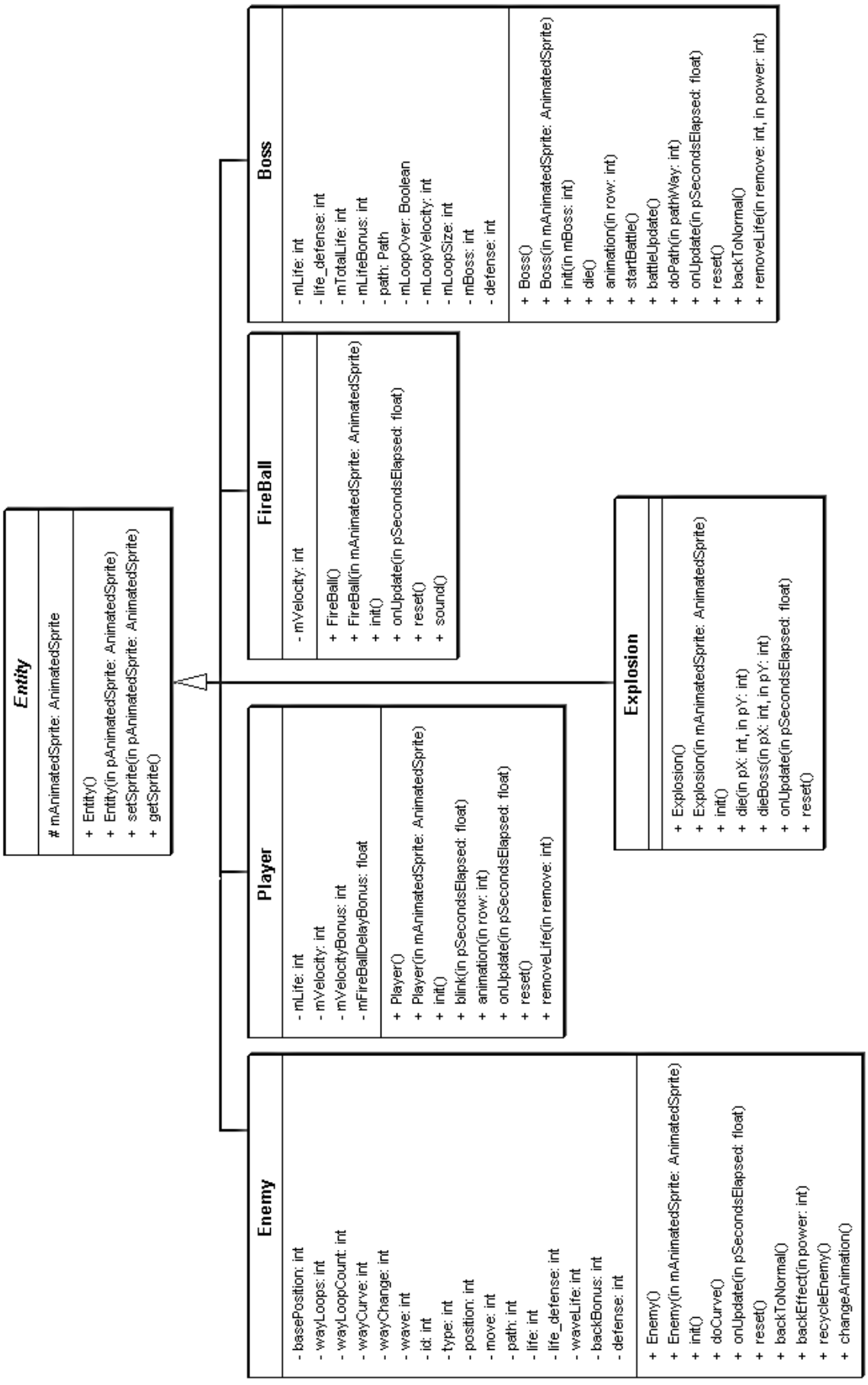

Figura 3.7 – Diagrama de Classes do pacote entity.

O pacote *Utils* contém as três classes utilitárias do sistema: a classe *Time* é responsável por controlar o tempo de espera (*delay*) entre os eventos; a classe *Text* é responsável por criar objetos para inserir textos com a formação já pré-definida; e a classe *Utils* é utilizada para adicionar métodos para conversão de unidade como, por exemplo, transformar um inteiro para formato de hora. Ver figura 3.8.

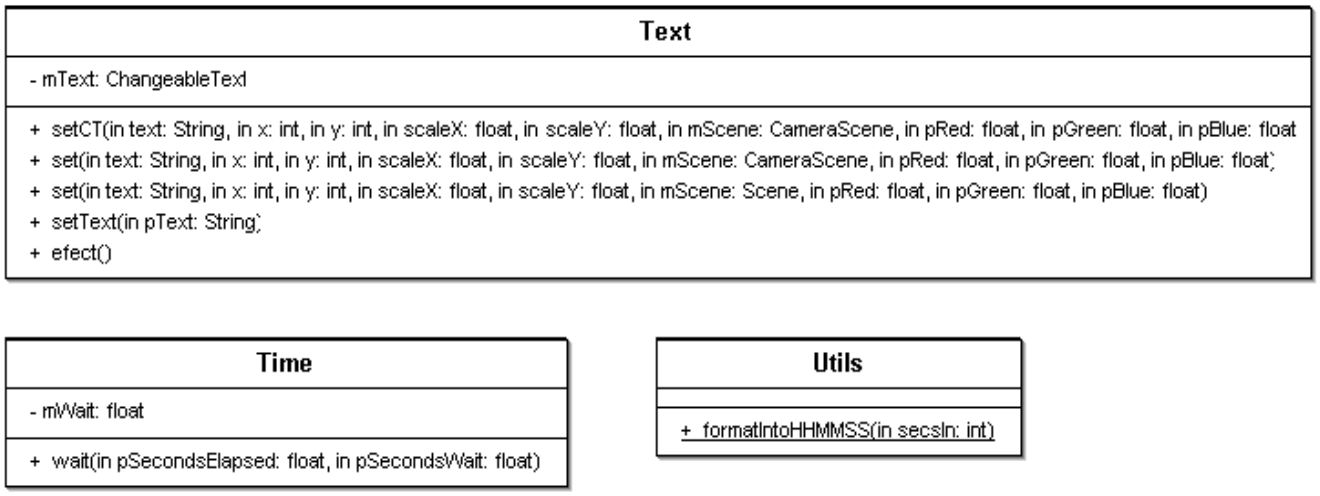

Figura 3.8 – Diagrama de Classes do pacote utils.

Dentro do pacote *Configuration* existe uma classe *Config* com atributos estáticos que são parâmetros para a inicialização do sistema e controle dos eventos durante o jogo. Ver figura 3.9

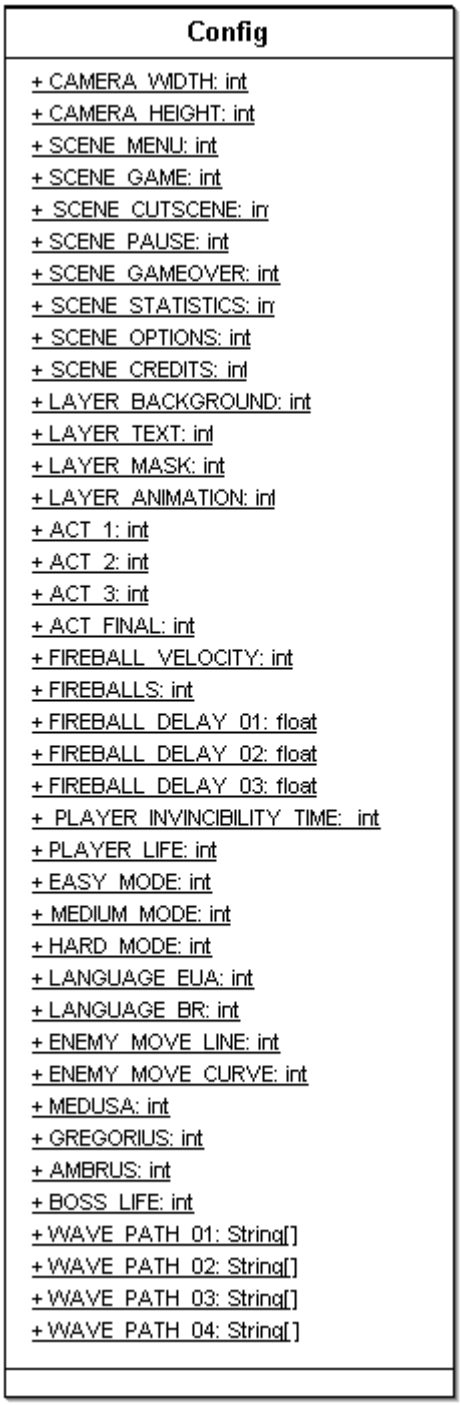

Figura 3.9 – Diagrama de Classes do pacote config.

O pacote *Batsonfire* é o principal, e contém três classes: *GameLoop*, *GameActivity* e *GameLoad*. A classe *GameActivity* é a classe principal definida no arquivo AndroidManifest.xml para iniciar o sistema. A classe *GameLoad* é do tipo Activity e mostra uma tela de espera (*Loading*) enquanto os recursos são carregados. A classe *GameLoop* é carregada pela classe *SceneGame* do pacote activity para executar o jogo. Veja a figura 3.10.

Figura 3.10 – Diagrama de Classes do pacote batsonfire.

A classe *GameLoop* usa o padrão *Observer* para executar os métodos automaticamente durante a mudança de estados no decorrer do jogo. Então, constantemente, são atualizados os estados do cenário, tempo de disparo das *fireballs*, movimentação do personagem, colisão entre *Sprites*, alertas de texto na tela, entre outros elementos. A variação de tempo de cada ciclo de processamento do jogo está relacionada à velocidade de processamento do equipamento no qual está ocorrendo a execução, então para controlar os eventos de forma segura é usada a classe utilitária *Time* para calcular os tempos de execução dos eventos. Isso evita que em cada configuração de hardware os eventos ocorram de maneira diferente. Na figura 3.11 mostra de forma global o relacionamento entre as classes do jogo.

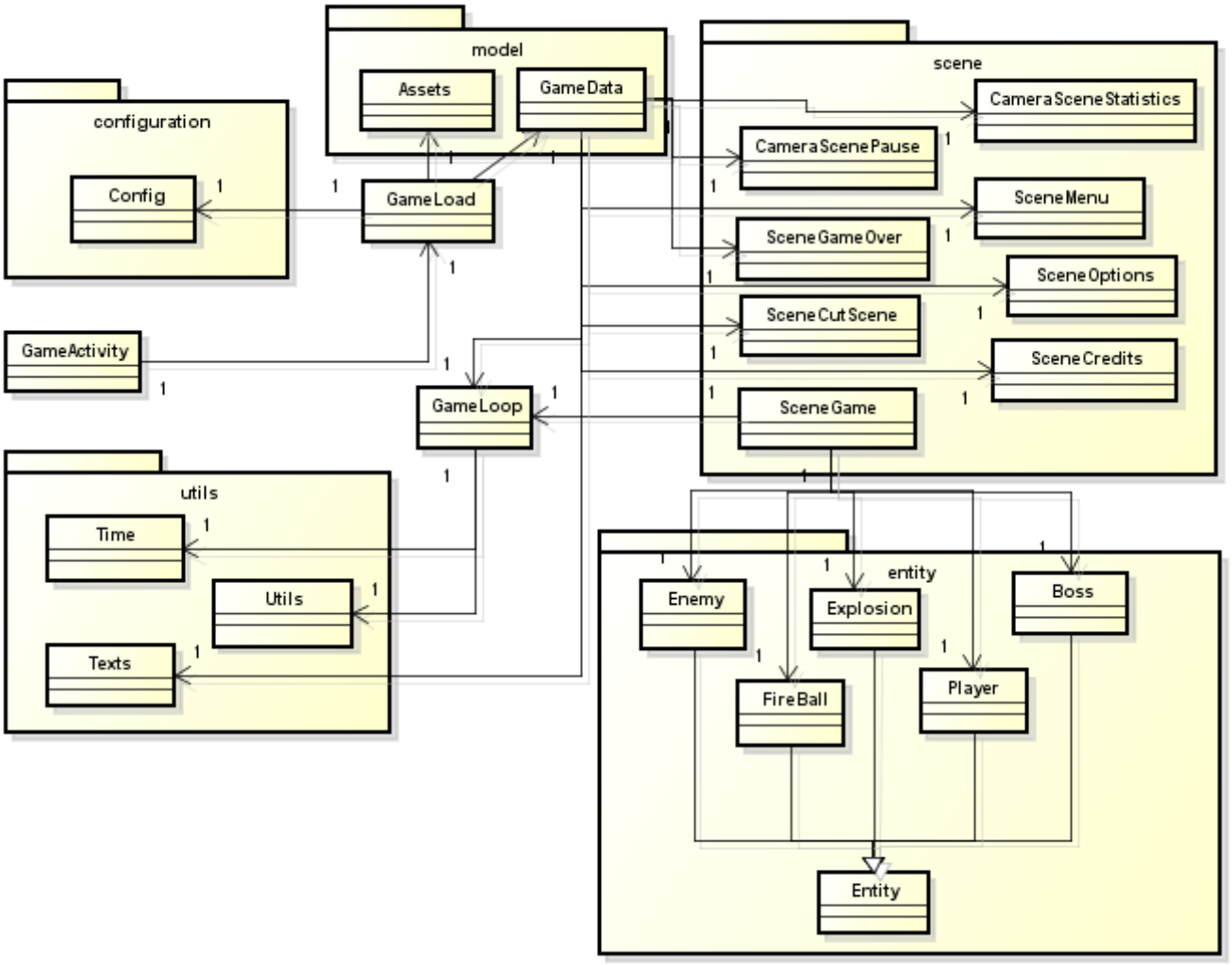

Figura 3.11 – Diagrama de Classes Geral.

Para iniciar um jogo a aplicação tem início na classe *GameActivity* que instancia um objeto da classe *GameLoad* que é responsável por carregar os recursos do jogo instanciando a classe *Assets* e apresentando uma tela de carregamento para o usuário enquanto executa esta tarefa. Após carregar os recursos a classe *GameLoad* chama a classe *GameData*, que instancia todas as classes do pacote *Activity* e apresenta a tela de menu. Todas as classes do pacote *Activity* são iniciadas desta forma. Quando o jogador clicar no botão “novo jogo”, o processo ocorrer de forma semelhante, mas agora é iniciada a tela *SceneGame* que irá fazer a carga dos inimigos usando como referência um vetor com a sequência dos inimigos na tela, definida na classe *Config* do pacote *Configuration*. Com base neste vetor é criado um conjunto (*pool*) de Objetos representando inimigos. Outros parâmetros iniciais são carregados da mesma forma, como nível de vida do jogador, entre outros. A figura 3.12 mostra o diagrama de sequência principal do jogo.

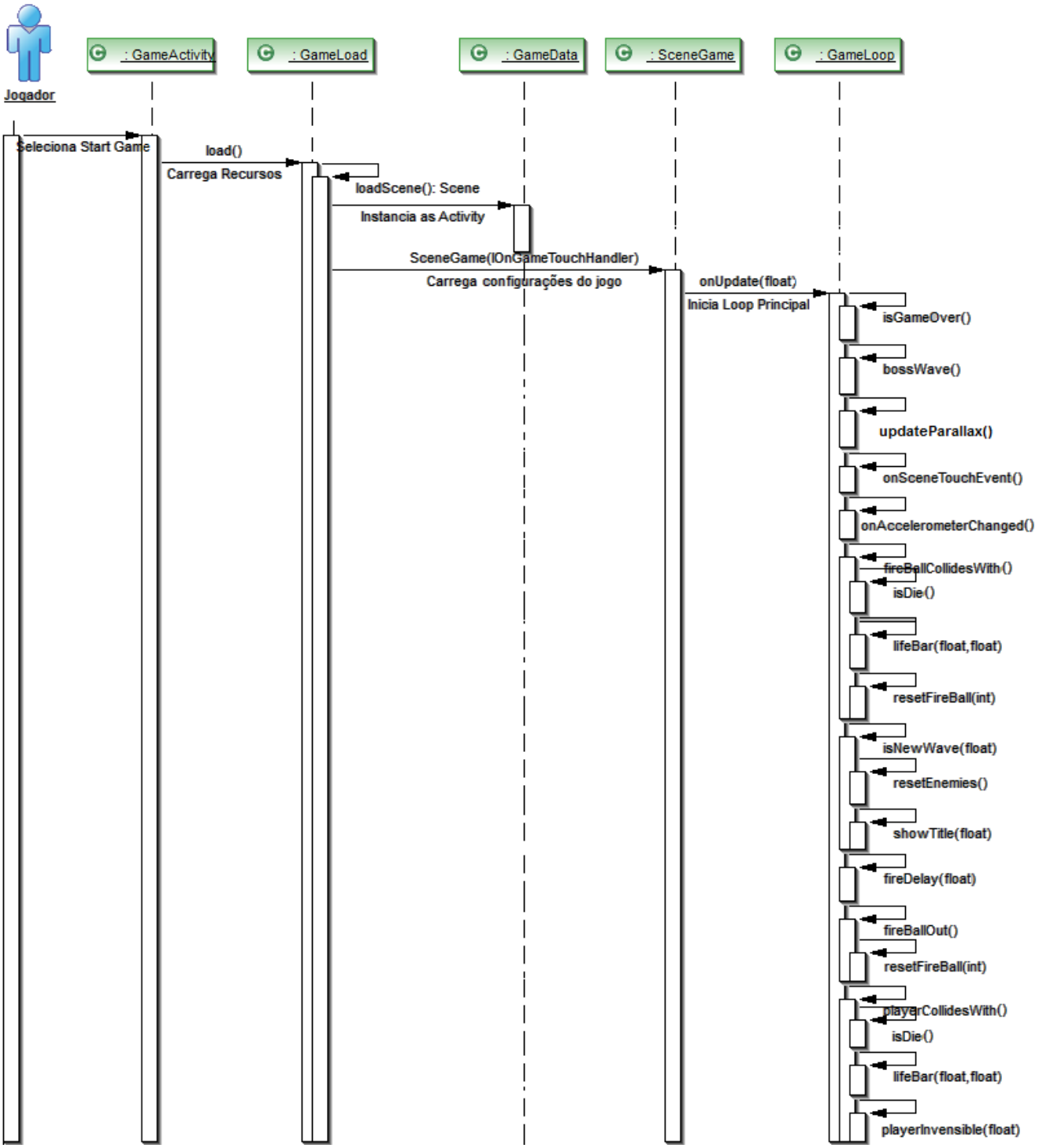

Figura 3.12 – Diagrama de sequência do jogo

4 RESULTADOS OBTIDOS

Para o desenvolvimento protótipo em um curto período de tempo aproximadamente 4 meses e foi dividido as atividades para um grupo de *blogueiros* de uma comunidade gamer que faço parte (KORJENIOSKI). As tarefas foram divididas em quatro competências: arte gráfica, som, enredo e programação. Após ter criado as tarefas foi publicado em um wiki para centralizar o trabalho e acompanhar o andamento do projeto. Cada um dos convidados escolheu a atividade que tinha maior afinidade e competência para executar. Infelizmente no grupo não existia ninguém com conhecimento sobre a plataforma Android então acabei fazendo todo o trabalho sozinho e sem um retorno sobre a qualidade do código. Desenvolver em uma plataforma nova sem um parceiro para trocar ideias por vezes leva o programador a nem sempre escolher o melhor caminho para resolver um problema. Outra dificuldade que tive para efetuar os testes e apresentar os resultados foi que eu era o único do grupo com um celular rodando Android. Então a solução para apresentar o que havia sido criado foi gravar vídeos com o jogo rodando e narrando os problemas e resultados obtidos e enviar para o Youtube para o grupo tivesse um retorno de como o jogo estava se comportando. Quando comecei a gravar os vídeos teve um resultado positivo. O grupo começou a ficar mais animado com o desenvolvimento, acredito que foi o fato de ver algo concreto do seu trabalho e esforço em ação. A principal dificuldade durante o desenvolvimento do jogo foi a falta de um padrão conhecido de desenvolvimento para jogo no Android já que o uso de uma Activity representa toda a funcionalidade da tela do software, então foi durante o desenvolvimento que a organização dos packages e criação das classes foram surgindo. Para o debug do programa foi usado, inicialmente, o emulador do Android no Windows, foi critico já que a renderização é lenta a solução foi fazer os testes direto no celular para ter o resultado esperado.

O sistema foi testado usando um celular Galaxy 5 com Android versão 2.2, instalando diretamente o arquivo .apk. O jogo está dividido em oito telas de atividades:

Menu - É a primeira tela do jogo e foi feita para a navegação onde é possível acessar as telas “Credits”, “Options” e “Game”. Foi usada uma imagem para criar o fundo do cenário e mais três imagens para criar os botões adicionando o evento de toque para acessar. A figura 4.1 mostra suas opções.

Figura 4.1 – Tela do menu do jogo responsável pela navegação das telas do jogo.

Options - Tela de configuração do jogo onde é possível calibrar o acelerômetro para a movimentação do jogador, desabilitar o áudio e mudar o idioma. A construção é semelhante a do menu e a única diferença é que pode ser habilitado o acelerômetro na hora de calibrar a posição do dispositivo. A figura 4.2 apresenta as configurações disponíveis da tela Options.

Figura 4.2 – Tela “Options” onde o jogador pode configurar o jogo para seu perfil.

Credits - Informações sobre a equipe responsável pelo desenvolvimento do jogo. É a tela que atribui a cada participante no desenvolvimento do jogo seu devido mérito. Foi usada uma imagem de máscara com um corte central no formato de um retângulo com transparência e outra imagem com a lista dos nomes que está localizada uma camada abaixo. Foi utilizado o efeito *Scrolling* para que a imagem com a lista dos nomes suba criando assim o efeito desejado. A figura 4.3 mostra a tela “Credits”.

Figura 4.3 –Tela "Credits": apresentação dos nomes dos responsáveis pelo desenvolvimento.

Cutscene - Esta tela foi feita baseada nas Cutscenes do jogo Ninja Gaiden, onde, na parte inferior, mostram-se os textos contando a história do jogo e, na parte superior, as ilustrações. O jogo leva em conta a internacionalização, então é possível adicionar outros idiomas com pouco esforço. A figura 4.4 mostra a tela da Cutscene.

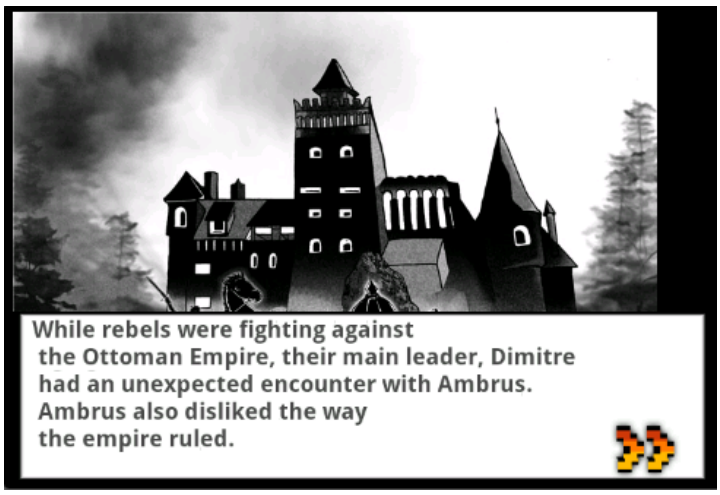

Figura 4.4 – Tela "Cutscene" é onde é contada a história do jogo usando textos e imagens.

Game - É o jogo propriamente dito. Aqui o jogador interage com os elementos usando os sensores do acelerômetro para movimentar o personagem e o touchscreen para disparar projéteis nos inimigos (figura 4.5).

Figura 4.5 – Tela “Game”: jogador disparando Fireballs contra inimigos.

O jogo tem três cenários: “Floresta de Znord”, “Ruínas do Castelo de Ambrus” e “Castelo do Ambrus”. Ver figura 4.6. No final de cada cenário o jogador irá enfrentar um Chefe (*Boss*), que deve ser vencido para se alcançar a próxima fase, ou vencer o jogo (figura 4.7).

Figura 4.6 – Tela “Game”: Inimigos da fase.

Figura 4.7– Tela “Game”: Boss da fase.

Game Pause - Esta tela é apresentada quando o jogador pressiona o botão menu do dispositivo. A construção dela é feita adicionando uma camada acima da tela do Game e em seguida é parado o “main loop” do jogo. A figura 4.8 mostra a tela de pausa ao ser ativada.

Figura 4.8 – Tela “Pause”: visualização das estatísticas durante o jogo.

Game Statistics - Ao final de cada onda de inimigos são mostradas as estatísticas do jogo. Esta tela foi construída com modal, da mesma forma que a tela “Pause”. Porém a engine não é parada e, após algum tempo, ela desaparece e o jogo continua. A figura 4.9 mostra as estatísticas do jogo ao se finalizar uma onda de inimigos.

Figura 4.9 – Tela "Game Statistics" mostra as estatísticas do jogo no final de cada onda de inimigos.

Game Over - Tela final do jogo que contém informações como o nível de dificuldade, número de inimigos eliminados, pontuação e tempo. Para criar esta tela foi utilizado o mesmo princípio da tela "Menu". A figura 4.10 mostra a tela de *Game Over* quando o jogo termina.

Figura 4.10 – Tela "Game Over": mostra as estatísticas do jogo e opções de navegação.

5 CONCLUSÃO

Este trabalho foi desenvolvido para apresentar a tecnologia Android para o desenvolvimento de jogos 2D. Com relação ao uso da tecnologia foi possível criar um jogo no estilo *Side-scrolling* com a plataforma sem muita dificuldade já que o ambiente de desenvolvimento é similar ao usado no desenvolvimento de aplicativos Java. O uso da técnica *parallax* com imagens estáticas para criação dos cenários foi muito útil para criar o efeito de profundidade nos cenários.

Uma sugestão para futuras implementações é trocar o *Parallax* com imagens estáticas por *TiledMaps* (mapas de imagens retangulares) para conseguir cenários maiores e com eventos pré-definidos no próprio cenário, como inimigos, itens e chefes de fase. Outro ponto de melhoria para atrair mais jogadores é o uso da API *OpenFeint* (OPENFEINT, 2011) que é uma rede social para jogadores e que tem suporte para jogos do Android e iPhone.

O OpenFeint dá destaque aos jogadores com maiores pontuações conforme eles vão ganhando pontos nos jogos. Ele disponibiliza listas de *Leaderboards* (Líderes) com as melhores pontuações do jogo. A premiação em pontos pode ser alcançada pelos *Achievements* (Realizações) durante o jogo. Para alcançar um *Achievement* o jogador deve completar uma tarefa específica como, por exemplo, finalizar o jogo em menos de 5 minutos ou conseguir matar o chefe da fase sem perder vida. Para armazenar as estatísticas do jogador para liberação dos *Achivements* pode ser utilizada o banco de dados SQLite (OPENFEINT, 2011). Desta forma, também é possível ir liberando prêmios para jogador como novas armas, vidas e etc. Todas estas técnicas de recompensa irão fazer com que o jogador volte a jogar mais vezes já que terá mais objetivos para alcançar e também ira ter destaque na comunidade de jogadores que participa.

O conhecimento sobre a tecnologia Java já foi fundamental para a criação do jogo, sem este conhecimento usar o Android teria sido uma tarefa mais difícil. A distribuição do jogo no Android Market não foi possível já que precisava de recursos sonoros e visuais mais elaborados para que o impacto do jogo no seu lançamento agrade os jogadores do gênero *Side-scrolling*. O projeto do jogo foi apresentado no blog www.sobcontrollers.com.br (KORJENIOSKI) e todas as futuras etapas do desenvolvimento até o lançamento será comentado no blog.

6 REFERÊNCIAS BIBLIOGRÁFICAS

2D, Examples Disponível em: <http://en.wikipedia.org/wiki/2D_computer_graphics>. Acesso em: 07 de mai. 2011.

ANDENGINE, Examples Disponível em: <http://code.google.com/p/andengine/>. Acesso em: 15 de abr. 2011.

ANDENGINE, Wiki Disponível em: < http://wiki.andengine.org/AndEngine>. Acesso em: 15 de abr. 2011.

ANDROID, Timeline Disponível em: <http://www.android.com/timeline.html>. Acesso em: 10 de abr. 2011.

ANDROID, Fundamentals Disponível em: http://developer.android.com/guide/topics/fundamentals.html>. Acesso em: 10 de abr. 2011.

APACHE, Apache License em: <http://en.wikipedia.org/wiki/Apache_License>. Acesso em: 09 de mai. 2011.

AUDACITY, AUDACITY Disponível em: <http://audacity.sourceforge.net/>. Acesso em: 05 de abr. 2011.

CANALYS, Android increases smart phone market leadership with 35% share em: <http://www.canalys.com/pr/2011/r2011051.html>. Acesso em: 09 de mai. 2011.

EHRINGER, The Dalvik Virtual Machine Architecture em: < http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf>. Acesso em: 09 de mai. 2011.

DEVELOPER, Platform Versions Disponível em: <http://developer.android.com/resources/dashboard/platform-versions.html>. Acesso em: 10 de abr. 2011.

DEVELOPER, SDK Versions Disponível em: <http://developer.android.com/sdk/index.html>. Acesso em: 10 de abr. 2011.

DEVELOPER, NDK Overview Disponível em: <http://developer.android.com/sdk/ndk/overview.html>. Acesso em: 10 de abr. 2011.

DEVELOPER, Media Player Disponível em: <http://developer.android.com/reference/android/media/MediaPlayer.html>. Acesso em: 27 de abr. 2011.

FARAGO, Peter. Apple and Google Capture U.S. Video Game Market Share in 2010 Disponível em: <http://blog.flurry.com/bid/60307/Apple-and-Google-Capture-U-S-Video-Game-Market-Share-in-2010>. Acesso em: 20 de abr. 2011.

ECLIPSE, ECLIPSE Disponível em: <http://www.eclipse.org/>. Acesso em: 05 de abr. 2011.

GAMMA, ERICH GAMMA, RICHARD HELM, RALPH JOHNSON, JOHN VLISSIDES Design Patterns.1.ed. Editora: Bookman. 2000.

GLBENCHMARK, GLBenmark Disponível em: <http://www.glbenchmark.com/>. Acesso em: 05 de abr. 2011.

KHRONOS, OpenGL ES API Registry Disponível em: <http://www.khronos.org/registry/gles/>. Acesso em: 16 de abr. 2011.

KHRONOS, OpenGL ES Specification Disponível em: < http://www.khronos.org/registry/gles/specs/1.0/opengles_spec_1_0.pdf>. Acesso em: 10 de fev. 2011.

KORJENIOSKI , Bats on Fire Conheça o projeto e a historia em: < http://www.sobcontrollers.com/game/design/bats-on-fire-conheca-o-projeto-e-a-historia/> Acesso em: 10 de fev. 2011.

MARKET, Android Market Publish em: <http://market.android.com/publish>. Acesso em: 10 de mai. 2011.

MIDI, Musical Instrument Digital Interface <http://en.wikipedia.org/wiki/MIDI> Acesso em: 05 de abr. 2011.

OPENFEINT, Openfeint Disponível em: <http://www.openfeint.com/about>. Acesso em: 05 de abr. 2011.

PARALLAX, Parallax Scrolling Disponível em: <http://en.wikipedia.org/wiki/Parallax_scrolling>. Acesso em: 05 de abr. 2011.

PNG, Portable Network GraphicsDisponível em: <http://en.wikipedia.org/wiki/Portable_Network_Graphics>. Acesso em: 05 de abr. 2011.

SHI, Yunhe; GREGG, David; BEATTY, Andrew; ERTL, M. Anton. Virtual Machine Showdown: Stack Versus Registers. Usenix. Disponível em: < http://db.usenix.org/events/vee05/full_papers/p153-yunhe.pdf> Acesso em: 17 agos. 2011.

SPRITE, Sprite <http://en.wikipedia.org/wiki/Sprite_%28computer_graphics%29> Acesso em: 05 de abr. 2011.

SQLITE, SQLite <http://www.sqlite.org/> Acesso em: 05 de abr. 2011.

VAMPIRETOOLS, Conde Drácula no século XV Disponível em: <http://vampiretools.blogspot.com/2007/09/conde-drcula-no-scxv.html>. Acesso em: 02 de abr. 2011.

WAVE ,Wave form Audio File Disponível em: <http://en.wikipedia.org/wiki/WAV>. Acesso em: 05 de abr. 2011.