UNIVERSIDADE FEDERAL DO PARANÁ – SETOR LITORAL
CURSO DE INFORMÁTICA E CIDADANIA

DESENVOLVIMENTO DE JOGOS 2D NA PLATAFORMA ANDROID

LUCIANO OLIVEIRA DOS SANTOS

MATINHOS/PR
2015

LUCIANO OLIVEIRA DOS SANTOS

DESENVOLVIMENTO DE JOGOS 2D NA PLATAFORMA ANDROID

Trabalho de Conclusão de Curso (TCC) apresentado ao Curso de Graduação em Informática e Cidadania da Universidade Federal do Paraná - Setor Litoral, como parte dos requisitos para a obtenção do grau de Bacharel em Informática e Cidadania.

Orientador: Professor Neilor Fermino Camargo.

MATINHOS/PR
2015

ATA DE AVALIAÇÃO DA DEFESA DO TRABALHO DE CONCLUSÃO DE CURSO

Aos trinta do mês de junho de dois mil e quinze, às 21 horas, no Setor Litoral da Universidade Federal do Paraná, reuniu-se a banca avaliadora do trabalho de conclusão de curso, constituída pelo professor Me. Márcio Hosoya Name e pela professora Dra. Silma Cortes da Costa Battezzati sob a presidência do Orientador, Professor Me. Neillor Fermino Camargo. O Trabalho de Conclusão do Curso de Bacharelado em Informática e Cidadania, do aluno Luciano Oliveira dos Santos, sob o título: "DESENVOLVIMENTO DE JOGOS 2D NA PLATAFORMA ANDROID", obteve o conceito APL. O aluno deverá efetuar as correções solicitadas pela banca e entregar a versão final em formato digital via CD-ROOM, até o dia 10 de julho de dois mil e quinze, na assessoria a Câmara do curso de Informática e Cidadania.

Neillor Fermino Camargo
Professor Orientador

Márcio Hosoya Name
Membro da banca avaliadora

Silma Cortes da Costa Battezzati
Membro da banca avaliadora

Luciano Oliveira dos Santos

AGRADECIMENTOS

Agradeço a Deus por ter me iluminado na busca de conhecimento para desenvolver este trabalho e pelas muitas bênçãos concedidas e por sua infinita misericórdia.

À minha família em especial ao meu pai / Pedro (em memória), e minha mãe Valdina que sempre me apoiaram com amor e a minha esposa Luciana que esteve ao meu lado desde o inicio deste curso tem me ajudado com muita paciência.

Ao orientador pedagógico Neilor Fermino Camargo e aos docentes Paulo Gaspar Graziola Junior e Almir Carlos Andrade e os demais docentes do curso por dedicarem seu tempo e compartilharem os seus saberes, me proporcionando conhecer e compreender esta profissão.

Agradeço à todos que de alguma forma contribuíram para esta conquista.

Muito Obrigado.

RESUMO

O presente trabalho de Conclusão de Curso (TCC), foi escrito para a obtenção do titulo de bacharel em Informática e Cidadania pela Universidade Federal do Paraná setor Litoral, e descreve o desenvolvimento de um aplicativo (jogo em 2D) com a utilização na plataforma Android, para dispositivos móveis como celulares, tablets entre outros. Portanto este projeto tem como objetivos específicos: Pesquisar a plataforma e jogos em 2D para dispositivos moveis; Avaliar as ferramentas para o desenvolvimento do jogo; Definir a plataforma, as ferramentas e o tema do jogo e Implementar o jogo em 2D. O trabalho foi desenvolvido utilizando a pesquisa bibliografica, com auxilio do uso da web, pesquisas em sites e ferramentas para o desenvolvimento do jogo 2D. Foi verificado que nos testes de funcionalidade do jogo apresentou-se um bom desempenho de performance e, os botões, como também as telas ficaram dentro das expectativas de configuração, sendo que foi observado a necessidade de implementar alguns ajustes e foi sugerido possíveis melhorias futuras para tornar este trabalho melhor e mais completo.

Palavras-Chave: Tecnologia, Java, Dispositivos Moveis.

ABSTRACT

The working Course Conclusion (TCC), was written for obtaining the bachelor's title in Information Technology and Citizenship from the Federal University of Paraná coast sector, and describes the development of an application (2D game) using the Android platform to mobile devices such as mobile phones, tablets and others. Therefore this project has the following objectives: Find the platform and 2D games for mobile devices; Evaluate the tools to game development; Define the platform, the tools and the theme of the game and implement the game in 2D. The study was conducted using bibliographic research, with the use of the web assistance, research on websites and tools for developing 2D game. It was found that in the game's functionality tests introduced himself performance good performance and buttons, as well as the screens were within the setting of expectations, and noted the need to implement some adjustments and suggested possible future improvements to make this better and more thorough job.

Keywords: Technology, Java, Furniture devices.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AAC – *Advanced Audio Coding*, codificação de audio avançado.

ADT – *Android Development Tools*, ferramenta de desenvolvimento Android.

AMR – *Adaptive Multi – Rate*, adaptativo para muitas taxa.

API – *Application Programming Interface*, Interface de programação de aplicações

APPS – *Applications*, aplicativos

APK – *Android Package*, é um arquivo compilado usado para instalar-se no *Android* em forma de zip.

BITMAP – Mapa de dígitos binário

BSD – *Berkeley Software Distribution*, é uma licença de código aberto

C/C++ – Linguagem de programação compilada de propósito geral.

.DB – Refere-se a *database* (banco de dados).

DDL – *Data Definition Language* , linguagem de definição de dados

DML – *Data Manipulation Language* ou linguagem de manipulação de dados

EDGE – *Enhanced Date for GSM Evolution* ou taxas de dados ampliados para a evolução do GSM.

EJBS – *Enterprise Java Beans* é um componente da plataforma *Java* que roda em um container de um servidor de aplicação.

GIF – *Graphics Interchange Format* ou formato para intercambio de gráficos.

GIMP – *General Image Manipulation Program* ou programa de manipulação de imagem geral

GPL – *General Public License* ou Licença Publica Geral.

GPS – *General Packet Service* ou serviço de pacote geral

GSM – *Global System for Mobile* ou sistema global para comunicação móvel

HTML – *Hyper Text Markup Language* ou Linguagem de Marcação de Hiper Texto.

IBM – *International Business Machines* ou Maquina de Negocio Internacional.

IDE – *Integrated Development Environment* ou Ambiente de desenvolvimento integrado (programa que reúne características e ferramentas de apoio ao desenvolvimento de software).

JDK – *Java Development Kit* ou Kit de Desenvolvimento Java

JPG – *Joint Photographic Experts Group*, é um formato de compressão de imagens, tanto em coloridas como em escala de cinza.

JVM – *Java Virtual Machine* ou Maquina Virtual Java, programa que carrega e executa as aplicações Java, convertendo os *bytecodes* em códigos de maquina.

JRE – *Java Runtime Environment* ou Ambiente de Tempo de Execução Java, usado para executar as aplicações da plataforma Java.

LIBC – É uma biblioteca de rotinas padronizadas da linguagem de programação C.

MPEG4 – *Moving Picture Experts Group-4*, é um padrão utilizado para a compressão de dados digitais de áudio e vídeo.

MP3 – É uma abreviação de MPEG, *Moving Picture Experts Group-1*, é um formato de compressão de áudio digital.

NDK – Kit de Desenvolvimento Nativo

NET – Denominação dada a internet, também conhecida como rede.

PNG – *Portable Network Graphics*, é u formato de arquivo para imagens gráficas de mapa de bits.

SDK – *Software Development Kit* ou Kit de desenvolvimento de software, aplicativo.

SGBD – Sistema Gerenciador de Banco de Dados.

SMS – *Short Message Service* ou Serviço de Mensagem Curta.

SQL – *Structured Query Language* ou Linguagem de Consulta Estruturada.

SVG – *Scalable Vector Graphic* ou Gráfico Vetorial Escalavel.

UML – *Unified Modeling Language* ou Linguagem de Modelagem.

VM – Virtual Machine ou Maquina Virtual.

WEB – *Word Wide Web*. Rede de alcance mundial.

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho descreve o desenvolvimento de um aplicativo (jogo 2D) com a utilização na plataforma Android, para dispositivos moveis como celulares, tablets entre outros.

Segundo Lecheta (2010), Android é uma plataforma de desenvolvimento completamente livre e de código fonte aberto para aplicativos móveis como smartphones, contendo um sistema operacional baseado em Linux 2.6, uma interface visual, GPS, com diversas aplicações já instaladas e ainda um ambiente de desenvolvimento bastante robusta e poderosa que utiliza a linguagem Java.

Portanto, este projeto tem como objetivos específicos: Pesquisar a plataforma e jogos em 2D para dispositivos moveis; Avaliar as ferramentas para o desenvolvimento do jogo; Definir a plataforma, as ferramentas e o tema do jogo; Implementar o jogo em 2D.

Para realização deste trabalho foi adotada a pesquisa bibliográfica com auxilio do uso da Web e pesquisas em sites.

Objetivando descrever o desenvolvimento do aplicativo (jogo 2D), busca-se apresentar o sistema Android, Ferramentas utilizadas para o desenvolvimento do jogo 2D, Jogos digitais e o desenvolvimento do jogo 2D, sendo feita uma breve descrição sobre todos estes itens.

No sistema Android foi falado sobre o histórico, a definição, a arquitetura da plataforma, núcleo Linux, bibliotecas, Android tempo/execução, quadro de aplicações e aplicações.

Encontra partida nas ferramentas utilizadas para o desenvolvimento do jogo 2D, foi falado do Dia Diagram Editor, do Gimp, do Eclipse, do Android SDK, do Java JDK, do SQLite, do ADT Plugin e da AndEngine (GLES2).

No entanto, em Jogos digitais foi feito uma breve descrição do histórico, dos tipos de jogos existentes e, também, sobre alguns jogos para dispositivos moveis.

E no desenvolvimento do jogo 2D foi feita a apresentação do jogo, iniciando um aplicativo de exemplo, a arquitetura do jogo e suas camadas, protótipos de telas, o diagrama de classes e suas camadas e também dos pacotes, métodos e classes, os quais são: pacotes bancodados, iniciofimjogo e jogando.

Ao fim deste trabalho, advindo do desenvolvimento deste aplicativo (Jogo), pretende-se disseminar e incentivar o estudo e a criação de jogos as pessoas que querem iniciar no desenvolvimento de jogos e demonstrar o conhecimento obtido no decorrer deste curso.

2 JUSTIFICATIVA

O interesse pelo tema surgiu a partir da soma do conhecimento obtido através de um modulo de Java na UFPR - Litoral e de um curso realizado pela web sobre Android que trouxe o desejo de desenvolver esse projeto que visa contribuir para o incentivo do estudo da linguagem Java e da plataforma Android no desenvolvimento de jogos 2D com a utilização em dispositivos móveis.

Tendo em vista o avanço da tecnologia dos dispositivos móveis no Brasil o aumento da utilização e o crescimento desta plataforma, observou-se a vantagem na criação de aplicativos. Pois o mercado dos aparelhos móveis cresce a cada dia, chegando nas mãos de mais de três bilhões de consumidores(LECHETA, 2013). As pessoas comuns buscam por aparelhos com diversos recursos para corresponder as suas necessidades, para que seja possível melhorar suas atividades do dia a dia ou ter um entretenimento portátil e moderno (LECHETA, 2010).

Segundo Lee (2005), existem quatro principais vantagens dos dispositivos móveis:

Portabilidade: capacidade de ser facilmente transportado;

Usabilidade: deve ser utilizável por diferentes tipos de pessoas;

Funcionalidade: servem a múltiplos propósitos e,

Conectividade: permitem conectar as pessoas e/ou sistemas e, transmitir e receber informações.

Desta forma, observa-se que para a escolha do desenvolvimento de um projeto é necessário optar por uma solução que tragam melhor benefício, em termos de eficiência, custo e tempo de desenvolvimento para a sua construção. Por meio desta optativa, foi utilizado a plataforma Android SDK, para desenvolvimento de jogos 2D.

3 OBJETIVOS

3.1 OBJETIVO GERAL

O objetivo geral deste trabalho, é descrever o desenvolvimento de um aplicativo (jogo) com a utilização na plataforma Android, para dispositivos móveis como celulares, tablets, entre outros.

3.2 OBJETIVO ESPECIFICOS

- Pesquisar plataforma e jogos em 2D para dispositivos móveis;
- Avaliar as ferramentas para o desenvolvimento do jogo;
- Definir as ferramentas e o tema do jogo;
- Implementar o jogo em 2D;

4 METODOLOGIA

Neste trabalho de conclusão de curso foi utilizada a pesquisa bibliografica, com auxilio do uso da web, pesquisas em sites e ferramentas para o desenvolvimento do jogo 2D. Ao considerar que há diversas alternativas de linguagens de programação para o Android, como por exemplo: C/C++, Net Framework, Python[1] e Scala[2], este projeto será realizado com a linguagem Java sendo a oficial da Google Inc. com a utilização dos seguintes Softwares: Gimp, Dia, Java Development Kit, Ide[3] Eclipse, Android SDK e o Plugin ADT e AndEngine para o Eclipse.

Esta tecnologia qual vem sendo evoluída ao longo dos tempos, trazendo uma melhor flexibilidade e inovações ao desenvolvimento voltado ao mercado de aplicativos móveis. Através da utilização cada vez maior de dispositivos móveis, os dispositivos móveis oferecem o poder de uso com vantagens da conectividade, a qualquer momento e em qualquer lugar, tornando seu uso muito importante não apenas pessoal, mas também profissional. (ABLESON; COLLINS; SEM, 2009).

[1] Linguagem de programação, interpretada, orientada a objetos de tipagem dinâmica e open source.

[2] É uma linguagem orientada a objetos pura, no sentido de que tudo é um objeto, incluindo números e funções. http://www.scala-lang.org/docu/files/ScalaTutorial-pt_BR.pdf.

[3] *Integrated Development Environment* ou Ambiente de desenvolvimento integrado (programa que reúne características e ferramentas de apoio ao desenvolvimento de software).

5 SISTEMA ANDROID

5.1 Histórico

Android é uma plataforma *open source* direcionada a dispositivos móveis que inclui um sistema operacional (SO), *middleware*[4] e aplicativos chaves, sendo a primeira plataforma de aplicativos para celulares e *tablets* de código-fonte aberto, inicialmente desenvolvido pela *Android Inc*, em 2003. Esta empresa foi fundada em outubro de 2003, por *Andy Rubin e Rich Miner* em Palo Alto, na Califórnia, no USA.

Em agosto de 2005 a *Google Corp*. comprou a *Android Inc*. transformando-a em uma empresa subsidiária. Dentro da *Google*, Rubin continuou o desenvolvimento do sistema, este baseado no núcleo do sistema operacional *Linux*.

No dia 05 de novembro de 2007, foi criada a *Open Handset Alliance* um grupo com mais de 40 empresas com a meta de criar padrões de código-fonte abertos para dispositivos móveis. Neste mesmo dia foi apresentado o seu primeiro produto: um sistema operacional construído sobre o *Kernel* do *Linux* com a versão 2.6.

O *Android SDK* foi apresentado aos desenvolvedores no dia 12 de novembro de 2007. No dia 02 de janeiro de 2008 iniciou um campeonato de desenvolvimento de aplicativos para a Plataforma *Android* criada pela Google com a oferta de 5 milhões de dólares para os melhores aplicativos que durou até 17 de abril de 2008.

No dia 28 de Agosto de 2008, foi anunciado o *Android Market* para a distribuição de aplicativos (apps), *games* e utilitários para seus dispositivos licenciados. E no dia 23 de setembro de 2008 foi lançada a primeira versão comercial o *Android 1.0*. O *Android* possui ainda suporte para desenvolvimento de aplicações desenvolvidas em diversas linguagens do Java, tais como C e C++, através do *Android NDK* ou Kit de Desenvolvimento Nativo (ANDROID, 2015).

O *Google* liberou todo o código fonte sobre a licença "*Apache*". Mesmo o código fonte sendo aberto para poder usar a marca *Android*, é necessário que o

[4] *Middleware* é um programa de computador que faz a medição entre o software e demais aplicações.

dispositivo seja certificado de acordo com o documento de definição de compatibilidade (KORJENIOSKI, 2011). Depois da certificação do equipamento ele terá acesso as aplicações fechadas, que inclui o *Android Market* (ANDROID, 2015).

5.2 Definição

Android é composto por uma pilha de software para dispositivos móveis incluindo um sistema operacional, um *middleware* e aplicativos chave. As aplicações são escritas usando uma linguagem Java e executam sobre uma máquina virtual a *Dalvik*, customizada para dispositivos com restrições de recursos, pouca capacidade computacional, baixa capacidade de armazenamento e com baterias com baixo nível de energia (AQUINO, 2007).

5.3 Arquitetura da Plataforma *Android*

O *Android* tem uma Arquitetura baseada no *kernel* do *Linux*, versão 2.6. O sistema funciona como uma camada de abstração entre o *hardware* e o restante da pilha de *software* da plataforma, já possuindo os recursos necessários para a execução das aplicações, como gerenciamento de memória, gerenciamento de processos, pilhas de protocolos de rede, módulo de segurança e vários outros módulos do núcleo de infraestrutura (AQUINO, 2007). Também é facilitado o surgimento de melhorias aos *drivers* já existentes. A arquitetura do sistema operacional *Android* é dividida em camadas, onde cada parte é responsável por gerenciar os seus respectivos processos (LECHETA, 2010).

É demonstrado as principais camadas da arquitetura do sistema operacional *Android* na (FIGURA 1).

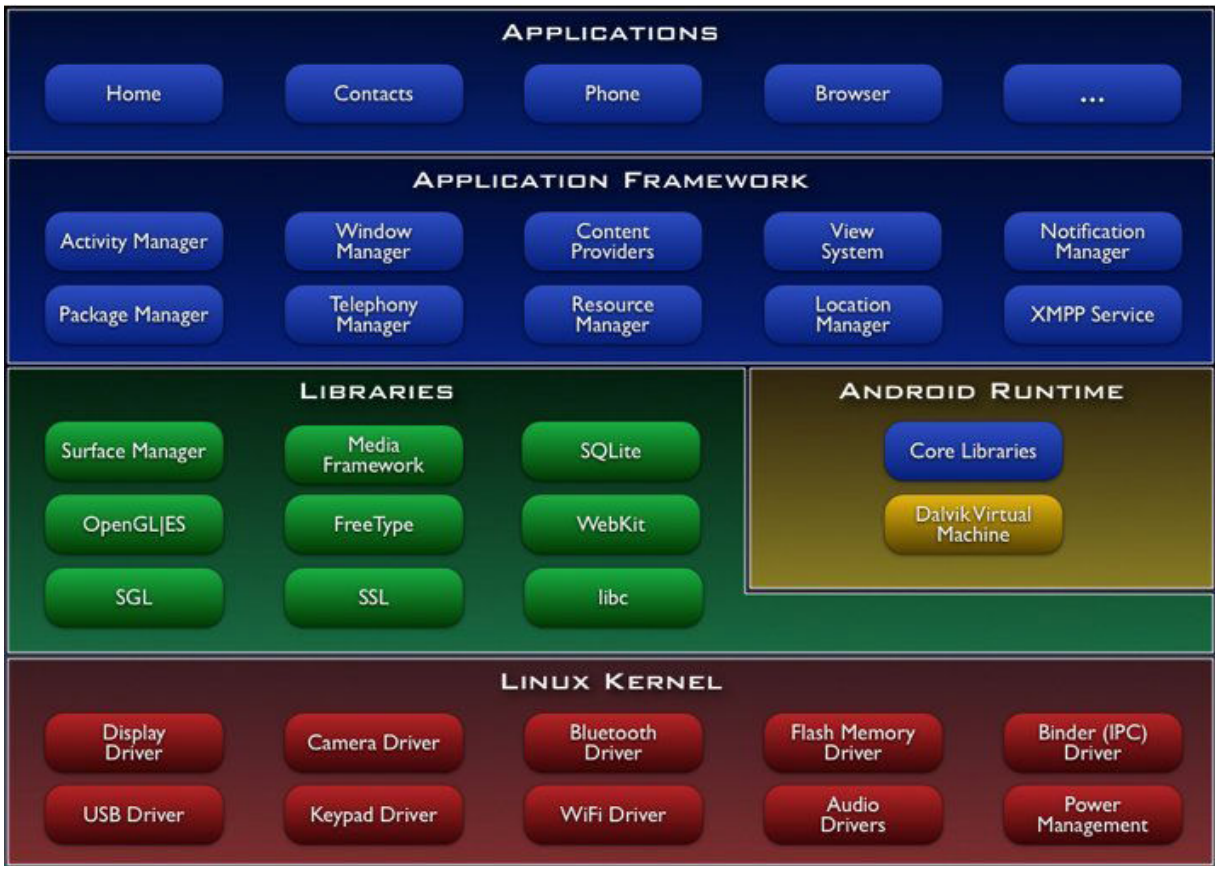

FIGURA 1 – CAMADA DA ARQUITETURA ANDROID

FONTE: GOOGLE IMAGE (2015).

A seguir, uma breve explicação sobre os módulos desta arquitetura dividida em cinco camadas sendo: Aplicações, Quadro de Aplicações, Bibliotecas, *Android* Tempo / Execução e o Núcleo *Linux* e apresentada.

5.3.1 Núcleo *Linux*

O *Android* tem uma arquitetura baseada no *kernel* do *Linux* com a versão 2.6, que funciona como uma camada de abstração entre o *hardware* e a pilha de *software* restante desta plataforma, possuindo vários recursos para a execução das aplicações, como o gerenciamento de processos, o gerenciamento de memória, as

pilhas de protocolos de rede, o modulo de segurança , vários módulos do núcleo de infraestrutura e serviços de sistema operacional (AQUINO, 2007).

5.3.2 Bibliotecas

Acima do *Kernel* estão as bibliotecas C/C++ utilizadas por diversos componentes do sistema, sendo uma implementação da biblioteca padrão do C (libc), com licença BSD e otimizada para dispositivos embarcados; bibliotecas para suporte a formatos de vídeos, áudios e imagens; um gerenciador que intermédia o acesso ao *display* e compõe as camadas de imagem 2D e 3D; o *engine* para navegadores *WebKit*[5]; bibliotecas para gráficos 2D (SGL) e 3D (*OpenGL ES*); um renderizador de fontes bitmap e vetoriais; e o banco de dados relacional *SQLite* (BORDIN, 2012).

5.3.3 *Android* Tempo / Execução

No *Android Runtime*, as aplicações escritas em Java são executadas em sua própria instância da VM (maquina virtual), mas as aplicações não são executadas em uma maquina virtual Java tradicional, sendo em uma outra chamada de *Dalvik*, que é uma maquina virtual baseada em registro otimizada para pouca memória e especialmente para dispositivos móveis, que por sua vez é executada em um processo separado dentro do sistema operacional, sendo isolada de futuras aplicações e facilitando o controle de recursos.

[5] *WebKit* é um motor de renderização utilizado em navegadores web para renderizar paginas.

5.3.4 Quadro de Aplicações

Nesta camada estão todos os componentes que permitem que novas estruturas sejam utilizadas para futuras aplicações, enfatizando a reutilização de código (DEVMEDIA, 2015). Na mesma é fornecida as funcionalidades necessárias para a construção de aplicativos, com a utilização das bibliotecas nativas, disponibilizando aos desenvolvedores as mesmas APIs utilizadas para o desenvolvimento de aplicações originais, permitindo que os programadores tenham os mesmos acessos ao sistema que os aplicativos da camada de aplicativos possuem. O *Application Framework* foi criado para abstrair a complexidade e simplificando a reutilização de procedimentos (GOMES, FERNANDES, FERREIRA, 2012).

5.3.5 Aplicações

Esta camada de aplicativos está localizada no topo da arquitetura do sistema operacional *Android*, composta por uma lista de aplicações padrões que incluem um cliente de e-mail, programa de SMS, calendário, mapas, navegador, gerenciador de contatos, e muitos outros que serão desenvolvidos pela comunidade, sendo todas estás aplicações escritas na linguagem Java (DEVMEDIA, 2015).

6 FERRAMENTAS UTILIZADAS PARA O DESENVOLVIMENTO DO JOGO

6.1 *DIA Diagram Editor*

O *DIA*, é um *software open-source*[6] licenciado sobre a licença GPL, um programa de criação de diagramas, voltado para diagramas de uso casual, podendo ser utilizado para desenhar diversos tipos diferentes de diagramas (GNOME.ORG, 2015). O software possui um design modular e atualmente conta com objetos especiais que auxiliam a desenhar modelos de diagramas de entidade relacional, diagramas UML, fluxogramas de rede, entre outros (GNOME.ORG, 2015).

A UML[7] é uma linguagem de modelagem, portanto, é somente parte do método para desenvolvimento de *software*. A UML é independente do processo, apesar de ser perfeitamente utilizada em processo orientado a casos de usos, centrado na arquitetura, iterativo e incremental (BOOCH, 2006).

Medeiros (2004), afirma que a UML não indica como se deve desenvolver um programa, apenas apresenta as formas que podem ser utilizadas para representá-lo, facilitando assim o seu desenvolvimento e entendimento.

Para a modelagem dos diagramas UML, foi escolhido o *Dia*, por apresentar facilidade na criação, por ser de código fonte aberto e por ter também uma versão *portable*, facilitando assim o seu uso.

O *Dia Diagram Editor* pode ser baixado para instalação no sistema operacional em: http://sourceforge.net/projects/dia-installer/files/dia-win32-installer/0.97.2/dia-setup-0.97.2-1.exe/download, e a para a instalação no *pendrive* pode ser baixado em: http://www.baixaki.com.br/download/dia-portable.htm.

---

[6] *Open-source* é um Código aberto, refere-se a software livre. Genericamente trata-se de software que respeita as quatros liberdades definidas pela Free Software Foundation.

[7] UML ou *Unified Modeling Language* é uma linguagem utilizada para modelagem dos dados, ou seja, com ela é possível desenhar através de diagramas as características e atribuições do sistema, permitindo uma visualização prévia do que será feito.

Abaixo podemos visualizar a área de trabalho do *Dia* (FIGURA 2).

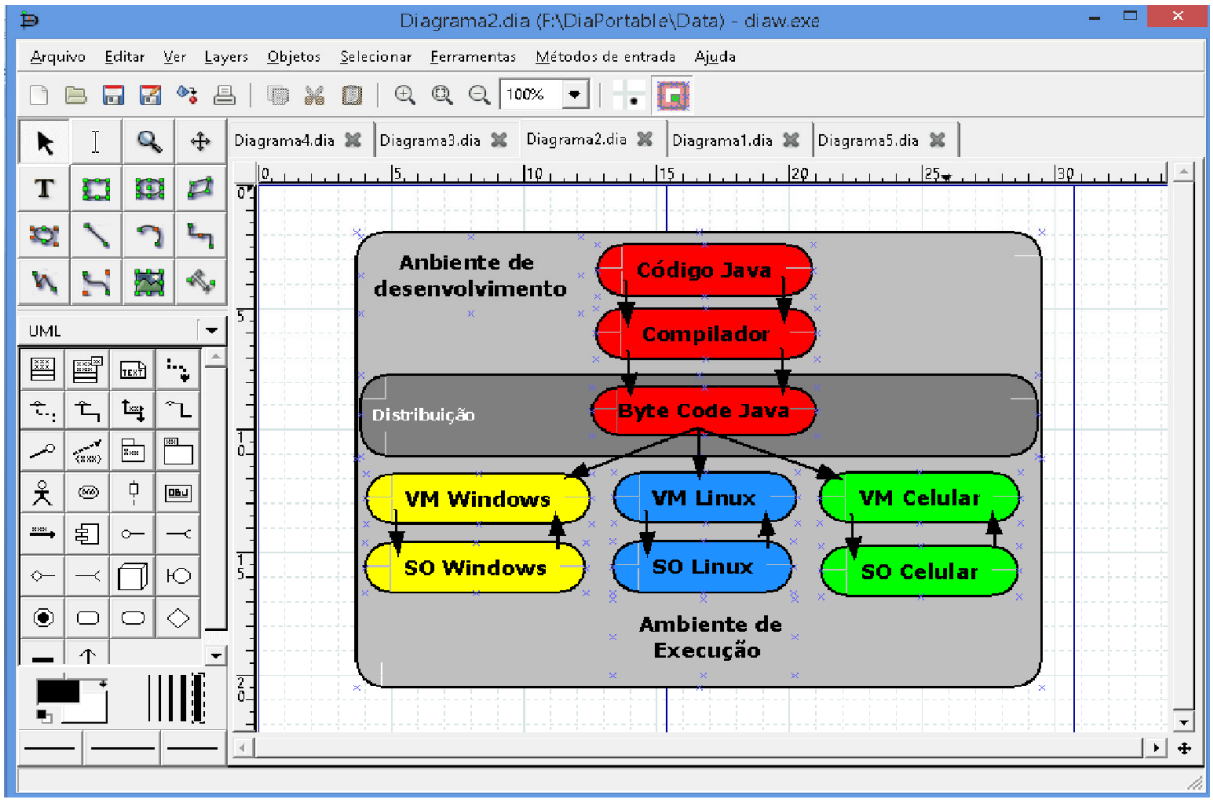

FIGURA 2- AREA DE TRABALHO DO DIA DIAGRAM

FONTE: O Autor (2015).

6.2 *Gimp*

Gimp (*GNU Image Manipulation Program*), é um *software open-source*, voltado principalmente para a criação e edição de imagens *raster*[8] ou *bitmap*, e em menor escala para desenho vetorial, podendo ser instalado direto no sistema operacional ou até em um *pendrive*, sendo a versão usada neste trabalho (TCC), foi a versão 2.8.14 (LINUX, 2015).

[8] Raster é o nome também dado ao bitmap, trata-se de imagens “pixelizadas”, ou seja, um conjuntos de pontos (pixéis) contido num quadro, cada um destes pontos possuindo um ou vários valores que descrevem a sua cor.

Para a manipulação de imagens foi escolhido o *Gimp* pela facilidade de uso, por utilizar vários formatos de arquivos, por poder instalar no *pendrive* podendo ser levado e utilizado em qualquer sistema operacional.

O *Gimp* pode ser baixado para instalação no sistema operacional em: http://www.baixaki.com.br/download/gimp.htm, e a para a instalação no *pendrive* pode ser baixado em: http://www.baixaki.com.br/download/gimp-portable.htm.

Abaixo pode-se visualizar a área de trabalho do *Gimp* (FIGURA 3).

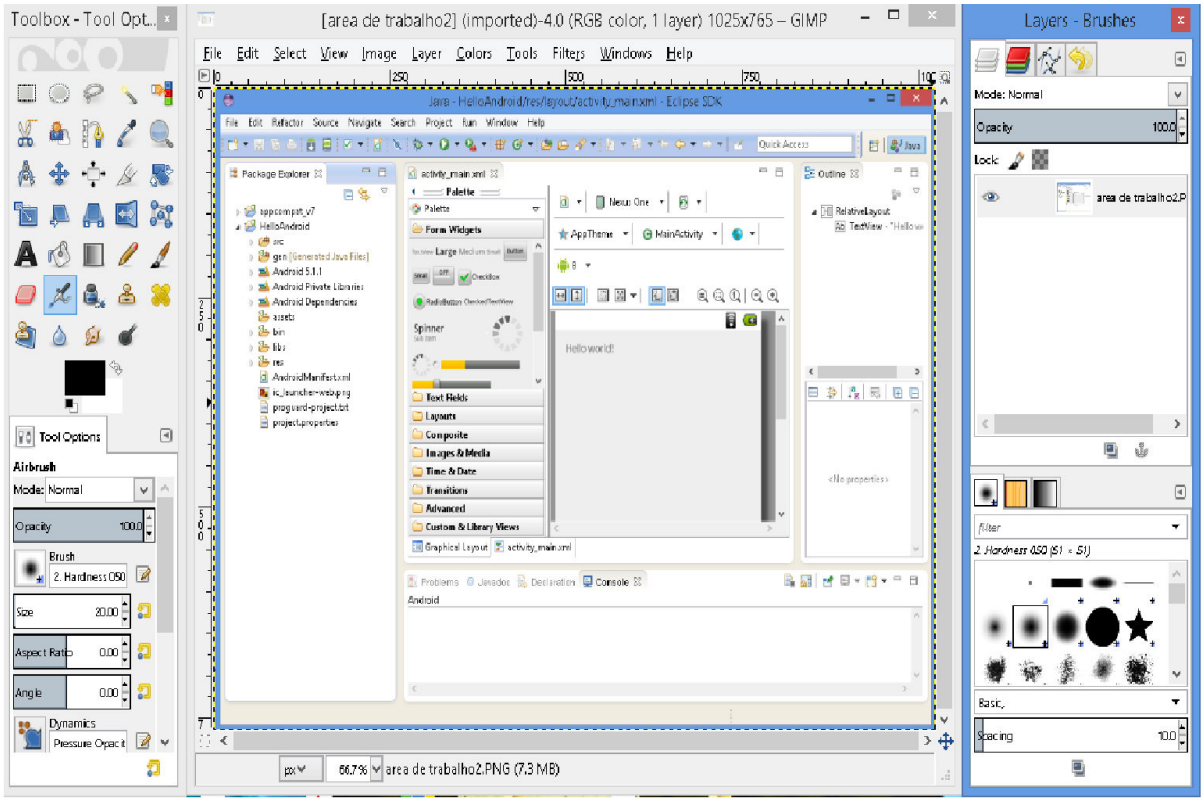

FIGURA 3 – AREA DE TRABALHO DO GIMP

FONTE: O Autor (2015).

6.3 *Eclipse*

Atualmente a *IDE Eclipse* é uma das ferramentas mais popular para o desenvolvimento de *software* em plataforma *Java*, possui facilidades na visualização dos arquivos contidos no projeto de forma clara e objetiva, ferramentas de gerenciamento de trabalho coletivo, compilação em tempo real, geração automática

de códigos, entre outras (LIMA, ET AL, 2015). O *Eclipse* permite que a área de trabalho seja personalizada conforme os critérios do desenvolvedor em relação ao projeto de desenvolvimento, podendo ser um projeto simples em *HTML*, ou até mesmo com o uso de aplicações em *EJBs*[9], diversos *frameworks* ou *J2ME* (SUN, 2015).

Como *open-source* há anos deixou de ser sinônimo de uma ferramenta sem recursos, com *bugs* e sem suporte algum, o *Eclipse* permite que se possa fazer em seu ambiente o mesmo que poderia ser feito com ferramentas pagas (LIMA, ET AL, 2015).

Atualmente, o responsável pela continuidade do desenvolvimento da ferramenta é o Consórcio *Eclipse*.org (ECLIPSE, 2015), criado pela IBM, empresa responsável pelo desenvolvimento da ferramenta em sua fase inicial e depois disponibilizada como projeto *open-source*. Hoje o consorcio é formado por grandes empresas de tecnologia de *software* e desde 2004, tornou-se independente sem a influencia direta da IBM (ECLIPSE, 2015).

A flexibilidade desta ferramenta é o que proporciona um diferencial ao desenvolvedor. Ele sempre trabalha em um *workbench*, isto é, um ambiente que pode ser configurado conforme suas necessidades com uso de perspectivas, além de diversas *views*[10] e editores (JOHAN, 2003). Com isso, pode-se adicionar recursos a um ambiente, criar novos ambientes de programação para outras linguagens ou para um propósito especifico.

Podemos visualizar a área de trabalho do *Eclipse* (FIGURA 4).

[9] EJB é uma arquitetura de componentes multi-plataforma para o desenvolvimento de aplicações Java, muiti-tier, distribuídas, escaláveis e orientadas a objetos.

[10] *Views* é uma visão ou vista de um agrupamento de informações.

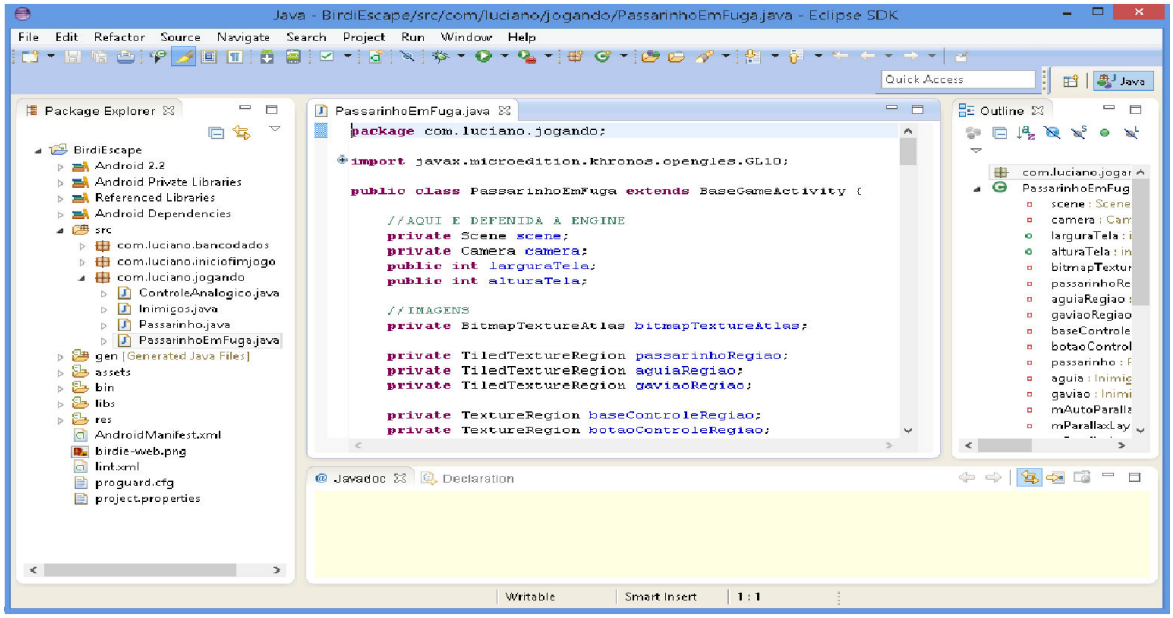

FIGURA 4 – AREA DE TRABALHO DO ECLIPSE

FONTE: O Autor (2015).

6.4 *ANDROID SDK*

O *Android SDK*, é uma ferramenta para o desenvolvimento e criação de aplicações, que apresenta suporte para mídias de áudio, vídeo e imagem, nos formatos GIF, JPG, PNG, MP3, MPEG4, AAC, AMR, H.264, bem como acelerador gráfico 3D, baseados no *OpenGL ES*[11] (MARTINS ET AL, 2011). O *Android SDK* permite que os desenvolvedores elaborem suas aplicações com a utilização de um dispositivo virtual (*Dalvik*) para os aparelhos de celular e *tablets*, desde utilitários a jogos que utilizem das funções oferecidas pelos aparelhos, como *touchscreen*, telefonia GSM, câmeras, bússola, GPS, acelerômetro, *Bluetooth*, *EDGE*, 3G e *wifi* (MARTINS ET AL, 2011). A plataforma traz um navegador integrado com base no código livre do motor *WebKit*, seus dados podem ser armazenados em um *SQLite* (MARTINS ET AL, 2011).

[11] OpenGL é uma multi-plataforma API gráfica que especifica uma interface de software padrão para 2D e 3D hardware de processamento gráfico. OpenGL ES é uma especificação OpenGL destinado a dispositivos embarcados.

Segundo Lecheta (2010), o *Android SDK* é um *software* utilizado para desenvolver aplicações no *Android*, que apresenta um emulador para simular o celular, ferramentas utilitárias e uma interface de programação de aplicativos (API) completa para a linguagem *Java*, com todas as classes necessárias para desenvolver as aplicações.

6.5 *JAVA JDK*

JDK[12] é um ambiente de desenvolvimento para a criação de aplicativos, *applets*, e componentes usando a linguagem de programação *Java* (ORACLE, 2015). Atualmente é uma das linguagens mais utilizadas e serve para qualquer tipo de aplicação, dentre elas: *desktop*, servidores, *mainframes*, aplicações móveis, jogos, *web* e muitas outras (DEVELOPER JUNIOR, 2015).

Java é multiplataforma, e usa uma máquina virtual (JVM). Os programas *Java* são copilados para um formato chamado *bytecode*[13], que é executado por qualquer sistema operacional, *software* ou dispositivo com um interpretador *Java* (LEMAY, 1999).

Para o desenvolvimento deste trabalho não poderia ter escolhido outra linguagem que não fosse a *Java*, pela sua compatibilidade, quantidade de bibliotecas disponíveis e por facilitar na programação com a integração da *IDE Eclipse* e com o *Android*, mas não foi utilizada a máquina virtual (JVM), pois o *Android* possui uma maquina virtual bem mais leve e com emuladores pré definidos conhecida como *Dalvik*.

Abaixo, podemos visualizar como acontece a compilação e execução de um programa em *Java* (FIGURA 5).

---

[12] JDK - Java Development Kit, em português: Kit de Desenvolvimento Java.

[13] *bytecode* são Codigos em bytes.

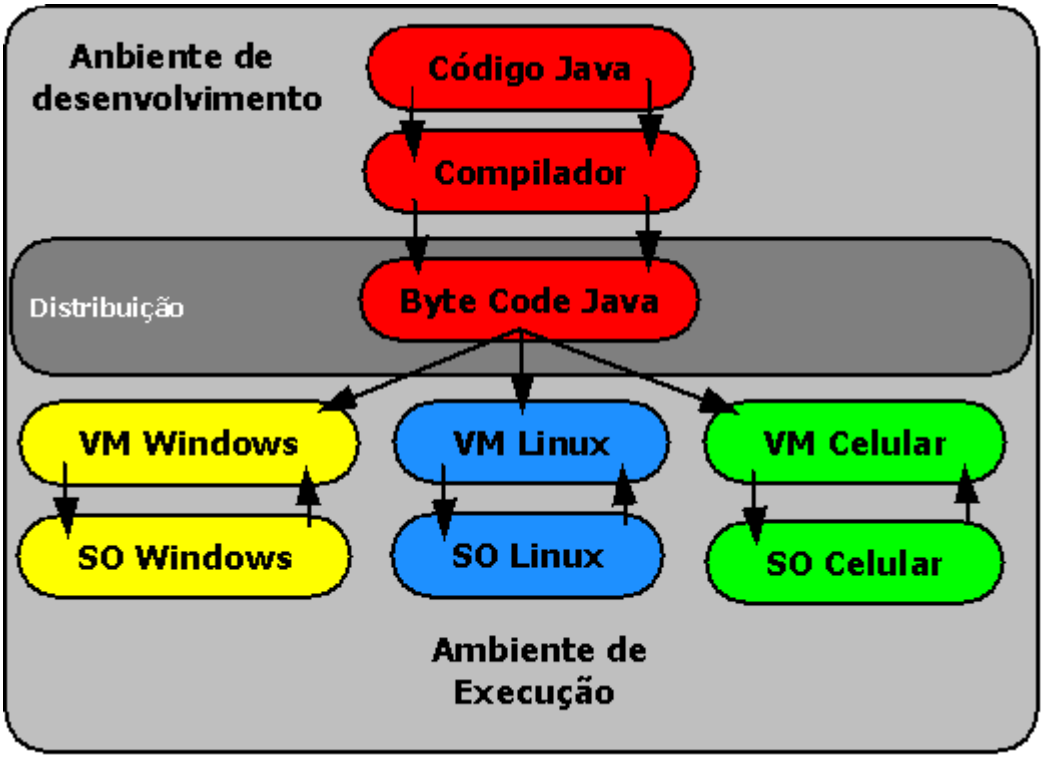

FIGURA 05 – COMPILAÇÃO E EXECUÇÃO EM JAVA

FONTE:GOOGLE IMAGE (2015) Modificado pelo autor (2015).

6.6 *SQLITE*

Segundo Vogell (2011), *SQLite* é um banco de dados *Open Source*, que é incorporado em *Android*. *SQLite* suporta recursos de banco de dados relacionais padrão, com sintaxe SQL, instruções e transações preparadas. Além disso, requer apenas um pouco de memória em tempo de execução. O mais importante é que o *SQLite* está disponível no sistema operacional *Android*, sem necessidade de utilizar um servidor ou até mesmo qualquer tipo de configuração.

Segundo Gonçalves (2011), na pratica, o *SQLite* funciona como um "mini-SGBD",[14] capaz de criar um arquivo em disco e ler e escrever diretamente sobre o arquivo. Sendo que o arquivo "criado possui a extensão ".db" e é capaz de manter diversas tabelas, permitindo ainda ao desenvolvedor criar e manipular seu Banco de

[14] Mini-SGBD é um mini sistema de gerenciamento de banco de dados.

Dados através dos comandos DDL e DML do SQL padrão. E, portanto, o “mini-SGBD” é utilizado para realizar a persistência de dados (SQLite, 2015).

6.7 *ADT Plugin*

O *ADT Plugin* é um *plugin* customizado para o *Eclipse*, que estende os recursos deixando configurar rapidamente novos projetos *Android*, criando uma interface de usuário do aplicativo, adiciona pacotes com base na *API Framework Android*, depura os aplicativos usando o *SDK* do *Android*, ferramentas e até mesmo exportar arquivos .apk assinados ou não assinados para a distribuição das aplicações (DEVELOPERS, 2015).

Foi desenvolvido para fornecer um ambiente integrado e poderoso para a construção de aplicações *Android*, sendo o jeito mais rápido para começar, e a sua utilização é a mais recomendada para desenvolver aplicações *Android* (MOTA, 2015).

6.8 *AndEngine (GLES2)*

A *AndEngine* é uma *Engine* que são kits de desenvolvimento de jogos, funcionam como o motor do jogo sendo responsável por todas as funcionalidades de baixo nível que o jogo precisa, é utilizado para aproveitar todas as características comuns em todos os jogos. A *AndEngine* veio para facilitar o desenvolvimento de jogos 2D, suas principais características são: sistemas de física e colisões (Box2D), sistema de partículas, suporte a Multiplayer, Auto escalamento para diferentes resoluções, suportes a gestos e *multitouch*, a realidade aumentada, a scripts, a *TileMaps* (TMX) e a gráficos vetoriais (SVG).

De acordo com STANG (2003) e BATTAIOLA ET AL (2004) os motores de jogos são softwares de desenvolvimento que fornecem diversos recursos que

permitem que o processo de implementação dos jogos seja feito com maior facilidade e rapidez.

7 JOGOS DIGITAIS

7.1 Histórico dos jogos

A história se inicia com os pioneiros e seus jogos simples sem nenhuma preocupação com a forma como esses jogos eram desenvolvidos; sendo o primeiro feito para funcionar em um osciloscópio, um instrumento eletrônico que permite observar sinais elétricos num tubo de raios catódicos, como a TV e as telas de computador (ARANHA, 2004).

O primeiro jogo da história surgiu em 1958. Segundo Amorin (2006), esse jogo foi criado pelo físico Willy Higinbotham e recebeu o nome de Tennis Programing, também conhecido como Tennis for Two. Ele era um jogo muito simples, jogado por meio de um osciloscópio e processado por um computador analógico.

Em 1961, no *Massachusetts Institute of Technology* (MIT), os pesquisadores criaram o jogo *Spacewar!.* Esse jogo foi programado em *Assembly* (nome da linguagem de programação de baixo nível) e executado em um computador DEC – PDP 1 (SOUZA e ROCHA, 2005).

Em 1966, o engenheiro Ralph Baer, criou uma máquina que era capaz de rodar os jogos por meio da TV, devido ao sucesso do jogo *Spacewar*!

Em 1967, após patentear o “chasing game”, um rudimentar jogo de Ping Pong, Ralph Baer foi reconhecido como o pai dos consoles;

Em 1971, foi desenvolvido por Nolan Bushnell, o primeiro jogo Arcade (fliperama);

Em 1972, Nasce o MAGNAVOX ODYSSEY 100, conhecido como o primeiro console caseiro. No mesmo ano surgiu a famosa ATARI;

Em 1976, inicia-se a nova geração dos videogames, após o surgimento dos consoles programáveis;

Em 1977, a empresa Warner Communications compra a ATARI que passa a ser conhecida como ATARI VCS;

Em 1979, Surge o primeiro portátil, o Milton Bradley Microvision;

Em 1982, é lançado o primeiro videogame vetorial, que vinha acompanhado de um monitor e também o surgimento da terceira geração dos consoles, com melhores gráficos;

Em 1984, houve uma queda nas vendas ocasionando o fato que ficou conhecido como “o crash de 1984”, levando várias empresas à falência, devido a enorme quantidade de jogos e consoles disponíveis no mercado;

Em 1985, com o lançamento do console NES e com uma política arrojada onde garantia a venda dos seus produtos a NINTENDO revolucionou o mercado;

Em 1988, a SEGA entra no mercado com o primeiro console 16 bits, o Mega Drive;

Em 1989, é lançado o GameBoy, o primeiro portátil da NINTENDO;

Em 1990, Surge o Super NINTENDO Entertainment System (SNES), com efeitos gráficos e sonoros mais evoluídos, sendo capaz de gerar imagens com até 256 cores;

Em 1993, com o primeiro console 64 bits, ATARI tenta reconquistar o mercado;

Em 1994, o famoso Playstation, que era projetado para manipular polígonos e ambientes virtuais 3D, é lançado pela SONY;

Em 1998, é lançado, o Dreamcast, sendo o último console da SEGA;

Em 2001, com uma arquitetura semelhante à de um PC, a Microsoft Corporation, desenvolve seu primeiro console o Xbox;

Em 2005, SONY(Playstation) e a Microsoft (Xbox), são as principais indústrias de console. Sendo que algumas das empresas que antes desenvolviam consoles continuaram no mercado de jogos, porém no ramo de desenvolvimento de jogos, como Atari e Sega;

Atualmente os jogos eletrônicos rendem milhões de dólares a seus desenvolvedores, sendo considerada uma excelente oportunidade de negócio (ABRAGAMES, 2004). E segundo Pettinga (2002), os jogos serviram/servem como forma de incentivo, onde as pessoas têm seus primeiros contatos com o computador.

7.2 Tipos de jogos

Os tipos de jogos são: aventura, ação, simulação, estratégia, esportivo, RPG e educativos.

Os jogos de aventura e de ação são caracterizados por apresentarem enigmas que surgem ao longo da exploração do mundo virtual. Nestes jogos, iniciam-se com um número mínimo de itens como armas, suprimentos, objetos, mas o objetivo é permitir que o jogador explore o cenário à procura de novos itens e a soluções de enigmas que possam levá-lo às próximas fases e, também, apresenta histórias que acompanham o jogo, como salvar uma princesa que foi raptada ou ir a procura de um tesouro perdido. O jogo é formado por cenários 2D ou 3D que geralmente o principal objetivo é representar um mundo virtual com personagens que fazem parte deste mundo imaginário (PEREIRA, 2006).

Os jogos de simulação representam e simulam situações, tarefas ou ações vivenciadas na realidade. São tentativas de recriar experiências do mundo real em um mundo virtual. As forças armadas dos Estados Unidos, desde 1950, utilizam simuladores como treinamento para o exército, podendo ser simuladas situações de guerra, de voo, de dirigir um tanque ou até mesmo operação de controle de uma base nuclear (PEREIRA, 2006).

Os jogos de estratégia requerem do jogador uma maior concentração, pensamento e planejamento, diferem-se dos demais por apresentarem as regras e os objetivos ao jogador, porém, para se cumprir os objetivos do jogo o jogador tem a liberdade de escolher qual a melhor estratégia a ser seguida, podendo ser jogado em tempo real e contra adversários reais, ou até mesmo contra o computador utilizando técnicas de inteligência artificial (IA) (PEREIRA, 2006).

Os jogos esportivos têm como objetivo simular uma partida de um esporte qualquer. Existem basicamente duas variações desse gênero: A primeira, o jogador tem como desafio administrar o esporte, por exemplo, administrar um time de

futebol, e o segundo, o jogador pertence a ação do esporte, por exemplo, controlar um jogador de futebol (PEREIRA, 2006).

Os jogos RPG (Role-Playing Games) tem como objetivo a cooperação e a criatividade do jogador, as quais são as principais características, sendo compostos por longas histórias e por grandes cenários. Nestes cenários há tesouros, objetos mágicos os quais devem ser procurados pelo jogador, aumentando o seu poder e buscando experiências para o personagem a fim de poder destruir seus inimigos que fazem parte do cenário. Este tipo de jogo é muito difundido pela internet devido a sua facilidade de ser jogado na opção multiplayer (jogar pela rede com diversos jogadores) (PEREIRA, 2006).

Os jogos educativos normalmente são voltados ao público infantil, sendo que ao jogar, atuam diretamente na aprendizagem do jogador. Segundo Dante e Barone (1998) “os jogos educativos computadorizados possuem, como uma de suas principais vantagens, um grande potencial para o processo de ensino e aprendizagem, por despertarem naturalmente o interesse dos alunos”. Estes tipos de jogos podem ser utilizados para as diversas disciplinas tais como: matemática, geografia, história, português, línguas estrangeiras, entre outras e, ainda são capazes de estimular o raciocínio humano. Segundo Bongiolo ET AL (1998) os jogos educativos devem expor os objetivos de maneira clara, bem como, propor regras ao invés de impô-las.

7.3 Jogos para dispositivos móveis

Um dos primeiros jogos dentro de um dispositivo móvel (celular) foi a Cobrinha, a qual vinha embutida no NOKIA 6110 em 1997 (FERNANDES, 2007). Era um jogo simples onde a única tarefa era permitir que a cobrinha se movesse pegando partes e ficando cada vez mais longa. No lançamento do jogo, nenhum celular da Nokia era capaz de exibir cores em sua tela, permanecendo o cursor preto com um contraste de fundo na cor cinza ou o fundo preto com um cursor em verde. A linguagem de programação desta época limitava o que poderia ser oferecido em

termos de jogos, sendo que estes jogadores permaneceram nesse mundo sem cor até 2001, quando foi introduzida a primeira tela colorida pela Nokia, tornando os jogos de celulares mais interessantes (FERNANDES, 2007).

A partir do uso das telas colorida, começaram os investimentos em jogos mais elaborados com gráficos surpreendentes e com qualidades de jogabilidade incríveis, com a utilização de várias plataformas. A seguir apresentamos alguns deste jogos os quais foram considerados os melhores de 2014.

Este foi considerado o melhor jogo de corrida de 2014 – *Sonic Racing Transformed*, é um jogo de corridas, mas com algumas diferenças, onde os veículos são transformados em barcos e até em aviões em meio a corrida (COUTINHO, 2015).

Sendo considerado o melhor jogo de ação de 2014 – *République*, é um jogo de ação do gênero furtivo. Também é um jogo instigante e com a jogabilidade casada com um ótimo enredo, com gráficos incríveis, colocando o jogador em um regime político com experiências cientificas com pessoas (COUTINHO, 2015).

Atualmente, este foi considerado o melhor de aventura de 2014 – *Adventure Game: The Wolf Among Us*. É um jogo com uma história incrível com um visual que é possível rodar em celulares menos potentes (COUTINHO, 2015).

Considerado o melhor jogo de luta de 2014 – *Marvel Contest of Champions*, é um jogo com uma jogabilidade simplificada e com ótimos gráficos, feito especialmente para celulares, o jogador terá o comando de personagens como Wolverine, Capitão América e Homem de ferro, que irá lutar contra os outros heróis Marvel em um torneio galáctico (COUTINHO, 2015).

O melhor RPG de 2014 – *The Banner Saga*, é um jogo de estratégia com campanha na medida certa para dispositivos móveis e com uma apresentação caprichada com belas apresentações (COUTINHO, 2015).

A maior surpresa de 2014 – *Just Dance Now*, é um dos jogos mais surpreendentes por transformar o celular em um controle para o jogo e, além disso, não é preciso ter um Xbox com *kinect*, somente um computador, *tablet* ou *smart*TV

para rodar o jogo, ele conta também com um multiplayer para várias pessoas (COUTINHO, 2015).

Este é o melhor jogo casual de 2014 – *Crossy Road e Timberman*, é um jogo onde o jogador deverá chegar o mais longe possível desviando dos carros e se cuidando para que uma águia o capture e tudo isso através de uma mecânica simples e viciante (COUTINHO, 2015).

Considerado o melhor jogo de plataforma de 2014 – *Leo's Fortune*, é um jogo onde o jogador controla uma bolinha de pelos, sendo este um grande inventor, o jogo conta com narração e legendas em português, os destaques alem do *puzzles*, são os gráficos que impressionam qualquer um pela alta qualidade (COUTINHO, 2015).

O melhor jogo de futebol de 2014 – FIFA 15: *Ultimate Team*, é considerado o melhor jogo do gênero, embora não de total liberdade ao jogador para jogar, a novidade desta versão é o modo "*Manager*", mais aprimorado permitindo que o jogador não jogue uma partida completa, apenas gerencie seu time (COUTINHO, 2015).

Consiste no melhor jogo de estratégia de 2014 – XCOM: *Enemy Within*, é um jogo que coloca o jogador no meio de uma invasão alienígena, o qual terá que lidar com os seres que estão invadindo a terra, sua jogabilidade é incrível, acompanhada por gráficos de primeira qualidade (COUTINHO, 2015).

Este é considerado o melhor jogo exclusivo para a plataforma *Android* em 2014 – *Half – Life 2*, é um jogo de tiro em primeira pessoa, possui uma das melhores campanha da história e é o melhor jogo de tiro de todos os tempos (COUTINHO, 2015).

Atualmente o melhor jogo exclusivo para IOS em 2014 – Vainglory, é um jogo que traz para os celulares o gênero MOBA, sendo poucos heróis, partidas rápidas e intensas, que conta com gráficos incríveis (COUTINHO, 2015).

Uns dos melhores jogo exclusivo para *Windows Phone* em 2014 – *Age of Empires: Castle Siege*, o jogo é marcado pela sua experiência parecida com o *Clash*

of Clans, mas é focado muito mais nos elementos que fizeram o sucesso de *Age of Empires* (COUTINHO, 2015).

Considerado o melhor jogo em Java em 2014 – Homem Aranha: *Ultimate Power*, é um jogo gratuito para baixar no estilo "*running*", onde o homem aranha é controlado pelo jogador o qual terá que derrotar vários inimigos, este tipo de jogo fazem muito sucesso em países como América Latina e Rússia (COUTINHO, 2015).

O pior jogo de 2014 – *Hobbit*: Luta pela Terra Média, é um jogo árcade com premissa simples e que engana o jogador pelas *screenshoots*, parece um jogo épico com muitas horas de aventura, mas é resumida em lutas sem sentido e combates vazios (COUTINHO, 2015).

O jogo mais aguardado para 2015 – *Lord of Fallen*, é um jogo de port interativo via web, mas foi anunciado uma nova versão para *smartphones* e *tablets* (COUTINHO, 2015).

Maior decepção de 2014 – *Godfire Rise of Prometheus*, é um jogo gratuito para o sistema operacional Android, o jogo decepciona ao deslizar na mecânica dos combates com movimentos muito lentos e por repetir muito os inimigos, sendo de uma aventura curta (COUTINHO, 2015).

Melhor jogo do ano de 2014 – *Monument Valley,* é uma jogo feito exclusivamente para dispositivos moveis, tem uma jogabilidade simples e cativa até mesmo quem não curte jogos, este puzzle se consagra como uma das melhores experiências móbile (COUTINHO, 2015)**.**

8 DESENVOLVIMENTO DO JOGO 2D

8.1 Apresentação do jogo

Neste momento, é apresentado o desenvolvimento do jogo 2D para celulares, *tablets* que utilizem o sistema operacional *Android* com a versão mínima 2.2, sendo este denominado como *BirdiEscape* – Passarinho em Fuga, onde o jogador tem que preservar a vida do passarinho que é uma arara-azul auxiliando em seu voo, fazendo com que ela desvie de seus inimigos e, ao desviar de seus inimigos, gavião e águia, e os inimigos chegarem ao final da tela do jogo, o jogador soma pontos. Porém quando um dos inimigos encostar na arara-azul, o jogador é direcionado para uma tela de *GameOver* onde pode ser escolhido entre continuar ou sair do jogo, caso o jogador decidir continuar, é retornado a tela do jogo, mas com a pontuação zerada e o recorde continua com a pontuação existente, ou se o jogador decidir sair, a pontuação sempre voltará ao inicio (zero), enquanto o recorde continuará com o valor finalizado não retornando ao valor zero. Fazendo com que o jogador sempre tenha que superar seu próprio recorde anterior, o qual fica guardado no banco de dados do dispositivo.

BirdiEscape foi desenvolvido com a linguagem de programação *Java*, utilizando como banco de dados a linguagem *SQLite*, que consiste em uma biblioteca baseada na linguagem SQL, atuando como um sistema de gerenciamento de pequeno porte “mini-SGBD” que é capaz de criar um arquivo em disco, ler e escrever diretamente sobre este arquivo. BirdiEscape foi baseado em um jogo desenvolvido pelos alunos: Tonykley Lobo, Rafael Bispo, Tiago Leite, Alison Luis e Augusto Matheus do Instituto Federal da Bahia.

8.2 Iniciando um aplicativo de exemplo

A partir deste ponto iniciaremos o desenvolvimento do jogo em 2D para a plataforma *Android*. Mas, para entender como funciona o *Eclipse* estaremos começando com o desenvolvimento de um aplicativo muito conhecido como "*HELLO ANDROID*".

Agora abrimos o *software eclipse* que neste momento já tem que estar instalado e configurado para o *Android SDK*, com o *plugin ADT*, para podermos desenvolver o Aplicativo.

Apresentamos a área de trabalho do *eclipse*, neste caso estamos utilizando o *Eclipse Classic* 4.2.2, conhecido como *JUNO* (FIGURA 6).

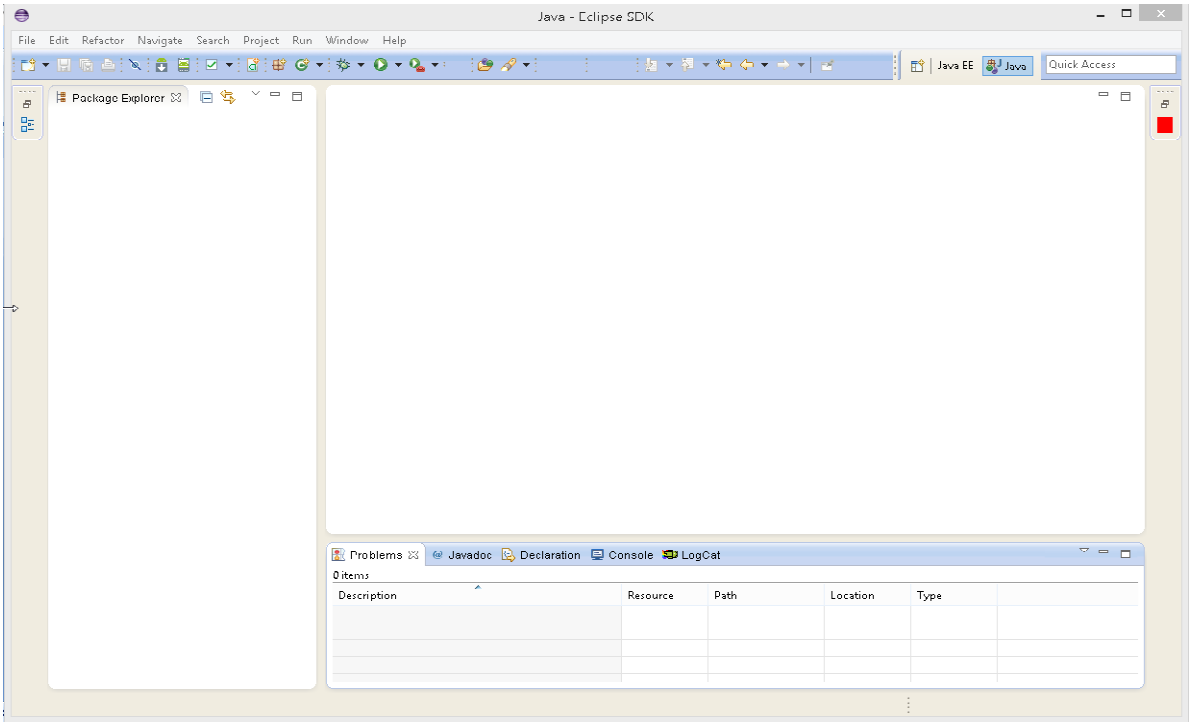

FIGURA 6 – AREA DE TRABALHO DO ECLIPSE CLASSIC 4.2.2

FONTE: O Autor (2015).

Iniciaremos o procedimento do aplicativo no *eclipse*, começando clicando em *File*, dentro de *File* passar o cursor em *New*, e dentro de *New* clicar em *Android*

Application Project, então abrirá uma nova janela com o nome de *New Android Application* (FIGURA 7).

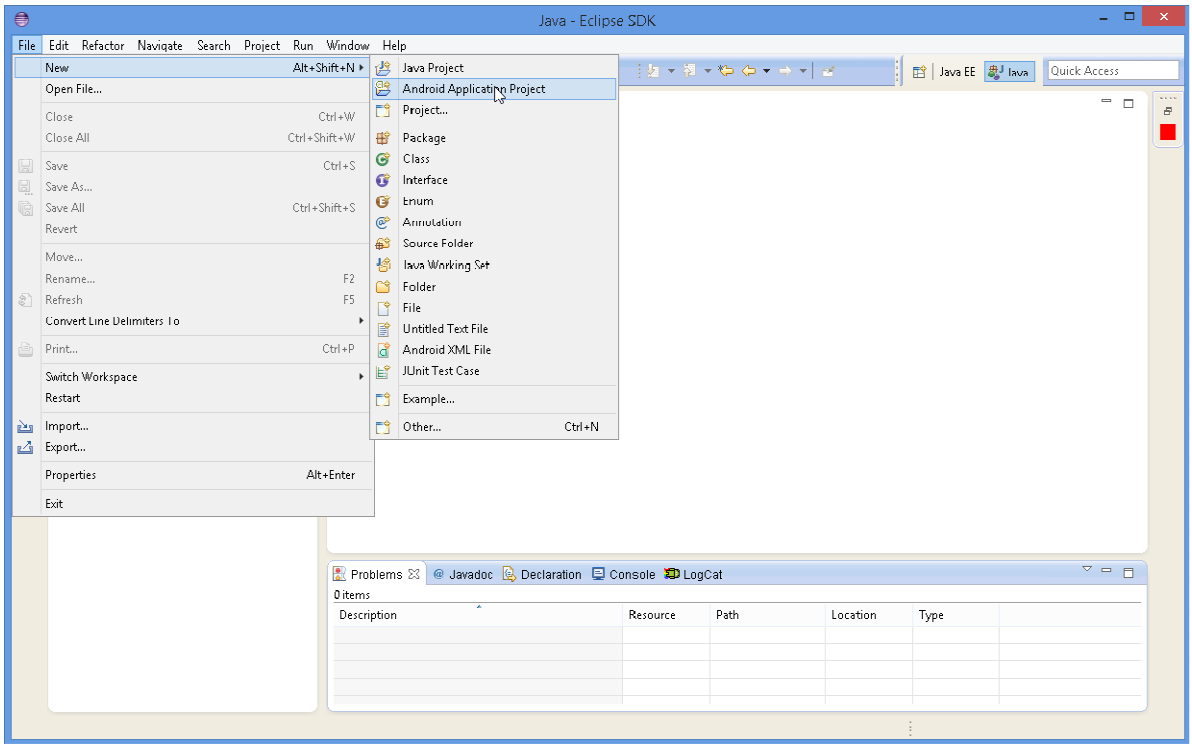

FIGURA 7 – INICIANDO UM NOVO PROJETO

FONTE: O Autor (2015).

Na janela “*New Android Application*” aberta logo abaixo mostra uma tabela onde são definidas algumas características do projeto. Os três primeiros campos referem-se à identificação da aplicação, sendo que, no campo “*Application Name*”, deve ser informado o nome que será conhecido o aplicativo que neste caso é o “*HelloAndroid*”, no campo “*Project Name*”, deve ser o nome do projeto que neste momento fica “*HelloAndorid*”, e no campo “*Package Name*”, deve ser informado o nome do pacote *Java* que neste caso poderá ser alterado ao decorrer do desenvolvimento, mas por enquanto fica como “ com.exemplo.*helloandroid*”.

Nos outros quatro campos, iniciando pelo campo “*Minimum Required SDK*”, deve ser escolhido a versão mínima do *Android* que o aplicativo exigirá para ser executado, no campo “*Target SDK*”, deve ser escolhida a versão a qual o aplicativo será projetado e testado. No campo “*Compile With*”, deve ser escolhida qual a versão do copilador será utilizada e no campo “*Theme*”, deve ser escolhido o tema

que deseja utilizar, alguns temas só funcionam em determinadas versões, sendo neste caso recomendo o *theme none* (FIGURA 8).

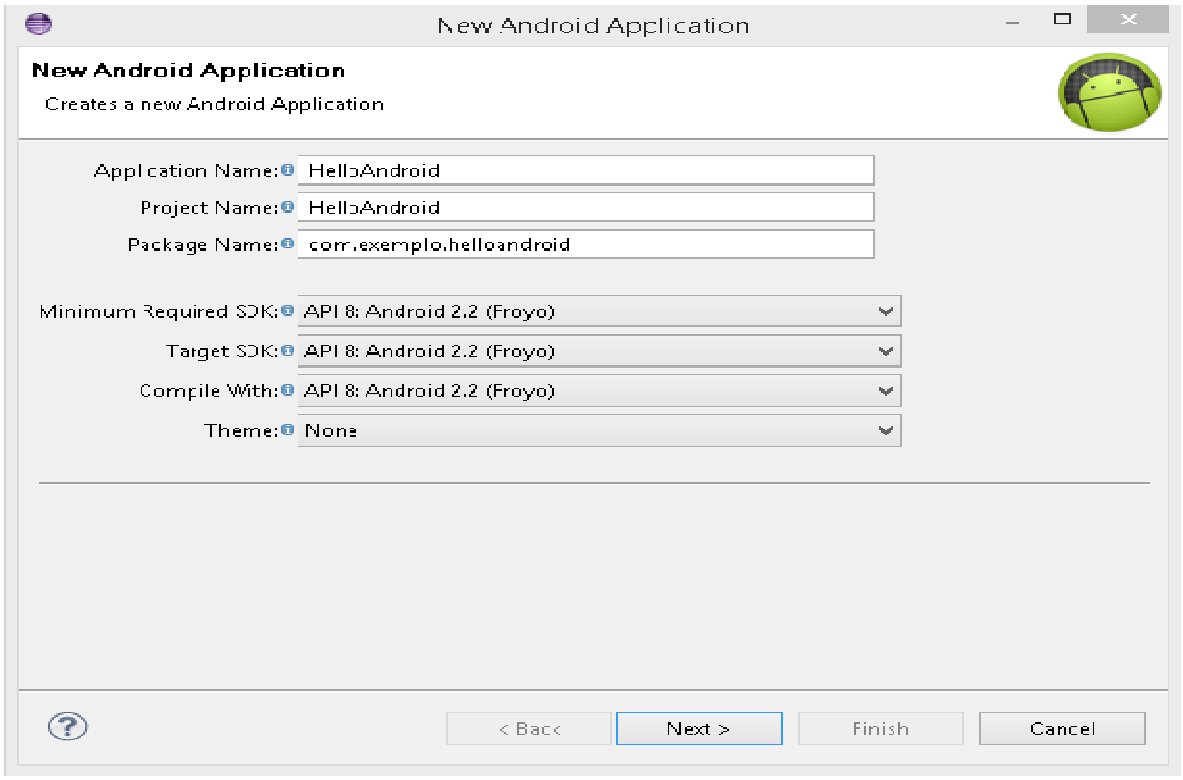

FIGURA 8 – DEFININDO VERSÃO DO PROJETO

FONTE: O Autor (2015)

Definimos as configurações do projeto, o *eclipse* já traz estas opções as quais são utilizadas e selecionadas, podendo ser alteradas, dependendo do desenvolvedor, mas convém alterar somente o local onde será armazenado o projeto, desmarcando o "*Create Project Workspace*" e, procurando um novo local, no botão "*Browser*" em "*Location*", caso desejar (FIGURA 9).

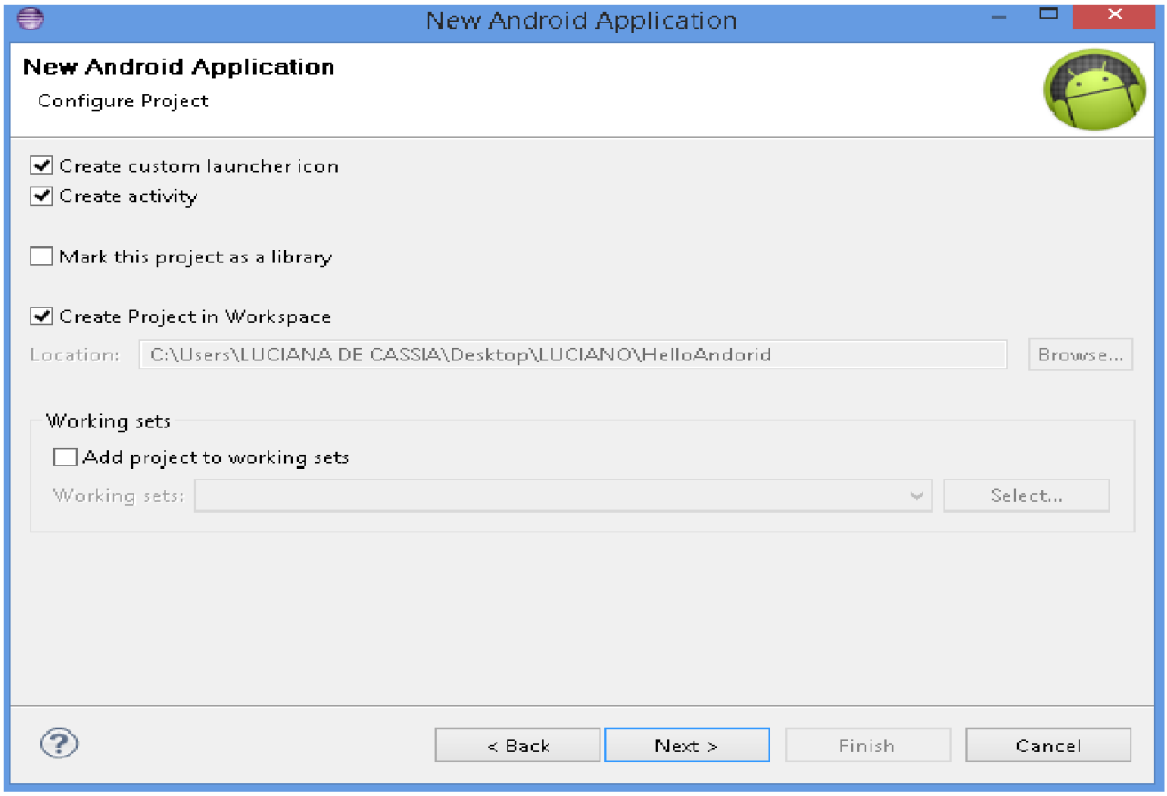

FIGURA 9 – CONFIGURANDO O PROJETO

FONTE: O Autor (2015).

É apresentado a tela de escolha de ícones do projeto, sendo possível alterar o ícone e também ter uma ideia do tamanho de cada resolução. Para escolher um novo ícone, clicar em "*Browser*", e abrirá uma janela para ser escolhida a imagem que quiser dentro do *browser,* mas se quiser continuar com o ícone padrão clique em *Next* (FIGURA 10).

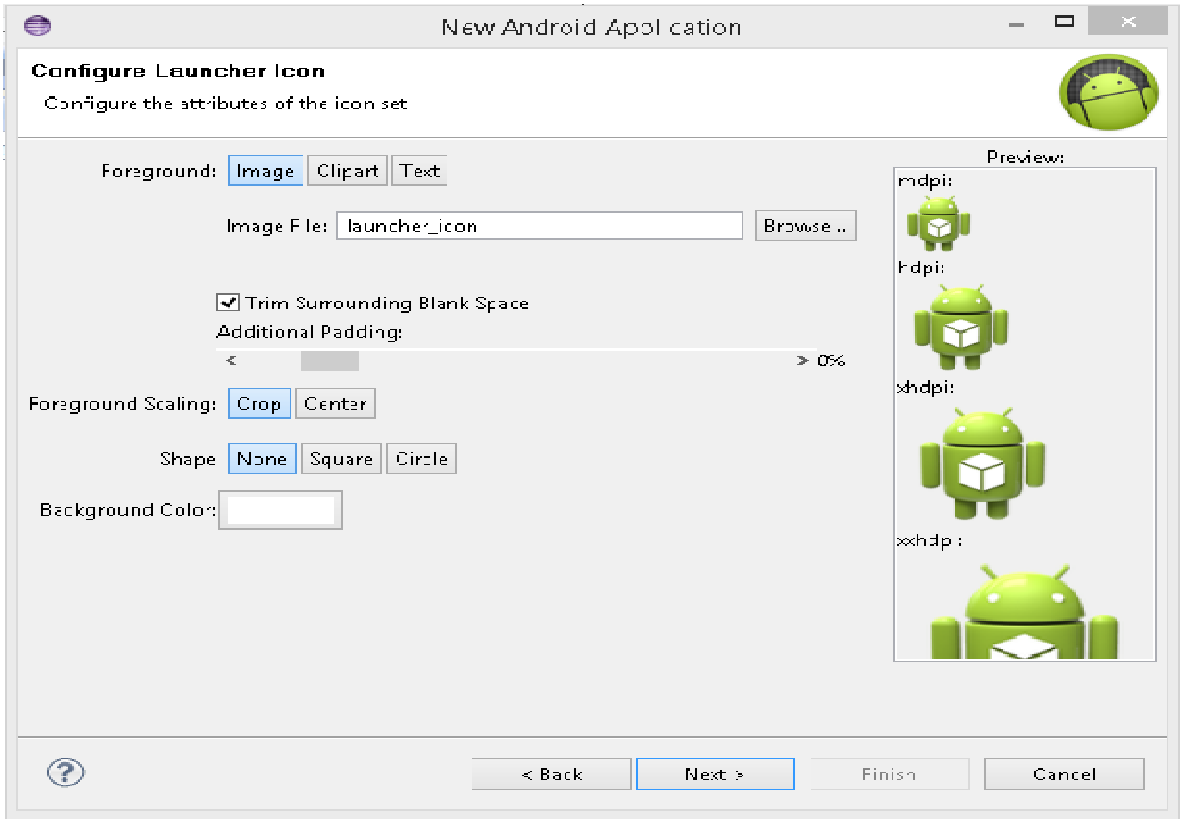

FIGURA 10 – CONFIGURANDO ICONES DA APLICAÇÃO

FONTE: O Autor (2015).

Nesta tela "*Create Activity*", estaremos escolhendo o tipo de tela do projeto, que neste caso vai ser a "*BLANK ACTIVITY*" e, também, pode ser escolhido o nome da *Activity* do projeto, sendo está *Activity* a classe responsável pela interação entre os componentes e o jogador. A activity esta relacionada às tarefas que uma aplicação poderá fazer, podendo ser uma tela inicial de um aplicativo, uma lista de itens, ou algo que possa ser apresentável para o usuário. Ela é a componente chave do *Android*, para que haja interação com a interface de usuário, uma classe deve herdar de uma *Activity*, mas neste caso deixaremos a "*CREAT ACTIVITY*", sem ser marcado, criaremos uma *Activity* para o projeto mais tarde, clique em *Finich* (FIGURA 11).

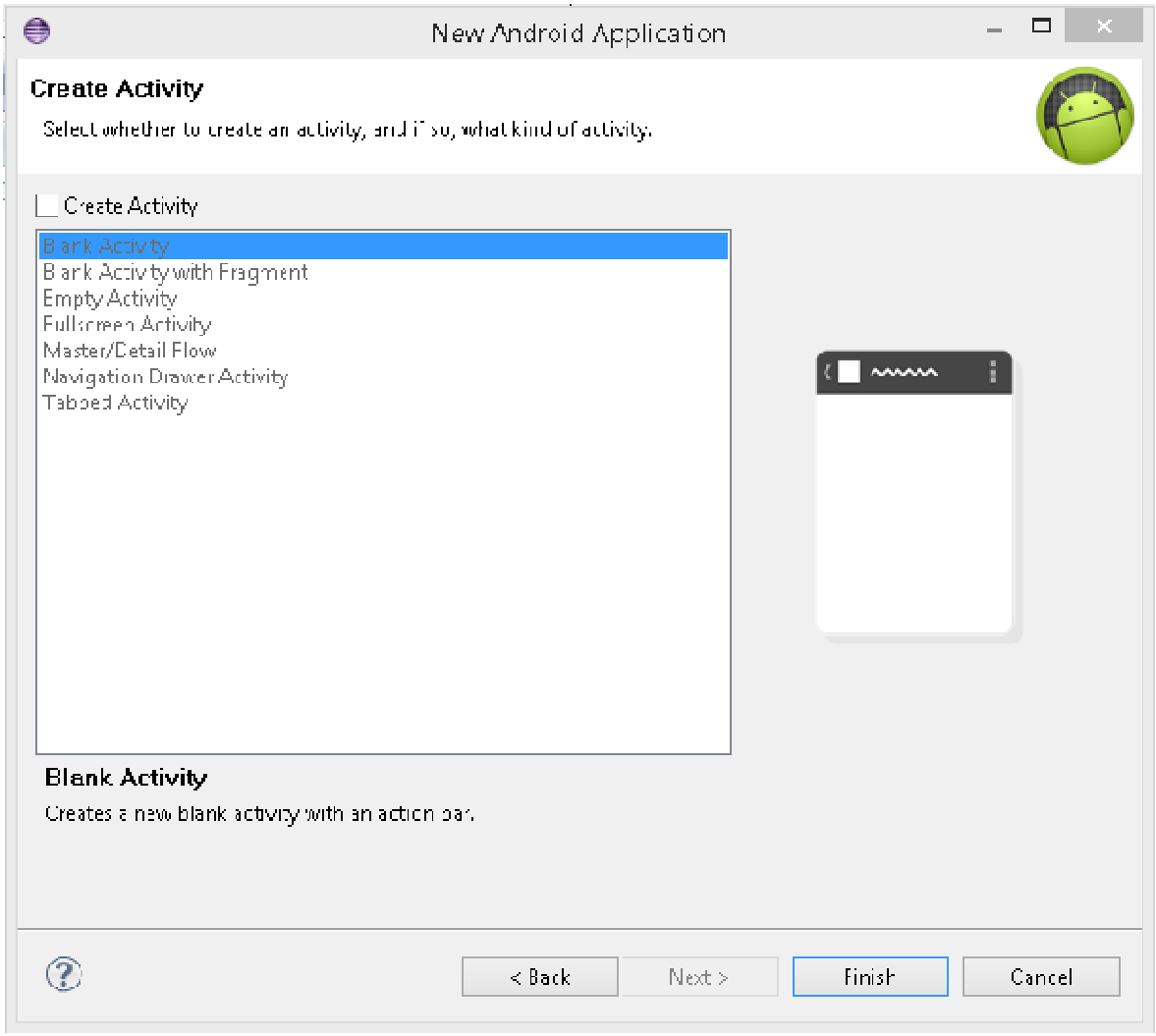

FIGURA 11 – TELA DE ESCOLHA DA ACTIVITY

FONTE: O Autor (2015).

A partir da estrutura do projeto, que foi criado, iremos construir o jogo "*BIRDIESCAPE*" (FIGURA 12).

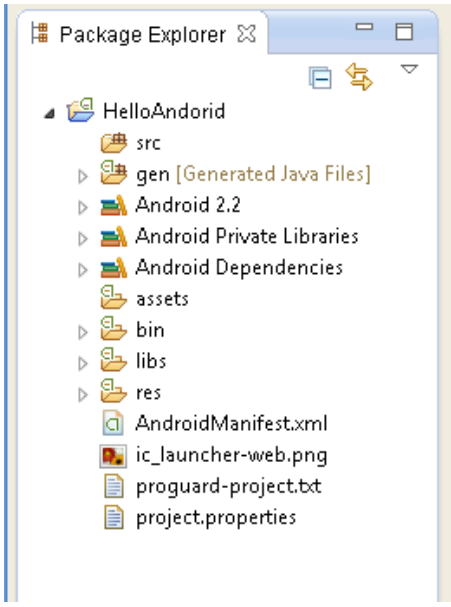

FIGURA 12 – PACKAGE EXPLORER

FONTE: O Autor (2015).

Na figura acima mostramos o *Package Explorer,* onde é encontrada a estrutura do projeto com suas pastas no *Eclipse*. Ele exibe todos os componentes da aplicação de amostra, onde estão os itens que serão usados e falaremos sobre estes itens.

A pasta "*SRC*", é onde se encontra os pacotes da aplicação e dentro destes pacotes são colocados as classes Java.

A pasta "*GEN*" é onde se encontra a classe *R.Java* que é gerada automaticamente e permite que a aplicação acesse qualquer recurso como arquivos e imagens utilizando as constantes desta classe.

A pasta "*ASSETS*" é onde estão os arquivos opcionais ao projeto, como a pasta "lós", que se encontram as imagens do aplicativo, a pasta "*raw*", que se encontram as musicas em mp3, que neste projeto não será usada.

A pasta "*BIN*" é onde estão os arquivos resultantes do seu projeto quando compilado.

A pasta "*LIBS*" é onde são adicionadas as bibliotecas adicionais do projeto como por exemplo o arquivo AndEngine. jar.

A pasta “*RES*” é onde estão os recursos da aplicação como as pastas “*drawables*” que estão os ícones da aplicação. Na pasta “*layout*” estão os arquivos “XML” de layouts para construção das telas da aplicação e a pasta “*values*” que estão os arquivos “XML” que são utilizados para outras configurações da aplicação.

Agora iremos transformar este exemplo de aplicativo no jogo “*BirdiEscape*”, com a utilização da “*AndEngine*” que é uma *engine* que vai facilitar a construção do jogo.

Para baixar esta “*AndEngine*” teremos que acessar a página do desenvolvedor desta *Engine* que é o Nicolas Gramlich no site *Github,* https://github.com/nicolasgramlich/AndEngine, e nesta página, procurar por “*GLES2*”, que também está marcado em vermelho para fazer download do arquivo zip (FIGURA 13).

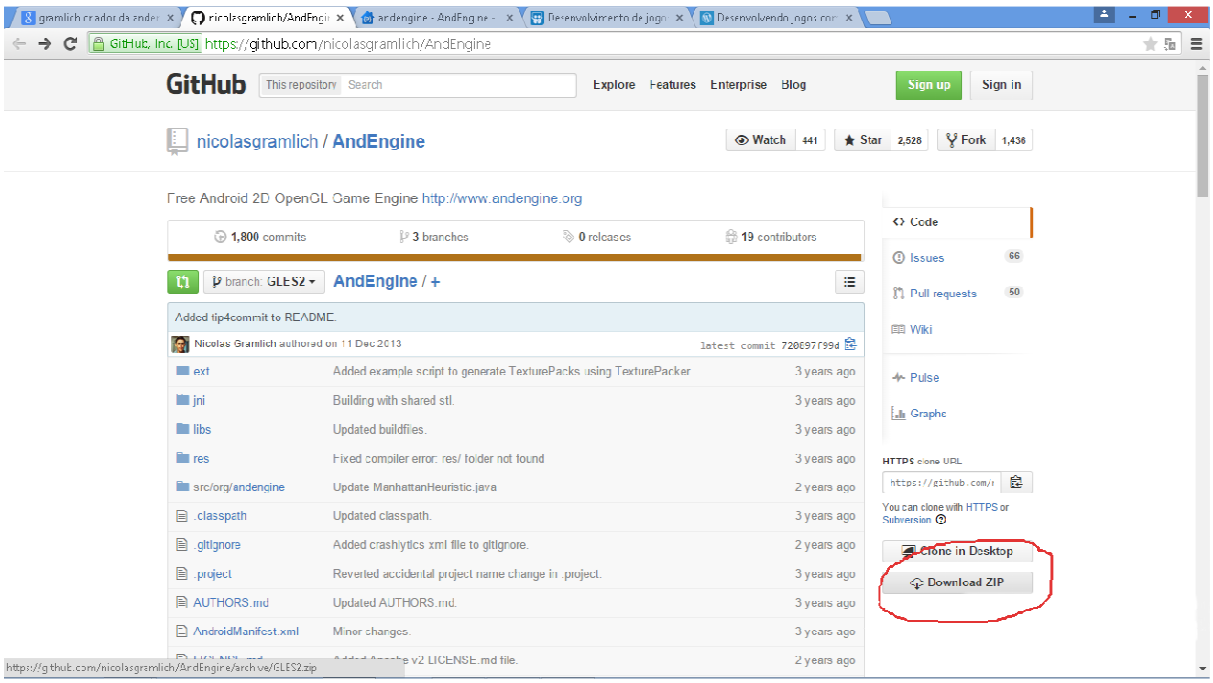

FIGURA 13 – SITE GITHUB

FONTE: GITHUB (2015) Modificada pelo autor (2015).

Após ter baixado o arquivo zip, deveremos importar este arquivo para o *eclipse*, pois o projeto criado se encontra como apresentamos logo abaixo (FIGURA 14).

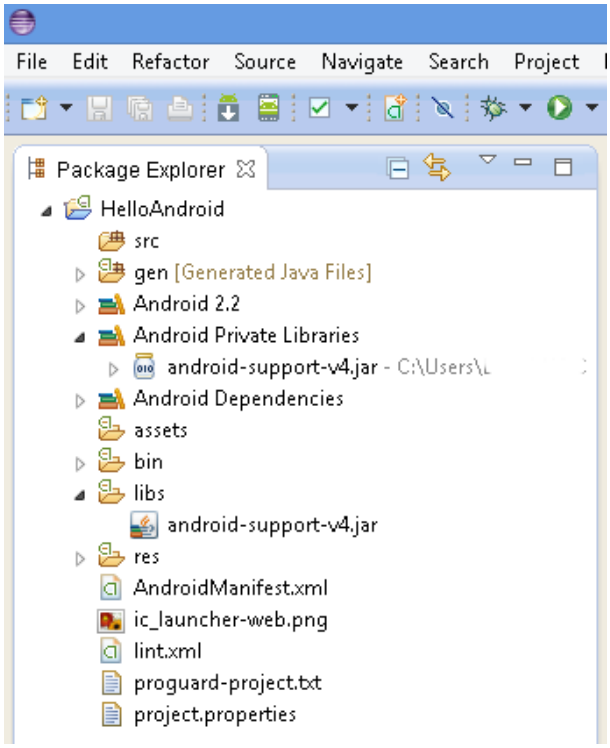

FIGURA 14 – EXTRUTURA DO PROJETO

FONTE: O Autor (2015).

Para importar o arquivo zip baixado é preciso extraí-lo em qualquer lugar que desejar, neste caso o arquivo está em uma pasta dentro do local C:/*eclipse SDK* 4.2.2/Aplicativos, está é uma pasta criada para os projetos, conforme mostramos logo abaixo (FIGURA 15).

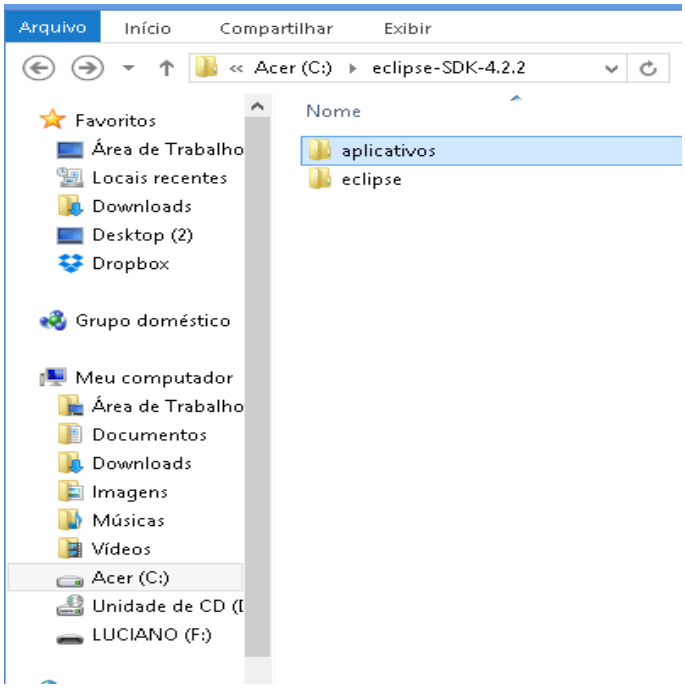

FIGURA 15 – LOCAL DO PROJETO

FONTE: O Autor (2015).

Iniciamos a importação, para isto basta clicar em *file* e *import,* e será aberto a tela de imortação (FIGURA 16).

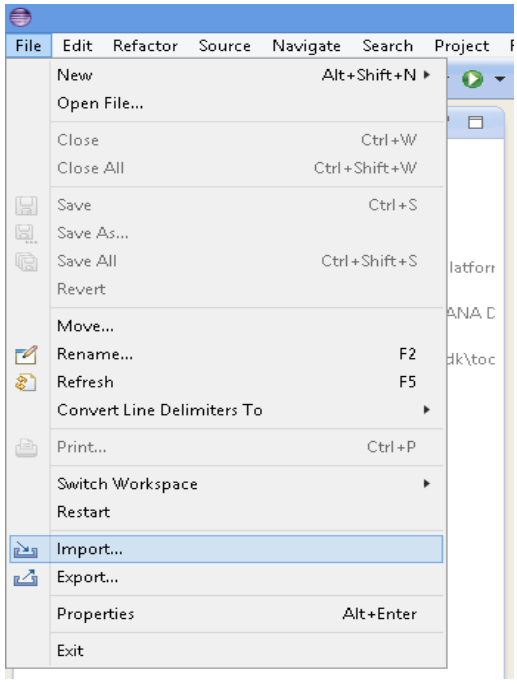

FIGURA 16 – INICIO DE IMPORTAÇÃO

FONTE: O Autor (2015).

Na tela de *import select*, para importar um projeto, selecione o modelo desejado que neste caso está na pasta "*Android*" que é o "*Existing Android Code Into Workspace*", clique em "*Next*" (FIGURA 17).

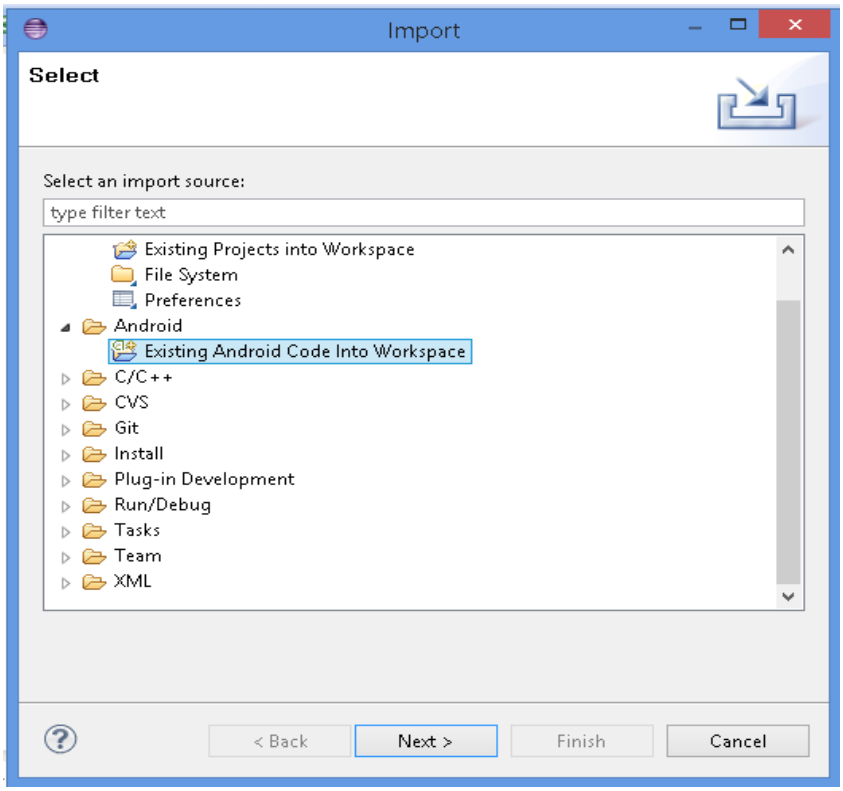

FIGURA 17 – TELA DE IMPORT SELECT

FONTE: O Autor (2015).

É apresentada a tela de *Import Projects*, nesta tela é escolhido o projeto para importação, clicando em "*Browse*r" abre uma outra tela (FIGURA 18).

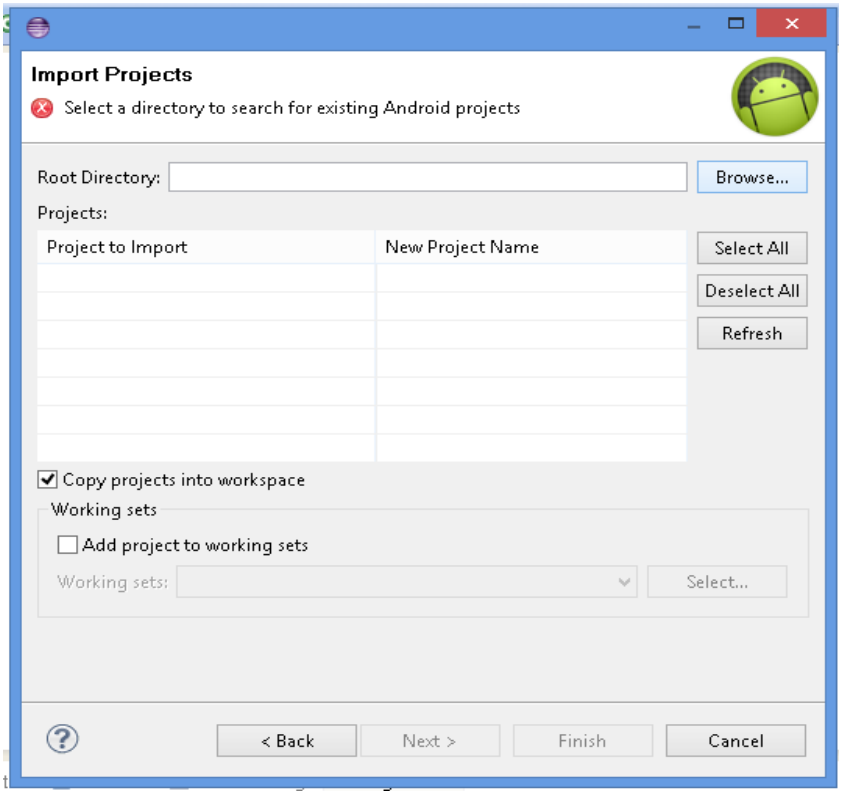

FIGURA 18 – TELA IMPORT PARA ESCOLHA DO DIRETÓRIO

FONTE: O Autor (2015).

Depois de clicar em *Browser*, abrirá a tela de “Procurar Pasta”, nesta tela é marcada a pasta desejada e então clicado em “*OK*” conforme é apresentado logo abaixo (FIGURA 19).

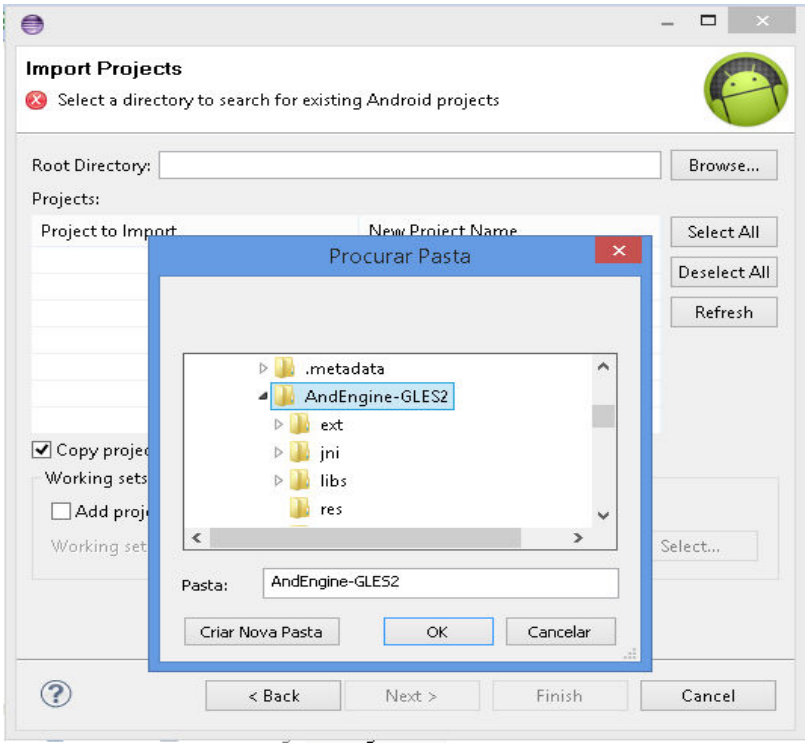

FIGURA 19 – TELA PROCURAR PASTA

FONTE: O Autor (2015).

Nessa tela "*Import Project*", já com todos os campos preenchidos, agora é só clicar em "*Finish*", que o projeto será importado (FIGURA 20).

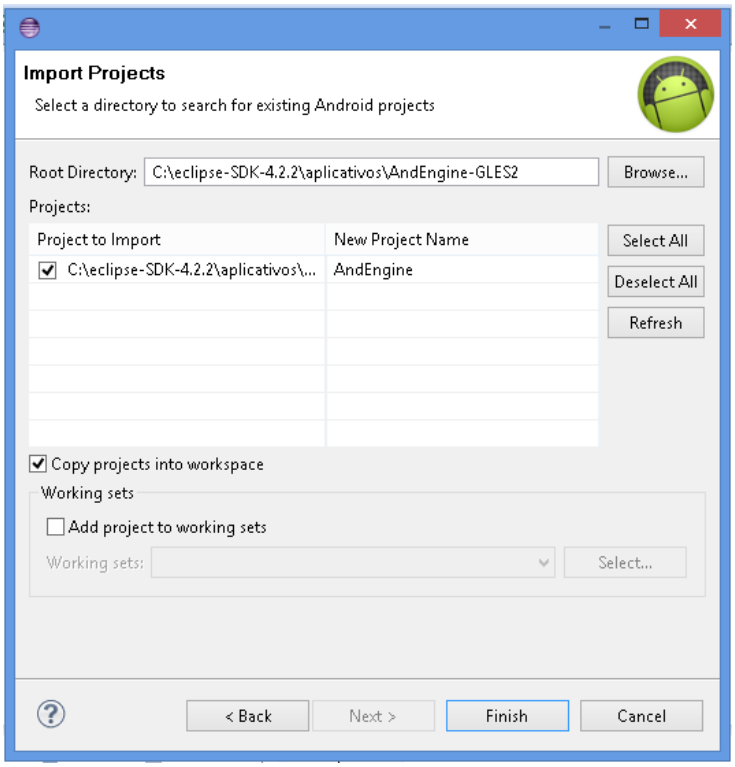

FIGURA 20 – TELA IMPORT PROJECTS

FONTE: O Autor (2015).

No “*Package Explorer*” com o projeto “*AndEngine*” já importado é necessário incluir as bibliotecas do *AndEngine* para nosso projeto (FIGURA 21).

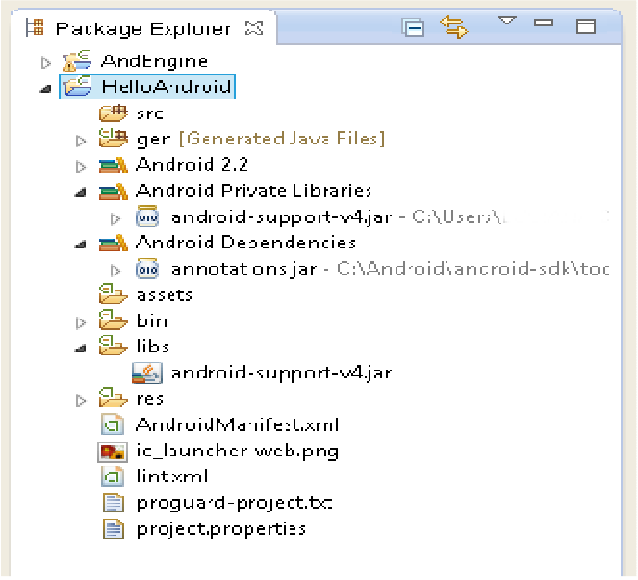

FIGURA 21 – PACKAGE EXPLORER

FONTE: O Autor (2015).

Agora iremos incluir o “*AndEngine*” no projeto, clique com o botão direito em cima do projeto e depois ir até “*Properties*” e clicar (FIGURA 22).

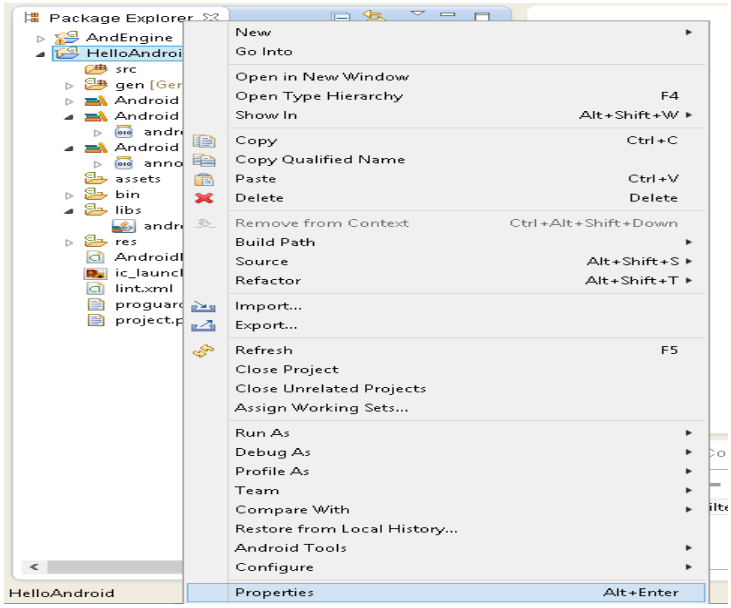

FIGURA 22 – PROPERTIES

FONTE: O Autor (2015).

Então, abrirá a tela “*Properties for HelloAndroid*” que é o projeto, então é clicado em “*Android*”, e é ativada a “*Is Library*” e clicado em “Add”, como mostra na (FIGURA 23).

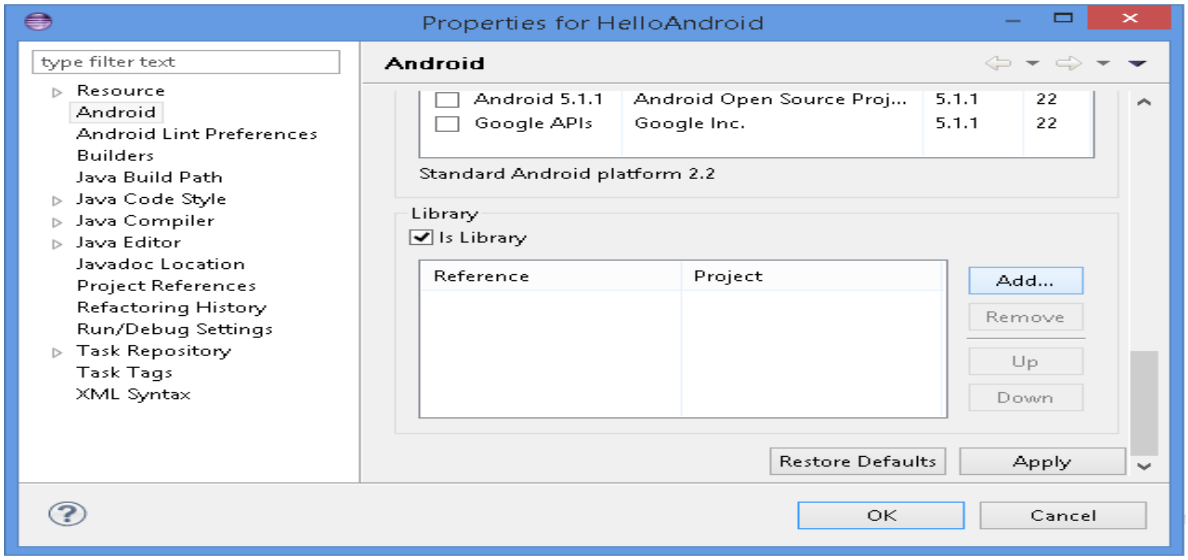

FIGURA 23 – TELA PROPERTIES FOR ANDROID

FONTE: O Autor (2015).

Então, é aberta a tela “*Project Selection*” onde é selecionada a “*AndEngine*” e clicado em “OK”, para que seja implementada a *library “AndEngine*” em nosso projeto (FIGURA 24).

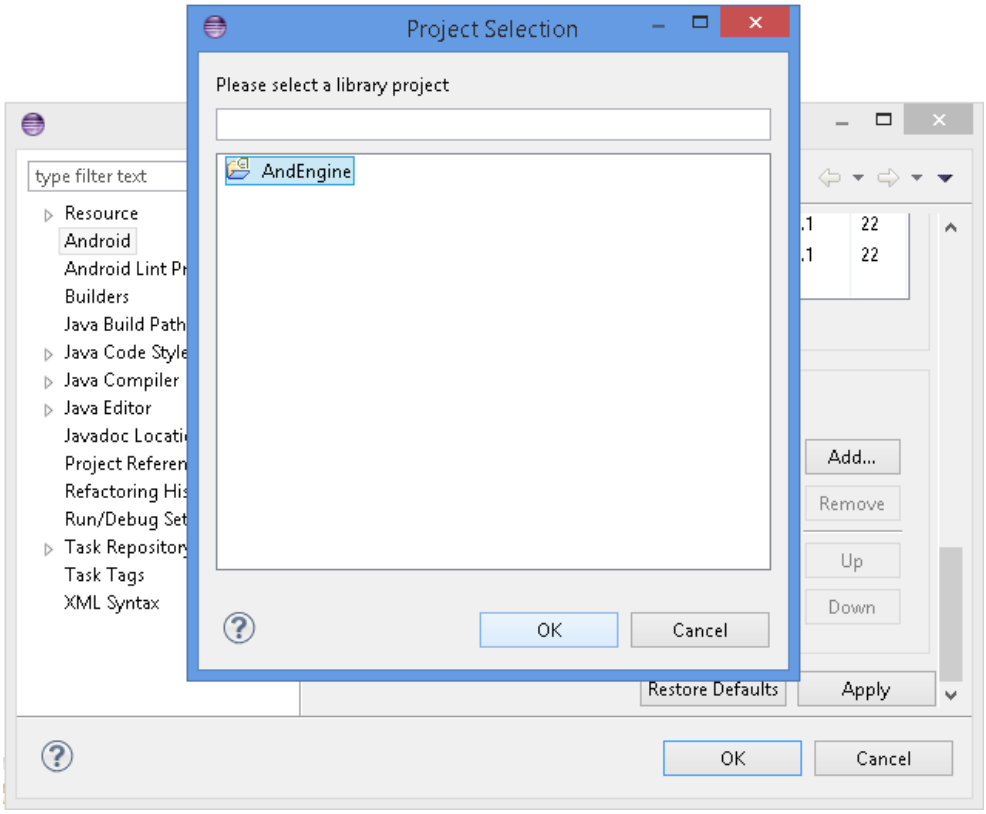

FIGURA 24 – TELA PROJECT SELECTION

FONTE: O Autor (2015).

Para finalizar teremos que clicar em “OK” na tela “*Properties for HellAndroid*”, com a referencia da “*AndEngine*” (FIGURA 25).

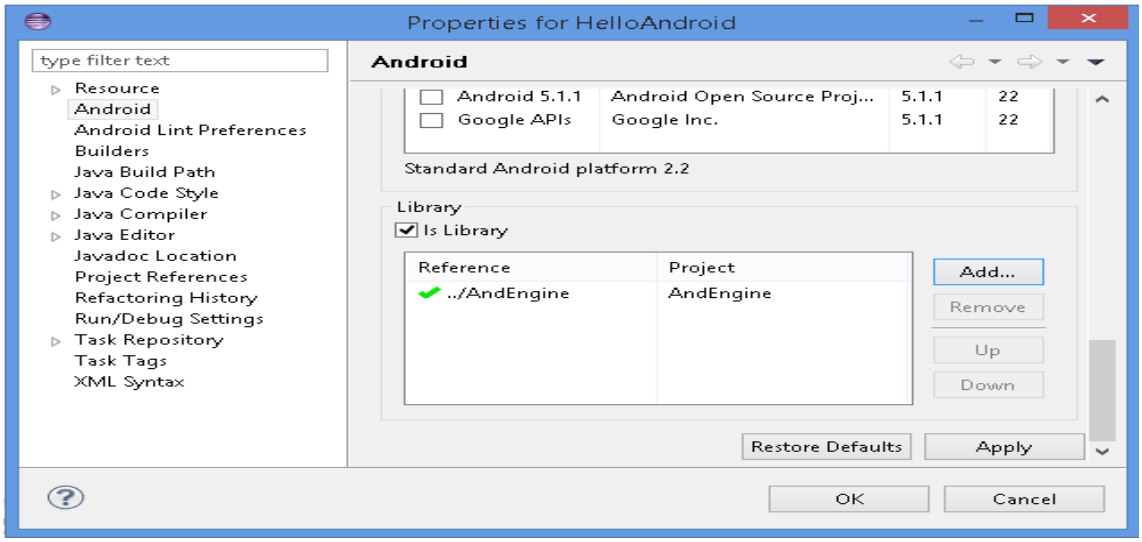

FIGURA 25 – PROPERTIES FOR HELLOANDROID

FONTE: O Autor (2015).

Nesta tela do "*Package Explorer*" podemos ver o projeto já implementado com a "*AndEngine*" (FIGURA 26).

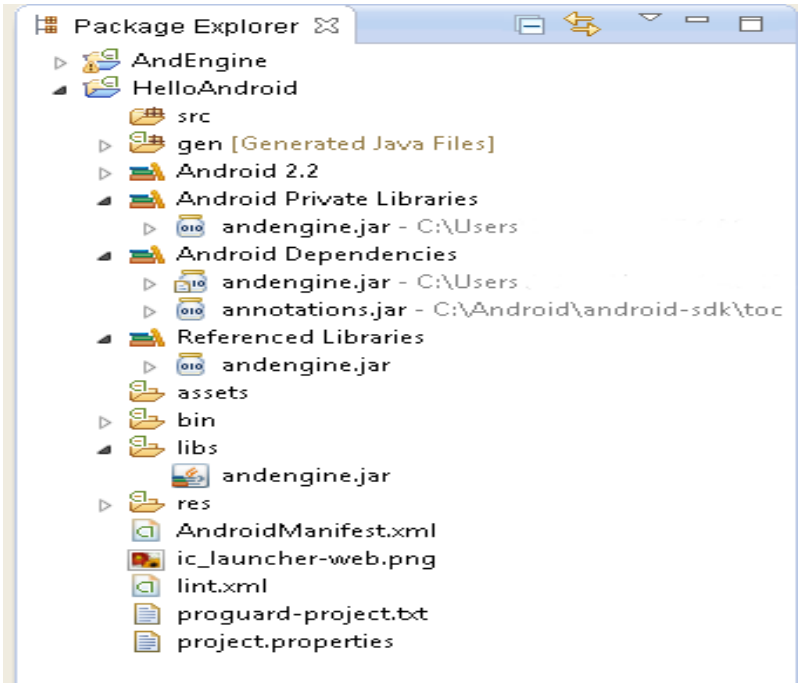

FIGURA 26 – TELA PACKAGE EXPLORER

FONTE: O Autor (2015).

A partir daqui é só alterar o nome do projeto para “*BirdiEscape*” ou outro nome que seja desejado. Para isso, é só clicar com o botão direito em cima do nome do projeto e ir em “*Refactor*” e clicar em “*Rename*”, como mostrado logo a baixo (FIGURA 27).

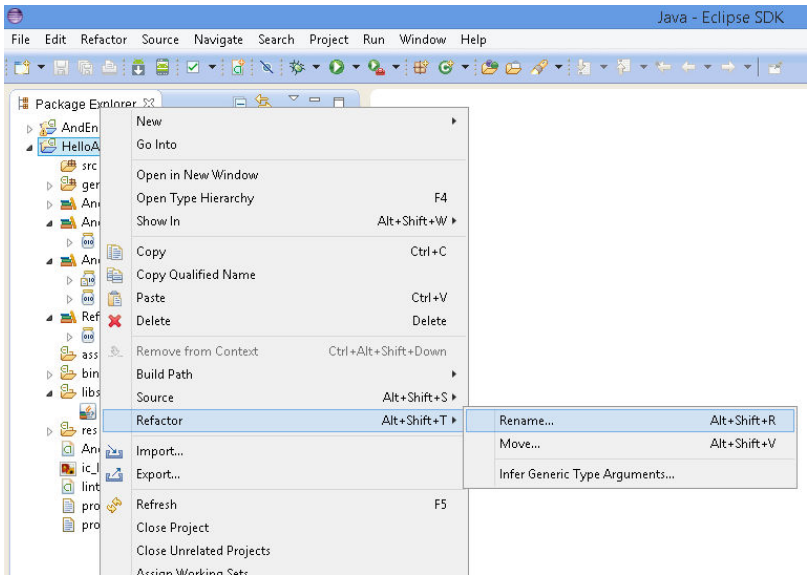

FIGURA 27 – RENOMEANDO O PROJETO

FONTE: O Autor (2015).

Na tela “*Rename Java Project*”, será preenchido com o novo nome do projeto e clicado em “*OK*” (FIGURA 28).

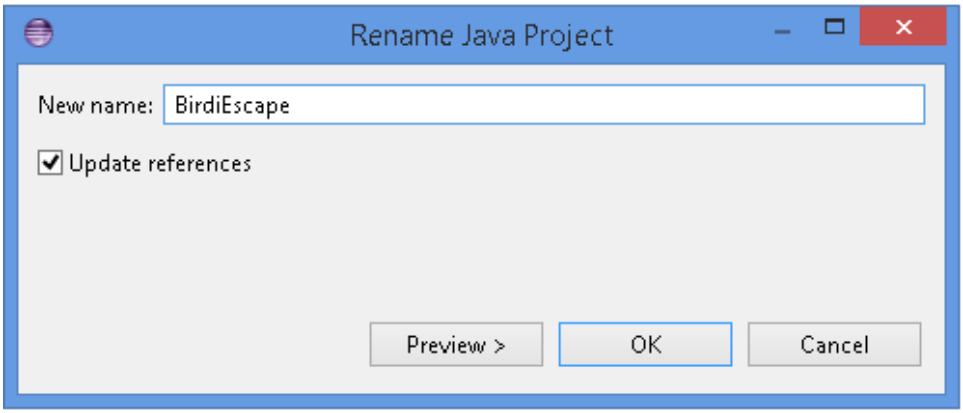

FIGURA 28 – TELA RENAME JAVA PROJECT

FONTE: O Autor (2015).

Para incluir os pacotes no projeto, clicar com o botão direito na pasta “*SRC*”, ir em “*New*” e clicar em “*Package*”, conforme mostrado logo a baixo (FIGURA 29).

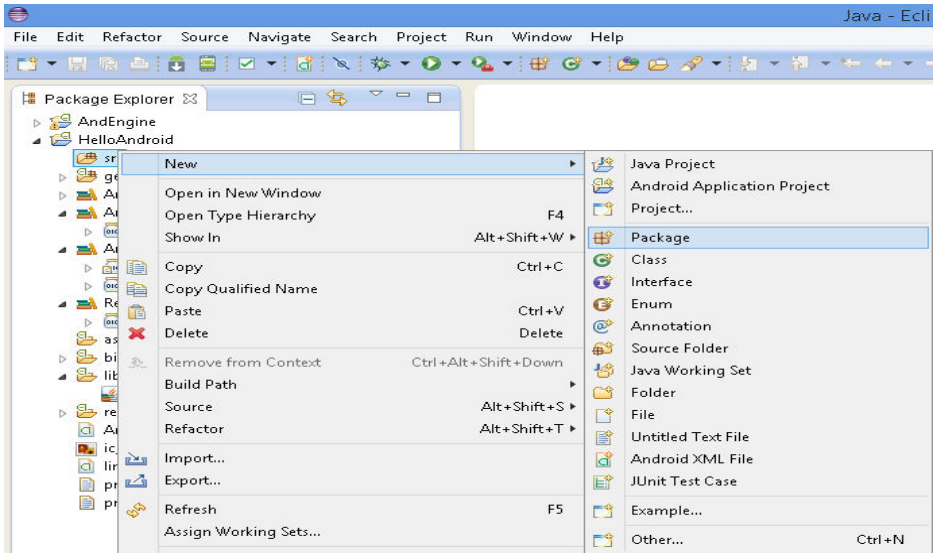

FIGURA 29 – CRIANDO PACOTES NO PROGETO

FONTE: O Autor (2015).

Nesta tela “*New Java Package*”, em “*Name*” preencheremos com o nome do novo pacote e clicado em “*Finish*”, os pacotes do jogo são: com.luciano.bancodados, com.luciano.iniciofimjogo e com.luciano.jogando. Contudo, estes pacotes como os demais arquivos do projeto podem ser alterados, neste caso será alterado para com.exemplo.bancodados (FIGURA 30).

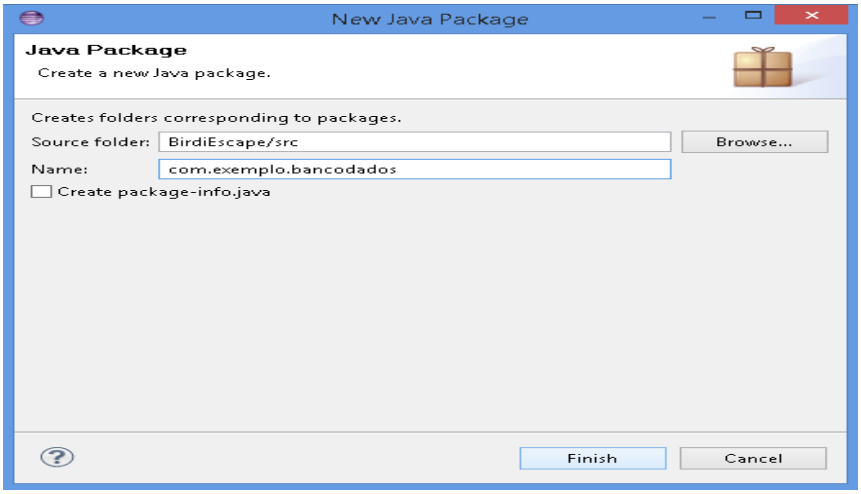

FIGURA 30 – NOVO PACOTE JAVA

FONTE: O Autor (2015).

Para incluir as classes no projeto, abra a pasta “*SRC*” e dentro desta pasta e clique com o botão direito no pacote criado anteriormente. Em “*New*” e clique em “*Class*”, conforme mostrado logo a baixo (FIGURA 31).

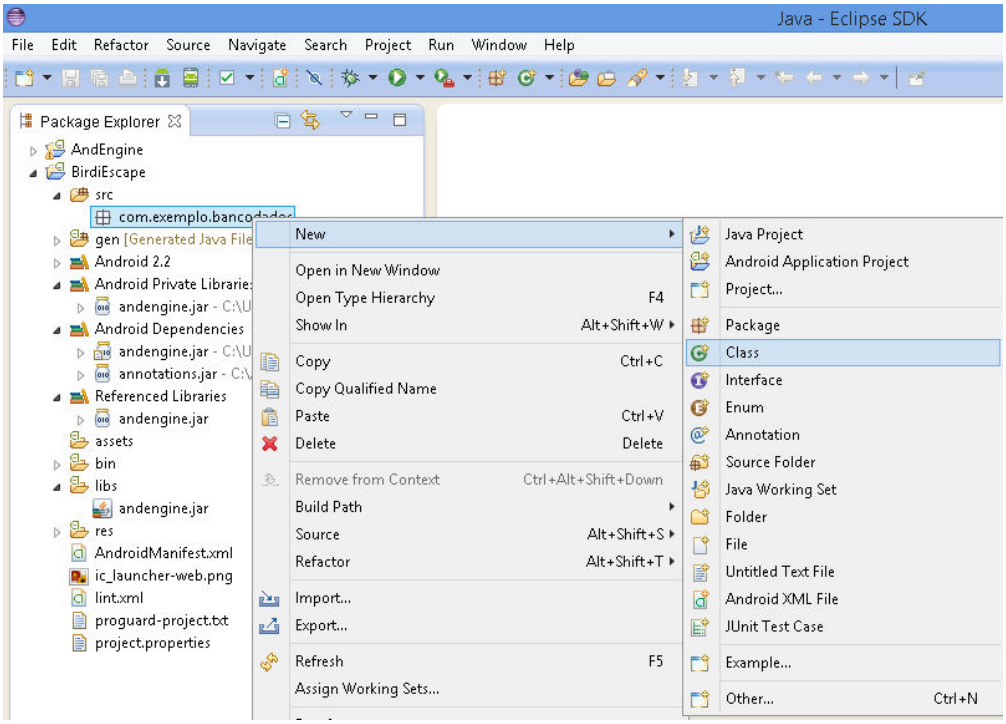

FIGURA 31 – INCLUINDO UMA CLASSE NO PACOTE

FONTE: O Autor (2015).

A tela “*New Java Class*”, em “*Name*” é preenchida com o nome da classe, em “*Modifiers*” será escolhido como *public*, clicar em “*Finish*”, incluir as outras classes do jogo (FIGURA 32).

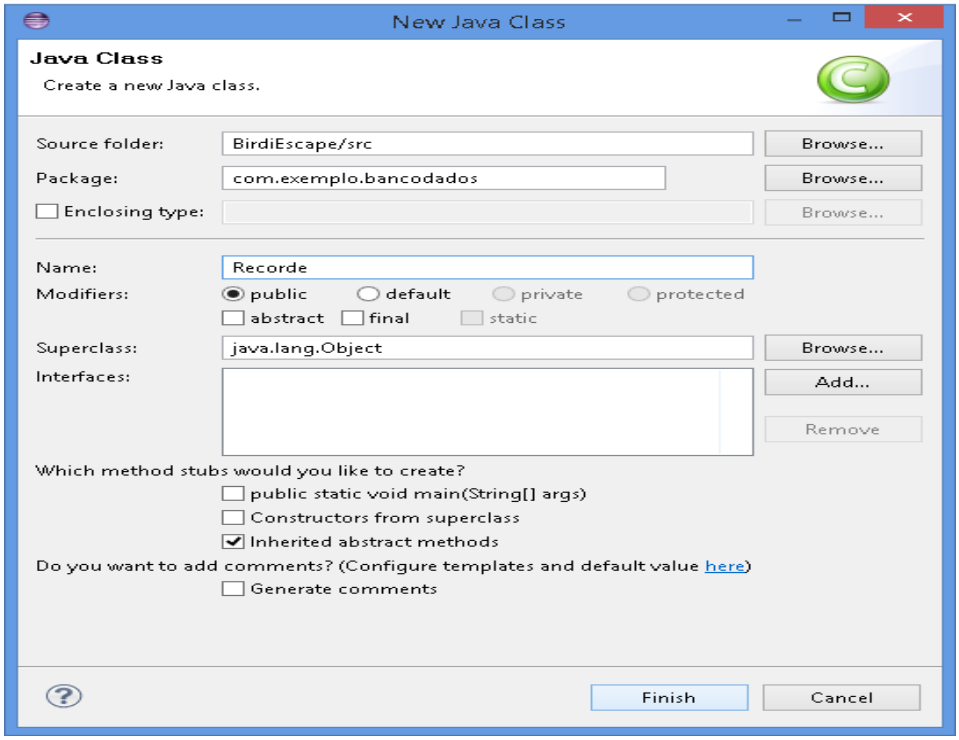

FIGURA 32 NOVA CLASSE JAVA

FONTE: O Autor (2015).

Agora estaremos mostrando a estrutura do jogo montado com todos os pacotes, classes, métodos e com o emulador funcionando com o jogo aberto no mesmo (FIGURA 33).

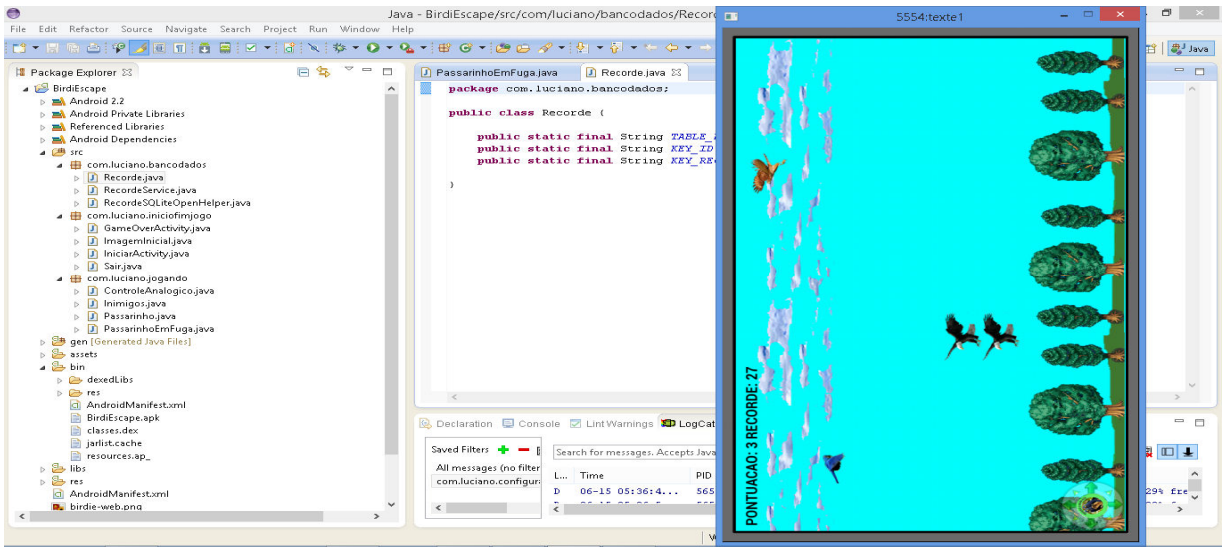

FIGURA 33 – ESTRUTURA DO JOGO PRONTO

FONTE: O Autor (2015).

8.3 Arquitetura do Jogo

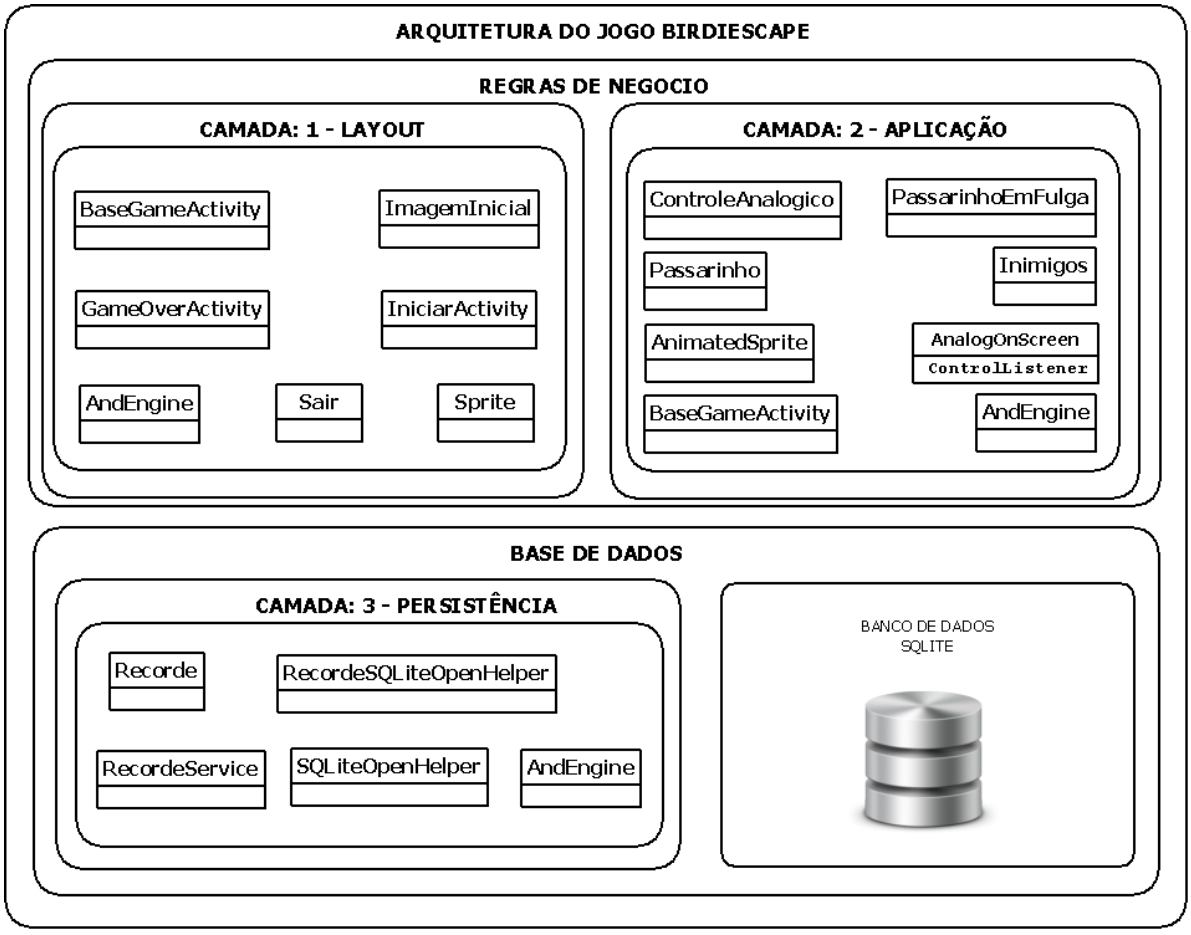

FIGURA 34 – ARQUITETURA DO JOGO EM CAMADAS

FONTE: O Autor (2015).

8.3.1 Camada 1 - Layout

BaseGameActivity: É uma *SuperClasse* da *engine AndEngine* que permitiu a implementação das subclasses no jogo.

ImagemInicial: É a classe que vai ajudar na definição da área que representará o botão sair do jogo, quando o botão sair é acionado é implementado o fim da *activity*.

GameOverActivity: É a classe que herda da classe *BaseGameActivity* da *AndEngine*. É ela quem define o que será mostrado na tela, quando há uma colisão entre os objetos do jogo, além da pontuação e recorde do jogador.

Iniciar*Activity*: É a classe que define o que será mostrado na tela ao iniciar o jogo, está classe também herda da classe Base*GameActivity*.

AndEngine: É a *Engine* que fornecerá as extensões e as classes para a implementação do jogo.

Sair: É a classe que herda os métodos da classe *Sprite* que faz parte da *engine AndEngine*, sendo ela responsável por definir a área que representará o botão sair do jogo.

Sprite: É uma Super Classe da *engine AndEngine*, a qual permitiu a implementação das subclasses no jogo.

8.3.2 Camada 2 - Aplicação

ControleAnalogico: É a classe que implementa a interface *AnalogOnScreenControlListener,* que possui as funções do projeto *AndEngine,* responsáveis pela criação de um *Joystick* analógico virtual, sendo este objeto criado através da *Sprite* que será representado na interface do sistema.

PassarinhoEmFulga: É a *Activity* do jogo, ela estende da *AndEngine* Base*GameActivity*, é nela que são instanciados os objetos das classes Passarinho, e Inimigos; onde são criado as *BitmapTextureAtlas*.

Passarinho: É a classe responsável pela criação e manipulação do objeto principal do jogo, onde é renderizada uma *Sprite* com 2 linhas e 3 colunas, em que é implementada a movimentação de ida e volta, invocada pela classe PassarinhoEmFulga.

Inimigos: É a classe que possui uma assinatura estendida da *AndEngine AnimatedSprite*, que é responsável por implementar movimentos as imagens PNG

chamadas de *Sprites* que são representadas virtualmente pelo movimento dos objetos.

AnimatedSprite: É uma Super Classe da *engine AndEngine*, permitindo a implementação de subclasses no jogo.

AndEngine: É a *engine* que fornecerá extensões e classes, para a implementação do jogo.

Base*GameActivity*: É uma Super Classe da *engine AndEngine*, que permitiu a implementação de subclasses no jogo.

AnalogOnScreenControlListener: É uma Super Classe da *engine AndEngine*, que permitiu a implementação de subclasses no jogo.

8.3.3 Camada 3 - Persistência

Recorde: É uma classe que declara os atributos estáticos que darão nome a tabela e o que será guardado nela.

Recorde*SQLiteOpenHelper*: É uma classe que herda da classe *SQLiteOpenHelper* e tem como função geral a criação do banco de dados utilizado no jogo.

Recorde*Service*: É uma classe que provê a criação da DataBase, a qual servirá para guardar a pontuação e o recorde do jogador.

AndEngine: É uma *engine* que fornecerá extensões e classes para implementação do jogo.

8.4 Protótipos de telas

FIGURA 35 – TELA DE INICIO

FONTE: O Autor (2015).

O protótipo do jogo “*BirdiEscape”*, vem demonstrar o funcionamento do mesmo. Apresentamos inicialmente como o botão “INICIAR” está posicionado na tela de abertura do jogo, sendo que ao tocar em cima deste nome “INICIAR”, o jogo será iniciado (FIGURA 35).

FIGURA 36 – TELA DO JOGO

FONTE: O Autor (2015).

É possível observar o Passarinho o qual o jogador terá que manter afastado dos outros considerados inimigos e, ao passar pelos seus inimigos, o jogador irá acumulando pontos. O controle do movimento do passarinho é feito através do deslizamento do dedo nas quatros posições, direita, esquerda, para cima e para baixo, por um *joystick* analógico que se encontra no canto inferior esquerdo da tela, quando ocorre a colisão do passarinho com um de seus inimigos esta tela e fechada e é aberta a tela de *Game Over* (FIGURA 36).

FIGURA 37 – TELA DE GAME OVER

FONTE: O Autor (2015).

Na tela de *Game Over* onde é salvo a pontuação e o recorde do jogador. Também é onde o jogador poderá escolher entre continuar o jogo, tocando no botão com o nome “CONTINUAR” ou sair do jogo tocando no botão com o nome de “SAIR”, mesmo após ter decidido sair do jogo a categoria recorde será salva para quando o jogador resolver jogar novamente, tendo que o jogador, superar seu próprio recorde (FIGURA 37).

8.5 Diagrama de Classes

Aqui é apresentado o diagrama de classes do jogo, o qual foi dividido em 3 camadas, sendo: *Layout*, Aplicação e Persistência.

8.5.1 Camada de *Layout*

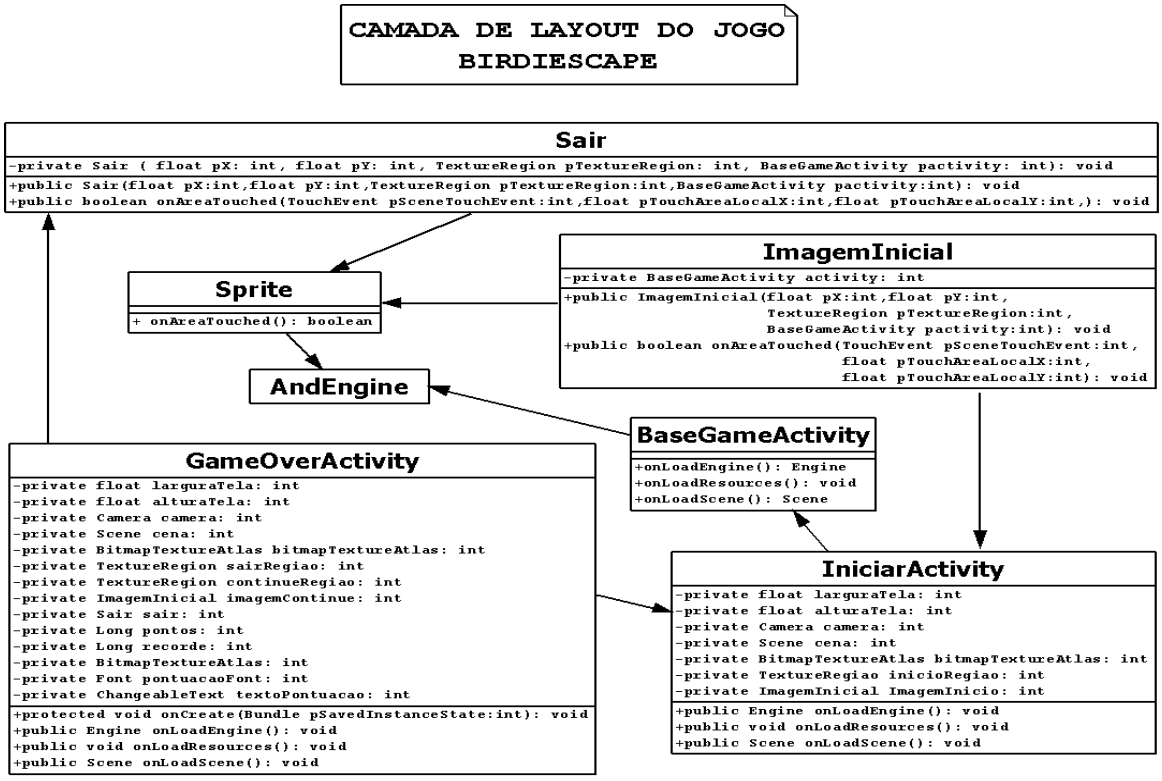

FIGURA 38 – DIAGRAMA DE CLASSE DA CAMADA DE LAYOUT

FONTE: O Autor (2015).

Podemos ver as relações entre as classes que geram as telas de menu e de *gameove*r. Podemos perceber a ligação entre as classes Sair e ImagemInicial com a classe *Sprite* da Biblioteca *AndEngine* e a relação de herança entre as classes *GameOverActivity* e Iniciar*Activity* com a classe Base*GameActivity,* também da biblioteca *AndEngine* (FIGURA 38).

8.5.2 Camada de Aplicação

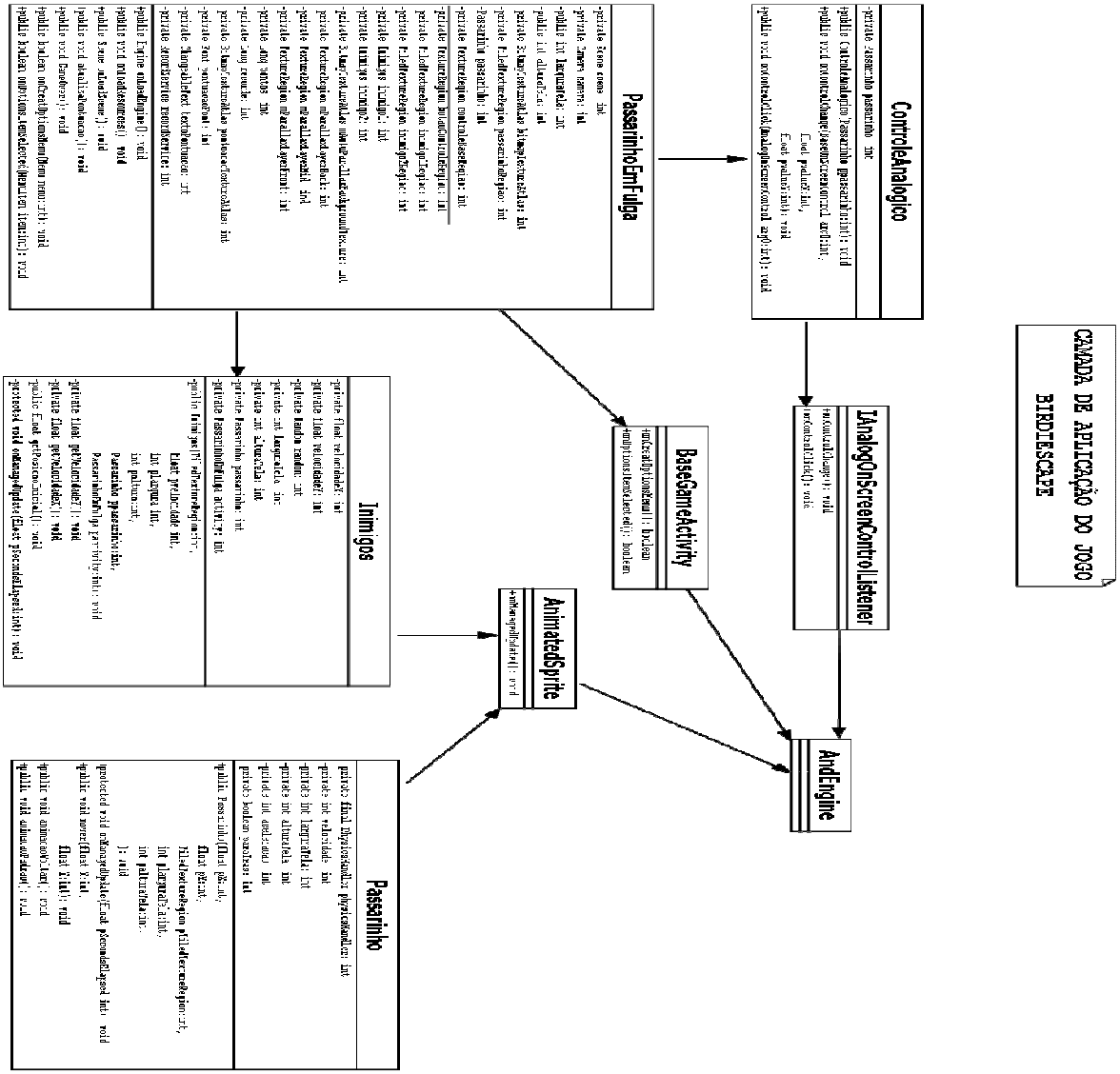

Figura 39 - DIAGRAMA DE CLASSES DA CAMADA DE APLICAÇÃO

FONTE: O Autor (2015).

É possível observar como fica evidente a relação entre as classes dessa camada e, que essa camada é a que produz todo o funcionamento do jogo, sendo possível perceber que a classe PassarinhoEmFulga é a classe que gere todas as outras classes chamando para o jogo (FIGURA 39).

8.5.3 Camada de Persistência

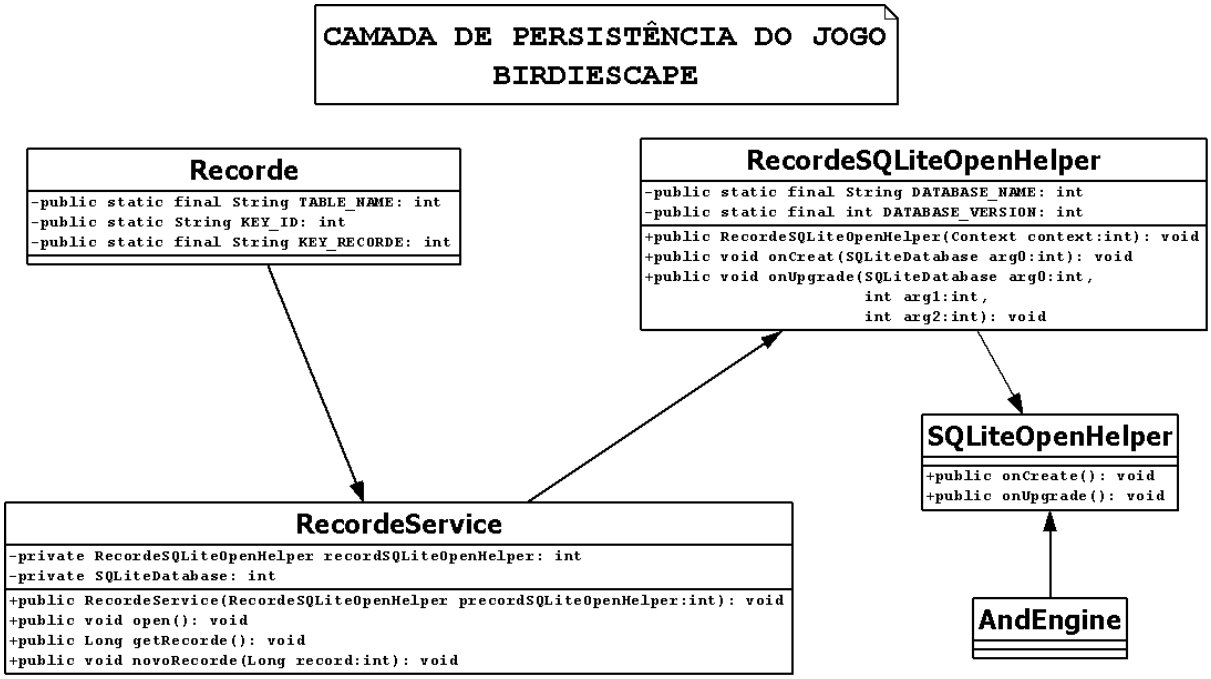

Figura 40 - DIAGRAMA DE CLASSES DA CAMADA DE PERSISTÊNCIA

FONTE: O Autor (2015).

No diagrama, estão as classes relacionadas com o banco de dados, nestas classes que serão armazenadas a pontuação temporária do jogador e o recorde atual da aplicação instalada no celular (FIGURA 40).

8.6 Descrição de Pacotes, Métodos e Classes

Neste momento é feita uma breve descrição dos pacotes, métodos e classes do jogo em desenvolvimento e, também é mostrado os códigos representados por figuras.

8.6.1 Pacote bancodados

Neste pacote que ocorre todo o processo de gerenciamento do banco de dados, sendo estas, leitura e gravação de recordes e pontuações, como já mostrado anteriormente no SGBD utilizado aqui o *SQLite*.

SQLite é uma biblioteca de *software* autossuficiente, sem a necessidade de um servidor, com o uso de configuração zero, transacional de banco de dados SQL, sendo que o seu código fonte está no domínio público (SQLITE, 2015).

Classe Recorde: É uma classe que declara atributos estáticos que darão nome as tabelas e o que serão armazenados nela. A classe é composta dos códigos mostrado logo abaixo (FIGURA 41).

```
2
3   public class Recorde {
4
5       public static final String TABLE_NAME = "recorde";
6       public static final String KEY_ID = "_id";
7       public static final String KEY_RECORDE = "recorde";
8
9   }
10
```

FIGURA 41 – CLASSE RECORDE

FONTE: O Autor (2015).

RecordeService: É a classe que provê a criação da database que servirá para guardar a pontuação e o recorde do jogador, que se encontram nesta classe os seguintes métodos:

Este código do método Recorde*Service*(), apenas altera o valor do atributo *recordSQLiteOpenHelper* para que seja guardado um novo recorde (FIGURA 42).

```
10
11      public RecordeService(RecordeSQLiteOpenHelper precordSQLiteOpenHelper) {
12
13          recordSQLiteOpenHelper = precordSQLiteOpenHelper;
14
15      }
```

FIGURA 42 – MÉTODO RECORDESERVICE

FONTE: O Autor (2015).

Através deste método *public void open*() irá iniciar a database do jogo (FIGURA 43).

```
16
17      public void open(){
18
19          database = recordSQLiteOpenHelper.getWritableDatabase();
20
21      }
```

FIGURA 43 – MÉTODO OPEN

FONTE: O Autor (2015).

O método getRecorde(), é o que faz a consulta na tabela do banco de dados, buscando o recorde atual para que depois seja retornado no método (FIGURA 44).

```
22
23      public Long getRecorde() {
24
25          Cursor cursor;
26          cursor = database.query(Recorde.TABLE_NAME,
27                  new String[]{Recorde.KEY_ID+",MAX("+Recorde.KEY_RECORDE+") as "
28                  +Recorde.KEY_RECORDE},
29                  null, null,
30                  null, null, null);
31          cursor.moveToFirst();
32          Long retorno = cursor.getLong(cursor.getColumnIndex(Recorde.KEY_RECORDE));
33          cursor.close();
34          return retorno;
35
36      }
```

FIGURA 44 – MÉTODO GETRECORDE

FONTE: O Autor (2015).

O código do método novoRecorde(), é o que faz a gravação do novo recorde do app, realizando assim, a atualização do recorde na tabela do banco de dados(FIGURA 45).

```
37
38      public void novoRecorde(Long record){
39
40          ContentValues values = new ContentValues();
41          values.put(Recorde.KEY_RECORDE, record);
42          database.update(Recorde.TABLE_NAME, values, "_id = ?",
43          new String[]{Long.toString(1)});
44
45      }
```

FIGURA 45 – MÉTODO NOVORECORDE

FONTE: O Autor (2015).

Classe RecordeSQLiteOpenHelper: É uma classe que herda da classe SQLiteOpenHelper. Esta classe tem como função geral criar o banco de dados do jogo. Nesta classe é encontrado os seguintes métodos:

Os códigos do método RecordeSQLiteOpenHelper(), vão definir a versão da database e o nome que será dado ao banco de dados do jogo(FIGURA 46).

```
11
12      public RecordeSQLiteOpenHelper(Context context) {
13
14          super(context, DATABASE_NAME, null, DATABASE_VERSION);
15
16      }
```

FIGURA 46 – MÉTODO RECORDESQLITEOPENHELPER

FONTE: O Autor (2015).

Nos códigos do método *OnCreate*(), que é um *Override* e cria o banco de dados do jogo passando o nome da tabela, sendo a *primary key* e o recorde inicial (FIGURA 47).

```
17
18      @Override
19      public void onCreate(SQLiteDatabase arg0) {
20
21          String CREATE_TABLE_RECORD = "CREATE TABLE "+ Recorde.TABLE_NAME +
22                  " ("+Recorde.KEY_ID + " integer primary key, " +
23                  Recorde.KEY_RECORDE + " long); ";
24          arg0.execSQL(CREATE_TABLE_RECORD);
25
26          String INICIALIZA_TABELA_RECORD = "INSERT INTO "+Recorde.TABLE_NAME +
27                  "("+Recorde.KEY_ID+","+Recorde.KEY_RECORDE+")VALUES(1,0);";
28
29          arg0.execSQL(INICIALIZA_TABELA_RECORD);
30
31      }
```

FIGURA 47 – MÉTODO ONCREATE

FONTE: O Autor (2015).

A partir desses códigos do método *onUpgrade*(), que são um *Override* da classe *SQLiteOpenHelper*, não houve a necessidade da implementação deste código (FIGURA 48).

```
32
33      @Override
34      public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2) {
35
36          // TODO Auto-generated method stub
37
38      }
```

FIGURA 48 – MÉTODO ONUPGRADE

FONTE: O Autor (2015).

8.6.2 Pacote iniciofimjogo

É neste pacote que é definido os objetivos das estruturas iniciais do jogo, como a tela inicial, tela do jogo e tela de saida. Este pacote contém as classes essenciais que trabalham em conjunto com a camada da aplicação e faz rodar o jogo.

Classe Sair: Nesta classe é definida a área que representará o botão sair do jogo, esta classe herda os métodos da classe *Sprite* que vem da biblioteca *AndEngine*. Sendo encontrado os seguintes métodos:

Nos códigos do método Sair(), é utilizado o construtor da Super Classe que define a área que servirá como um botão no jogo (FIGURA 49).

```
10
11      public Sair(float pX, float pY, TextureRegion pTextureRegion,
12      BaseGameActivity pactivity) {
13
14          super(pX, pY, pTextureRegion);
15          this.activity = pactivity;
16
17      }
```

FOGURA 49 – MÉTODO SAIR

FONTE: O Autor (2015).

Nos códigos do método *onAreaTouched*(), que é um *Override* de um método da classe *Sprite*, é definido o que será feito após a área definida ser clicada, neste caso o aplicativo desta classe será fechada. A *activity.finish*(), é acionada através da definição das variaveis *floatpTouch*AreaLocalX e *floatpTouch*AreaLocalY (FIGURA 50).

```
18          @Override
19          public boolean onAreaTouched(TouchEvent pSceneTouchEvent,
20                  float pTouchAreaLocalX, float pTouchAreaLocalY) {
21
22              activity.finish();
23              return super
24                      .onAreaTouched(pSceneTouchEvent, pTouchAreaLocalX,
25                      pTouchAreaLocalY);
26
27          }
```

FIGURA 50 – MÉTODO ONAREATOUCHED

FONTE: O Autor (2015).

Classe *GameOverActivity*: Esta classe herda da classe da *AndEngine* chamada de *BaseGameActivity*. Essa classe define o que aparecerá quando o jogo for finalizado, acionando o *Game Over*.

Nos códigos do método *onCreate*(), que vem da herança da classe *BaseGameActivity*, onde é gerado um *log*, mostrando na tela a pontuação atual e o recorde do jogo (FIGURA 51).

```
51      @Override
52      protected void onCreate(Bundle pSavedInstanceState) {
53          if (this.getIntent() != null && this.getIntent().getExtras() != null) {
54              Bundle bundle = this.getIntent().getExtras();
55              pontos = bundle.getLong("pontos");
56              recorde = bundle.getLong("recorde");
57          }
58          super.onCreate(pSavedInstanceState);
59      }
```

FIGURA 51 – MÉTODO ONCREATE PONTUAÇÃO

FONTE: O Autor (2015).

Esses códigos do método *Engine onLoadEngine*(), buscam o tamanho da tela do dispositivo, através do *getSystemService*(*Context WINDOW_SERVICE*), vindo a guardar o tamanho da tela nos atributos “this larguraTela” e “this alturaTela” (FIGURA 52).

```
65      public Engine onLoadEngine() {
66          // AQUI E PEGO O TAMANHO DA TELA DO APARELHO
67          Display display = ((WindowManager) getSystemService(Context.WINDOW_SERVICE))
68                  .getDefaultDisplay();
69          this.larguraTela = display.getWidth();
70          this.alturaTela = display.getHeight();
```

FIGURA 52 – MÉTODO ONLOADENGINE

FONTE: O Autor (2015).

Aqui é apresentado os códigos que define a câmera (FIGURA 53).

```
71
72          // AQUI E DEFENIDA A CAMERA
73          this.camera = new Camera(0, 0, this.larguraTela, this.alturaTela);
74
```

FIGURA 53 – DEFINIÇÃO DE CAMERA

FONTE: O Autor (2015).

Neste método é carregado as opções da *Engine*, passando a resolução da tela e as configurações da câmera para que sejam repassadas as configurações do *Engine* para o dispositivo (FIGURA 54).

```
75
76          // AQUI E DEFENIDA AS OPCOES DA ENGINE
77          final EngineOptions engineOptions = new EngineOptions(true,
78                  ScreenOrientation.LANDSCAPE, new RatioResolutionPolicy(
79                          this.larguraTela, this.alturaTela), this.camera);
80          final Engine engine = new Engine(engineOptions);
81          return engine;
82      }
```

FIGURA 54 – ENGINEOPTIONS

FONTE: O Autor (2015).

Os códigos do método *onLoadResources*(), são carregados os recursos necessários para gerar a cena do *Game Over*, carregando as texturas das regiões de toque da tela, também é carregada as definições do plano de fundo e carregado as fontes da pontuação (FIGURA 55).

```
84    public void onLoadResources() {
85        // AQUI E DEFINIDO A REGIAO DE TQUE DA TELA
86        BitmapTextureAtlasTextureRegionFactory.setAssetBasePath("los/");
87        this.bitmapTextureAtlas = new BitmapTextureAtlas(512, 512,
88                TextureOptions.BILINEAR);
89        this.sairRegiao = BitmapTextureAtlasTextureRegionFactory
90                .createFromAsset(this.bitmapTextureAtlas, this, "SairJogo.png", 200, 0);
91        this.continueRegiao = BitmapTextureAtlasTextureRegionFactory
92                .createFromAsset(this.bitmapTextureAtlas, this, "ContinuarJogo.png", 0, 0);
93        this.mEngine.getTextureManager().loadTexture(this.bitmapTextureAtlas);
94
95        // AQUI E DEFENIDO O PLANO DE FUNDO
96        this.mBackgroundTexture = new BitmapTextureAtlas(1024, 1024, TextureOptions.DEFAULT);
97        mBgTexture = BitmapTextureAtlasTextureRegionFactory.createFromAsset
98        (this.mBackgroundTexture, this, "fundoJogolongo.png", 0, 0);
99        this.mEngine.getTextureManager().loadTextures(this.mBackgroundTexture);
100
101       // AQUI E DEFENIDO O TIPO, TAMANHO E A COR DA PONTUACAO
102       pontuacaoTextureAtlas = new BitmapTextureAtlas(256, 256,
103               TextureOptions.BILINEAR_PREMULTIPLYALPHA);
104       this.pontuacaoFont = new Font(pontuacaoTextureAtlas, Typeface.create(
105               Typeface.DEFAULT, Typeface.BOLD), 30, true, Color.BLACK);
106       this.mEngine.getTextureManager().loadTexture(pontuacaoTextureAtlas);
107       this.mEngine.getFontManager().loadFont(pontuacaoFont);
108   }
```

FIGURA 55 – MÉTODO ONLOADRESOURCES QUE CARREGA AS CENAS DO GAMEOVER

FONTE: O Autor (2015).

Através destes códigos do método *Scene onLoadScene*(), é criada a cena do *Game Over*, sendo escolhido a posição do plano de fundo, a posição do botão continuar, a posição do botão sair e, também, é definido onde a pontuação e o recorde do jogador irá ficar na tela de *Game Over* do jogo (FIGURA 56).

```
110     public Scene onLoadScene() {
111         cena = new Scene();
112 
113         // AQUI E DEFINIDO A POSICAO DO PLANO DE FUNDO(IMAGEM/SPRITE)
114         final int centroA = (int) ((larguraTela - mBgTexture.getWidth()) / 2);
115         final int centroB = (int) ((alturaTela - mBgTexture.getHeight()) / 2);
116         SpriteBackground bg = new SpriteBackground(new Sprite(centroA, centroB,
117                 this.mBgTexture));
118         cena.setBackground(bg);
119 
120         // AQUI E DEFINIDO A POSICAO DO BOTAO CONNTINUAR
121         int centroX = (int) (this.larguraTela / 2) - 110;
122         int centroY = (int) (this.alturaTela / 2) + 20;
123         this.imagemContinue = new ImagemInicial(centroX, centroY,
124                 this.continueRegiao, this);
125         cena.attachChild(this.imagemContinue);
126         cena.registerTouchArea(this.imagemContinue);
127 
128         // AQUI E DEFINIDO A POSICAO DO BOTAO SAIR
129         int centroZ = (int) (this.larguraTela / 2) - 60;
130         int centrow = (int) (this.alturaTela / 2)+110;
131         this.sair = new Sair(centroZ, centrow, this.sairRegiao, this);
132         cena.attachChild(this.sair);
133         cena.registerTouchArea(this.sair);
134 
135         // AQUI E DEFENIDO O TEXTO E A POSICAO DA PONTUACAO
136         String textoPontos = "Pontuacao: " + pontos + " / Recorde: " + recorde;
137         int centroXpontos = (int) ((this.larguraTela / 2) - (this.pontuacaoFont
138                 .getStringWidth(textoPontos) / 2));
139         int centroYpontos = (int) ((this.alturaTela / 2) - (this.pontuacaoFont
140                 .getLineHeight() / 2))+ 80;
141         textoPontuacao = new ChangeableText(centroXpontos, centroYpontos,
142                 this.pontuacaoFont, textoPontos);
143         cena.attachChild(textoPontuacao);
144 
145         return cena;
```

FIGURA 56 – MÉTODO ONLOADSCENE

FONTE: O Autor (2015).

Classe ImagemInicial: É a classe que define a área em que será representado o botão Sair do jogo. Esta classe herda os métodos da classe *Sprite*, que pertence a biblioteca *AndEngine*, e é encontrada nesta classe os seguintes métodos:

ImagemInicial(): onde é utilizado o construtor da Super Classe para definir a área que servirá como se fosse um botão no jogo (FIGURA 57).

```
17
18      public ImagemInicial(float pX, float pY, TextureRegion pTextureRegion,
19      BaseGameActivity pactivity) {
20
21          super(pX, pY, pTextureRegion);
22          this.activity = pactivity;
23
24      }
```

FIGURA 57 – MÉTODO IMAGEMINICIAL

FONTE: O Autor (2015).

A *onAreaTouched*(): é uma *Override* de um método da classe *Sprite*, neste método é definido o que será realizado após a área ser pressionada, neste caso o aplicativo será acionado quando instanciado o objeto *Intent*, passando para a classe PassarinhoEmFulga e a iniciando (FIGURA 58).

```
25
26      @Override
27      public boolean onAreaTouched(TouchEvent pSceneTouchEvent,
28              float pTouchAreaLocalX, float pTouchAreaLocalY) {
29
30          Intent intent = new Intent(activity,PassarinhoEmFuga.class);
31          activity.startActivity(intent);
32          activity.finish();
33          return super
34                  .onAreaTouched(pSceneTouchEvent, pTouchAreaLocalX, pTouchAreaLocalY);
35
36      }
```

FIGURA 58 – MÉTODO ONAREATOUCHED

FONTE: O Autor (2015).

Classe IniciarActivity: Esta classe é quem define o que aparecerá na tela quando o jogo for iniciado, sendo que esta classe herda da classe *BaseGameActivity* da *AndEngine*. Os códigos desta classe que busca o tamanho da tela do dispositivo,

guarda o tamanho da tela nos atributos “this.larguraTela” e “this.alturaTela” (FIGURA 59).

```
42
43          // AQUI E PEGO O TAMANHO DA TELA DO APARELHO
44          Display display = ((WindowManager) getSystemService(Context.WINDOW_SERVICE))
45                  .getDefaultDisplay();
46          this.larguraTela = display.getWidth();
47          this.alturaTela = display.getHeight();
48
```

FIGURA 59 – TAMANHO DA TELA DO DISPOSITIVO

FONTE: O Autor (2015).

Após ser definido o tamanho da tela, é apresentado os códigos que definem a câmera (FIGURA 60).

```
48
49          // AQUI E DEFENIDA A CAMERA
50          this.camera = new Camera(0, 0, this.larguraTela, this.alturaTela);
51
```

FIGURA 60 – DEFINE A CAMERA

FONTE: O Autor (2015).

Nestes códigos do método *onLoadEngine*(), são carregadas as opções da *Engine*, passando a resolução da tela e as configurações da câmera, para que sejam passadas as configurações da *Engine* para o dispositivo (FIGURA 61).

```
51
52          // AQUI E DEFENIDA AS OPCOES DA ENGINE
53          final EngineOptions engineOptions = new EngineOptions(true,
54                  ScreenOrientation.LANDSCAPE, new RatioResolutionPolicy(
55                          this.larguraTela, this.alturaTela), this.camera);
56          final Engine engine = new Engine(engineOptions);
57          engineOptions.setWakeLockOptions(WakeLockOptions.SCREEN_ON);
58          return engine;
59
60      }
```

FIGURA 61 – OPÇÕES DA ENGINE

FONTE: O Autor (2015).

Nos códigos do método *onLoadResources*(), onde é direcionada para a pasta das imagens que serão utilizadas, são carregados os recursos necessários para gerar o botão Iniciar do jogo e, também, é carregada as texturas das regiões de toque na tela do jogo e o plano de fundo (FIGURA 62).

```
61
62      public void onLoadResources() {
63
64          // AQUI E DIRECIONADA PARA A PASTA ONDE FICAM AS IMAGENS
65          BitmapTextureAtlasTextureRegionFactory.setAssetBasePath("log/");
66
67          // AQUI E DEFENIDO O BOTAO INICIAR COM A UTILIZACAO DA IMAGEM
68          this.bitmapTextureAtlas = new BitmapTextureAtlas(512, 512,TextureOptions.BILINEAR);
69          this.inicioRegiao = BitmapTextureAtlasTextureRegionFactory
70                  .createFromAsset(this.bitmapTextureAtlas, this, "menu.png",0,0);
71          this.mEngine.getTextureManager().loadTexture(this.bitmapTextureAtlas);
72
73          // AQUI E DEFENIDO O PLANO DE FUNDO COM A UTILIZACAO DA IMAGEM
74          this.mBackgroundTexture = new BitmapTextureAtlas(1024, 1024, TextureOptions.DEFAULT);
75          mBgTexture = BitmapTextureAtlasTextureRegionFactory.createFromAsset(this.mBackgroundTexture,
76          this, "fundoInicio.png", 0, 0);
77          this.mEngine.getTextureManager().loadTextures(this.mBackgroundTexture);
78
79      }
```

FIGURA 62 – MÉTODO ONLOADRESOURCES

FONTE: O Autor (2015).

Nos códigos do método *onLoadScene*(), onde é criada a cena de inicio do jogo, é escolhida a posição da imagem do plano de fundo, através dos atributos centroA e centroB, e, em cima desta imagem, é escolhida a posição do botão iniciar através dos atributos centroX e centroY (FIGURA 63).

```
80
81      public Scene onLoadScene() {
82
83          cena = new Scene();
84
85          // AQUI E DEFENIDO A POSICAO DO PLANO DE FUNDO COM A UTILIZACAO DA IMAGEM
86          final int centroA = (int) ((larguraTela - mBgTexture.getWidth()) / 2);
87          final int centroB = (int) ((alturaTela - mBgTexture.getHeight()) / 2);
88          SpriteBackground bg = new SpriteBackground(new Sprite(centroA, centroB,
89                  this.mBgTexture));
90          cena.setBackground(bg);
91
92          // AQUI E DEFENIDA A POSICAO DO BOTAO INICIAR
93          int centroX = (int)((this.larguraTela/2))- 180;
94          int centroY = (int)((this.alturaTela/2))- 50;
95          this.imagemInicio = new ImagemInicial(centroX, centroY, this.inicioRegiao,this);
96          cena.attachChild(this.imagemInicio);
97          cena.registerTouchArea(this.imagemInicio);
98
99          return cena;
100
101     }
```

FIGURA 63 – MÉTODO ONLOADSCENE CRIA O INICIO DO JOGO

FONTE: O Autor (2015).

8.6.3 Pacote jogando

Este é o principal pacote do jogo, é nele que se encontra o motor, e as classes que comandam os movimentos e os atritos do passarinho no jogo.

Classe ControleAnalogico: É a classe que implementa a *IAnalogOnScreenControlListener*, que são uma das funções da *AndEngine*, que é responsável pela criação de *joystick* analogico virtual, sendo que este objeto é criado através da *Sprite*, que será representada na interface do sistema e, também, é usado o atributo do tipo Passarinho que vai representar a classe que será controlada pelo objeto, apresentada logo abaixo (FIGURA 64).

```
6
7  public class ControleAnalogico implements IAnalogOnScreenControlListener {
8
9      private Passarinho passarinho;
10
```

FIGURA 64 – CLASSE CONTROLEANALOGICO

FONTE: O Autor (2015).

Nos códigos do método ControleAnalogico(), será instanciado o passarinho que é recebido pela classe ControleAnalogico (FIGURA 65).

```
10
11     public ControleAnalogico(Passarinho ppassarinho) {
12
13         this.passarinho = ppassarinho;
14
15     }
```

FIGURA 65 – MÉTODO CONTROLEANALOGICO

FONTE: O Autor (2015).

Nos códigos do método *onControlChange*(), onde é exibido os parâmetros que envolvem os dados, quando o jogador movimenta o *joystick* analógico, esses valores são atribuidos ao método mover da classe Passarinho, vindo a controlar o passarinho através deste *joystick* (FIGURA 66).

```
16
17     public void onControlChange(BaseOnScreenControl arg0, float pValueX,
18     float pValueY) {
19
20         passarinho.mover(pValueX, pValueY);
21
22     }
```

FIGURA 66 – MÉTODO ONCONTROLCHANGE

FONTE: O Autor (2015).

Classe PassarinhoEmFulga: Esta é a *activity* que é herdada da classe *BaseGameActivity* da *AndEngine*. Nesta classe são instanciados os objetos das classes Passarinho e Inimigos, são criados as *BitmapTextureAtlas*, que são como telas para pintura de imagens, as quais são definidas as regiões onde serão pintadas todas as texturas *TextureRegions*. Se houver uma má definição das regiões, pode ocasionar em uma sobreposição das imagens e, para o perfeito funcionamento da *activity,* é necessário o auxilio de uma *Scene* e da Camera que mostra o que está sendo visualizado no momento, conforme mostrado logo abaixo (FIGURA 67).

```
37
38  public class PassarinhoEmFuga extends BaseGameActivity {
39
40      //AQUI E DEFENIDA A ENGINE
41      private Scene scene;
42      private Camera camera;
43      public int larguraTela;
44      public int alturaTela;
45
46      // AQUI E DEFINIDA AS IMAGENS
47      private BitmapTextureAtlas bitmapTextureAtlas;
48
49      private TiledTextureRegion passarinhoRegiao;
50      private TiledTextureRegion inimigo1Regiao;
51      private TiledTextureRegion inimigo2Regiao;
52
53      private TextureRegion baseControleRegiao;
54      private TextureRegion botaoControleRegiao;
55
56      private Passarinho passarinho;
57      private Inimigos inimigo1;
58      private Inimigos inimigo2;
59
60      // AQUI E DEFINIDO O PLANO DE FUNDO E AS REGIOES
61      private BitmapTextureAtlas mAutoParallaxBackgroundTexture;
62
63      private TextureRegion mParallaxLayerBack;
64      private TextureRegion mParallaxLayerMid;
65      private TextureRegion mParallaxLayerFront;
66
```

FIGURA 67 – PASSARINHOEMFULGA

FONTE: O Autor (2015).

Nesta classe *public Engine onLoadEngine*(), que é a *engine* principal da *activity*, é criada uma instancia do gerenciador de recordes, sendo utilizado um método *WindowManager,* o qual pega o valor da altura e da largura da tela do dispositivo e envia os valores para a tela de visualização da câmera. A câmera muda o modo de visualização para *ScreenOrientation.LANDSCAPE*. Também foi utilizado uma politica de escala *FillResolutionPolicy* para que seja ajustado ao máximo a câmera ao tamanho da tela do jogo (FIGURA 68).

```
80
81      public Engine onLoadEngine() {
82
83          RecordeSQLiteOpenHelper recordSqliteOpenHelper = new RecordeSQLiteOpenHelper(
84                  this);
85          recordService = new RecordeService(recordSqliteOpenHelper);
86          recordService.open();
87
88          // AQUI E PEGO O TAMANHO DA TELA DO APARELHO
89          Display display = ((WindowManager) getSystemService(Context.WINDOW_SERVICE))
90          .getDefaultDisplay();
91          this.larguraTela = display.getWidth();
92          this.alturaTela = display.getHeight();
93
94          // AQUI E DEFENIDA A CAMERA
95          this.camera = new Camera(0, 0, this.larguraTela, this.alturaTela);
96
97          // AQUI E DEFENIDA AS OPCOES DA ENGINE
98          final EngineOptions engineOptions = new EngineOptions(true,
99                  ScreenOrientation.LANDSCAPE, new FillResolutionPolicy(
100                         ), this.camera);
101         final Engine engine = new Engine(engineOptions);
102         return engine;
103
104     }
```

FIGURA 68 – CLASSE ENGINE

FONTE: O Autor (2015).

Nos códigos do método *onLoadResources*(), o tema do jogo ganha cor, pois é indicado onde estarão as imagens que se encontram na pasta assets e, também, é inicializado o *BitmapTextureAtlas*, como se fosse um quadro, que depois é preenchido com as imagens como se fosse pintado (FIGURA 69).

```
105
106     public void onLoadResources() {
107
108         // AQUI E DEFENIDO O CAMINHO PADRÃO DENTRO DA PASTA /ASSETS PARA AS
109         // IMAGENS
110         BitmapTextureAtlasTextureRegionFactory.setAssetBasePath("los/");
111
112         // AQUI E INICIALIZADO O BITMAPTEXTUREATLAS,
113         // PENSE ELE COMO UM QUADRO QUE E PREENCHIDO COM VARIAS IMAGENS
114         this.bitmapTextureAtlas = new BitmapTextureAtlas(512, 512,
115         TextureOptions.DILINEAR);
116
117         // AQUI E CARREGADA A IMAGEM PASSANDO O BITMAPTEXTUREATLAS.
118         this.passarinhoRegiao = BitmapTextureAtlasTextureRegionFactory
119                 .createTiledFromAsset(this.bitmapTextureAtlas,
120                 this, "Passarinho.png", 0, 0, 4, 2);
121         this.baseControleRegiao = BitmapTextureAtlasTextureRegionFactory
122                 .createFromAsset(this.bitmapTextureAtlas,
123                 this, "baseControle.png",384,0);
124         this.botaoControleRegiao = BitmapTextureAtlasTextureRegionFactory
125                 .createFromAsset(this.bitmapTextureAtlas,
126                 this, "botaoControle.png",384,128);
127         this.inimigo1Regiao = BitmapTextureAtlasTextureRegionFactory
128                 .createTiledFromAsset(this.bitmapTextureAtlas,
129                 this, "inimigo1.png", 20, 200, 1, 1);
130         this.inimigo2Regiao = BitmapTextureAtlasTextureRegionFactory
131                 .createTiledFromAsset(this.bitmapTextureAtlas,
132                 this, "inimigo2.png", 60, 300,1,1);
133
```

FIGURA 69 – MÉTODO ONLOADRESOURCES GANHANDO COR

FONTE: O Autor (2015).

Ainda no método *onLoadResources*(), são carregadas as imagens para a *Engine*, definindo o plano de fundo e, também, é definido o tipo, tamanho e a cor da pontuação, que serão mostrado no jogo (FIGURA 70).

```
133
134         // AQUI E CARREGADA AS IMAGENS PARA A ENGINE
135         this.mEngine.getTextureManager().loadTexture(this.bitmapTextureAtlas);
136
137         // AQUI E DEFENIDO OS PLANOS DE FUNDO
138         this.mAutoParallaxBackgroundTexture = new BitmapTextureAtlas
139         (1024, 1024);
140         this.mParallaxLayerFront = BitmapTextureAtlasTextureRegionFactory
141                 .createFromAsset(this.mAutoParallaxBackgroundTexture,
142                 this, "floresta.png", 0, 0);
143         this.mParallaxLayerBack = BitmapTextureAtlasTextureRegionFactory
144                 .createFromAsset(this.mAutoParallaxBackgroundTexture,
145                 this, "fundo.png", 0, 188);
146         this.mParallaxLayerMid = BitmapTextureAtlasTextureRegionFactory
147                 .createFromAsset(this.mAutoParallaxBackgroundTexture,
148                 this, "nuvens.png", 0, 669);
149         this.mEngine.getTextureManager().loadTexture(mAutoParallaxBackgroundTexture);
150
151         // AQUI E DEFENIDO O TIPO, TAMANHO E COR DA PONTUACAO
152         pontuacaoTextureAtlas = new BitmapTextureAtlas(256, 256,
153                 TextureOptions.BILINEAR_PREMULTIPLYALPHA);
154         this.pontuacaoFont = new Font(pontuacaoTextureAtlas, Typeface.create(
155                 Typeface.DEFAULT, Typeface.BOLD), 30, true, Color.BLACK);
156         this.mEngine.getTextureManager().loadTexture(pontuacaoTextureAtlas);
157         this.mEngine.getFontManager().loadFont(pontuacaoFont);
158     }
```

FIGURA 70 – CONTINUAÇÃO DO MÉTODO ONLOADRESOURCES

FONTE: O Autor (2015).

Através do código do método *public Scene onLoadScene*() é criada a cena do jogo, onde a cena começa a ganhar forma. Tudo o que foi definido no método onLoadResources() agora é visualizado na interface da tela, através da criação da cena pela *scene* = *NewScene*() (FIGURA 71).

```
159
160     public Scene onLoadScene() {
161
162         // AQUI E CRIADO A CENA DO JOGO
163         scene = new Scene();
164
```

FIGURA 71 – MÉTODO SCENE ONLOADSCENE

FONTE: O Autor (2015).

A partir dos códigos do método onLoadScene() é utilizado a AndEngine AutoParallaxBackground, que cria a movimentação dos backgrounds, dando o efeito de que a tela continua em movimento (FIGURA 72).

```
164
165         // AQUI E DEFENIDO O PLANO DE FUNDO
166         final AutoParallaxBackground autoParallaxBackground = new AutoParallaxBackground
167         (0, 10, 10, 5);
168         autoParallaxBackground.attachParallaxEntity(new ParallaxEntity(0.0f,
169                 new Sprite(0, alturaTela - this.mParallaxLayerBack.getHeight(),
170                 this.mParallaxLayerBack)));
171         autoParallaxBackground.attachParallaxEntity(new ParallaxEntity(-5.0f,
172                 new Sprite(0,40, this.mParallaxLayerMid)));
173         autoParallaxBackground.attachParallaxEntity(new ParallaxEntity(-10.0f,
174                 new Sprite(0, alturaTela - this.mParallaxLayerFront.getHeight(),
175                 this.mParallaxLayerFront)));
176         scene.setBackground(autoParallaxBackground);
177
```

FIGURA 72 – DEFININDO O PLANO DE FUNDO

FONTE: O Autor (2015).

Nestes códigos do método *onLoadScene*() é colocado o passarinho entre a floresta e as nuvens, gerando a referência de centro da imagem para os cálculos de colisão e é iniciado com o *scene.attachChild*(passarinho) (FIGURA 73).

```
177
178         // AQUI E DEFENIDO A POSICAO DO PASSARINHO
179         int centroXpassarinho = (this.larguraTela/2)-(this.passarinhoRegiao.getWidth()/2);
180         int centroYpassarinho = (this.alturaTela/2)-(this.passarinhoRegiao.getHeight()/2);
181         this.passarinho = new Passarinho(centroXpassarinho, centroYpassarinho,
182                 this.passarinhoRegiao,this.larguraTela, this.alturaTela);
183         this.passarinho.setScale(1.25);
184         scene.attachChild(passarinho);
185
```

FIGURA 73 – DEFININDO A POSICAO DO PASSARINHO

FONTE: O Autor (2015).

Nos codigos do método *onLoadScene*(), é entregue o controle para o jogador, obtendo o passarinho que foi criado e entregue à sua referência para o *ControleListener* e, também, é marcada a posição do controle e seu tamanho na tela e passado para rodar na cena (FIGURA 74).

```
185
186         // AQUI E DEFENIDO O CONTROLE
187         AnalogOnScreenControl controle = new AnalogOnScreenControl(
188                 0,this.alturaTela-this.baseControleRegiao.getHeight(),
189                 this.camera,this.baseControleRegiao,this.botaoControleRegiao,
190                 0.1f, 200,new ControleAnalogico(this.passarinho));
191
192         controle.getControlBase().setBlendFunction(GL10.GL_SRC_ALPHA,
193         GL10.GL_ONE_MINUS_SRC_ALPHA);
194         controle.getControlBase().setAlpha(0.5f);
195         controle.getControlBase().setScaleCenter(0, 128);
196         controle.getControlBase().setScale(1f);
197         controle.getControlKnob().setScale(1f);
198         controle.refreshControlKnobPosition();
199         scene.setChildScene(controle);
200
```

FIGURA 74 – DEFININDO CONTROLE

FONTE: O Autor (2015).

Nos códigos do método *onLoadScene*(), na parte que carregam os inimigos do passarinho, o texto da pontuação, do recorde e retorna a cena pronta do jogo, os inimigos aparecerão da seguinte maneira: com a relação da quantidade de inimigos a uma determinada pontuação, sendo a quantidade minima dos inimigos, a aguia (2) e o gavião (1) (FIGURA 75).

```
185
186         // AQUI E DEFINIDO OS INIMIGOS
187         for (int i = 1; i < 3; i++) {
188             this.inimigo1 = new Inimigos(this.inimigo1Regiao, 50f*i,
189             this.larguraTela, this.alturaTela, this.passarinho,this);
190             this.inimigo1.setScale(1.2f);
191             scene.attachChild(this.inimigo1);
192         }
193         for (int j = 1; j < 2; j++) {
194             this.inimigo2 = new Inimigos(this.inimigo2Regiao, 50f*j,
195             this.larguraTela, this.alturaTela, this.passarinho, this);
196             this.inimigo2.setScale(1.2f);
197             scene.attachChild(this.inimigo2);
198         }
199
200         // AQUI E DEFENIDO A PONTUACAO E O RECORDE
201         recorde = recordService.getRecorde();
202         textoPontuacao = new ChangeableText(12, 12, this.pontuacaoFont,
203         "PONTUACAO: " + pontos + " RECORDE: "+ recorde - "   ");
204         scene.attachChild(textoPontuacao);
205
206         return scene;
207     }
```

FIGURA 75 – CÓDIGO DEFININDO INIMIGOS

FONTE: O Autor (2015).

Os códigos do método atualizaPontuacao(), que atualiza os pontos, quando o jogador consegue desviar o passarinho de seus inimigos, são guardados no banco de dados, mas o recorde só é atualizado se o valor da pontuaçao passar do valor do recorde, quando é analisado pelo *this.recordService*.novoRecorde(recorde), assim que a pontuação passar do recorde, passa a ser atualizado junto ao banco de dados (FIGURA 76).

```
208
209     public void atualizaPontuacao(){
210         this.pontos++;
211         if(this.pontos > this.recorde){
212             this.recorde = this.pontos;
213             this.recordService.novoRecorde(recorde);
214         }
215         this.textoPontuacao.setText("PONTUACAO: " + pontos +
216         " RECORDE: "+ recorde);
217           if (this.pontos % 20 == 0) {
218               this.inimigo1 = new Inimigos(this.inimigo1Regiao,
219               50f*1,this.larguraTela, this.alturaTela, this.passarinho, this);
220               this.inimigo1.setScale(1.2f);
221               scene.attachChild(this.inimigo1);
222           }
223           if (this.pontos % 50 == 0) {
224               this.inimigo2 = new Inimigos(this.inimigo2Regiao,
225               50f*1,this.larguraTela, this.alturaTela, this.passarinho, this);
226               this.inimigo2.setScale(1.2f);
227               scene.attachChild(this.inimigo2);
228           }
229     }
```

FIGURA 76 – MÉTODO ATUALIZAPONTUACAO

FONTE: O Autor (2015).

Nos códigos do método *gameOver*() é chamada a intente da *game over* e, é mostrado o *score* (FIGURA 77).

```
231
232     public void gameOver(){
233         Bundle bundle = new Bundle();
234         bundle.putLong("pontos", pontos);
235         bundle.putLong("recorde", recorde);
236         Intent intent = new Intent(this,GameOverActivity.class);
237         intent.putExtras(bundle);
238         this.startActivity(intent);
239         this.finish();
240
241     }
```

FIGURA 77 – MÉTODO GAMEOVER

FONTE: O Autor (2015).

Através dos códigos do método *onCreateOptionMenu*(), o jogo recebe uma pausa, quando é acionado o pause do dispositivo pelo jogador, e ao liberar o dispositivo da pause, aparece um menu com as opções de continuar ou sair (*public boolean onOptionsItemSelected*(MenuItem item)) que verifica qual foi a escolha do jogador, e a executa (FIGURA 78).

```
242
243     @Override
244     public boolean onCreateOptionsMenu(Menu menu) {
245         menu.add("Continuar");
246         menu.add(Menu.NONE, 2, 2, "Sair");
247         return super.onCreateOptionsMenu(menu);
248     }
249
250     @Override
251     public boolean onOptionsItemSelected(MenuItem item) {
252         switch (item.getItemId()) {
253         case 2:
254             this.finish();
255             break;
256         default:
257             break;
258         }
259         return super.onOptionsItemSelected(item);
260     }
```

FIGURA 78 – MÉTODO ONCREATOPTIONMENU

FONTE: O Autor (2015).

Classe Passarinho: Esta classe possui uma assinatura herdada da *AndEngine AnimatedSprite*, sendo responsável por dar movimento às imagens PNG, chamadas de *Sprites* que representa virtualmente os movimentos dos objetos. É possivel verificar as variáveis que serão usadas, tais como a larguraTela e a auturaTela que são responsaveis pelos limites laterais, inferior e superior do espaço de movimento da *Sprite*, a aceleração que é a velocidade padrão de movimento da *Sprite*, uma variavel da *AndEngine* chamada de *PhysicsHander* a qual vai ser responsável por armazenar os dados do objeto como velocidade, posição, estado, etc..., e, ainda, uma variável (paraTras) que indicará qual o sentido do movimento (FIGURA 79).

```
6
7   public class Passarinho extends AnimatedSprite {
8       private final PhysicsHandler physicsHandler;
9       private int velocidade = 100;
10      private int larguraTela;
11      private int alturaTela;
12      private int aceleracao = 200;
13      private boolean paraTras = true;
14
```

FIGURA 79 – CLASSE PASSARINHO

FONTE: O Autor (2015).

Os códigos do método Passarinho(), são os principais da classe Passarinho, sendo o seu construtor, que através dele é criado uma instância do Passarinho que recebe parâmetros de orientação com a posição na tela do jogo (pX, pY), onde a imagem da *Sprite* será pintada (*pTiledTextureRegion*) e, os limites da tela que são definidos por (plarguraTela, palturaTela), criando um parâmetro de escala que divide o valor da tela do dispositivo por 1000f, para encontrar um tamanho apropriado para a tela e depois retorna todas as informações do objeto (FIGURA 80).

```
14
15      public Passarinho(float pX, float pY,
16              TiledTextureRegion pTiledTextureRegion,
17              int plarguraTela, int palturaTela) {
18
19          super(pX, pY, pTiledTextureRegion);
20          this.animacaoPadrao();
21          this.physicsHandler = new PhysicsHandler(this);
22          this.registerUpdateHandler(this.physicsHandler);
23          this.larguraTela = plarguraTela;
24          this.alturaTela = palturaTela;
25          Float escala = plarguraTela / 1000f;
26
27          if(escala <0){
28
29              escala = Cf;
30
31          }
32
33          this.setScale(escala);
34
35      }
36      public PhysicsHandler getPhysicsHandler() {
37
38          return physicsHandler;
39
40      }
```

FIGURA 80 – MÉTODO PASSARINHO

FONTE: O Autor (2015).

Os códigos do método *onManagedUpdate*(), são sobrescritos e são responsáveis por indicar ao aplicativo a velocidade padrão que foi indicada na variável velocidade, do movimento e da direção, que verifica o limite da tela, passando para o objeto, o qual não receberá mais a velocidade de movimento, no laço de verificação constante, atualizando o *physicsHandler* (FIGURA 81).

```
41
42      @Override
43      protected void onManagedUpdate(float pSecondsElapsed) {
44
45          if (this.mX < 0) {
46
47              this.physicsHandler.setVelocityX(this.velocidade);
48          } else if (this.mX + this.getHeight() > this.larguraTela) {
49
50              this.physicsHandler.setVelocityX(-this.velocidade);
51
52          }
53          if (this.mY < 0) {
54
55              this.physicsHandler.setVelocityY(this.velocidade);
56          } else if (this.mY + this.getHeight() > this.alturaTela) {
57
58              this.physicsHandler.setVelocityY(-this.velocidade);
59
60          }
61
62          super.onManagedUpdate(pSecondsElapsed);
63
64      }
```

FIGURA 81 – MÉTODO ONMANAGEDUPDATE

FONTE: O Autor (2015).

Há três métodos interligados na classe: (I) método mover(*float* x, *float* y) responsavel pela verificação do sentido em que o objeto esta se movimentando na tela de acordo com os comandos do jogador e implementa a visualização do movimento da *Sprite*; (II) animacaoVoltar(): - este método é invocado quando há movimento no eixo X, sendo o eixo horizontal negativo, indicando movimento para traz; (III) animacaoPadrão(), que é usado quando o objeto está parado ou em movimento para frente, sendo assim o passarinho terá movimento para frente e para trás (FIGURA 82).

```
65
66      public void mover(float x,float y){
67          if(x < 0){
68              this.animacaoVoltar();
69          }else{
70              this.animacaoPadrao();
71          }
72          this.physicsHandler.setVelocity(x * this.aceleracao,
73          y * this.aceleracao);
74      }
75      public void animacaoVoltar(){
76          if(!this.paraTras){
77              this.animate(new long[]{this.velocidade,
78              this.velocidade,this.velocidade},1,3,true);
79              this.paraTras = true;
80          }
81      }
82      public void animacaoPadrao(){
83          if(this.paraTras){
84              this.animate(new long[]{this.velocidade,
85              this.velocidade,this.velocidade},4,6,true);
86              this.paraTras = false;
87          }
88      }
```

FIGURA 82 – MÉTODO MOVER

FONTE: O Autor (2015).

Classe Inimigos: É a classe que possui uma assinatura herdada da *AndEngine AnimatedSprite*, que é responsavel por dar movimento as imagens PNG chamadas de *Sprites,* que representa virtualmente o movimento dos objetos. É possivel observar as variáveis que serão usadas como a velocidadeX, a velocidadeY, que são responsáveis pela velocidade de movimento padrão da *Sprite* em função do eixo X e Y; a larguraTela e a alturaTela que são responsáveis pelos limites laterais, inferior e superior do espaço de movimento da *Sprite;* a variável *AndEngine PhysicsHander* que é responsável por armazenar os dados do objeto, como velocidade, posição, estado e etc.. e, também, uma variável Randon que gera valores aleatórios para o movimento dos Inimigos e, por fim, uma variável do tipo passarinho que será seu alvo no jogo e a *activity* onde estarão acontecendo as interações (FIGURA 83).

```
 8
 9  public class Inimigos extends AnimatedSprite {
10
11      private PhysicsHandler physicsHandler;
12      private float velocidadeX;
13      private float velocidadeY;
14      private Random random;
15      private int larguraTela;
16      private int alturaTela;
17      private Passarinho passarinho;
18      private PassarinhoEmFuga activity;
```

FIGURA 83 – CLASSE INIMIGOS

FONTE: O Autor (2015).

O método Inimigos() é o principal método da classe Inimigos. No construtor é criada uma instância de Inimigos que recebe os parâmetros de orientação, uma velocidade (pVelocidade), onde a imagem da *Sprite* será pintada com (*pTiledTextureRegion*), os limites da tela definidos por plarguraTela. E palturaTela, o objeto de interação ppassarinho e o ambiente *pactivity*, que é criado um parâmetro de escala que divide o valor da tela do dispositivo por 1000f para encontrar um tamanho apropriado para a tela do jogo e, por fim retorna essas informações do objeto, e os objetos surgem no fim da tela são iguais ao tamanho dela (FIGURA 84).

```
19
20      public Inimigos(TiledTextureRegion pTiledTextureRegion, float
21      pVelocidade,
22              int plargura, int paltura, Passarinho ppassarinho,
23              PassarinhoEmFuga pactivity) {
24          super(-1, -1, pTiledTextureRegion);
25          this.physicsHandler = new PhysicsHandler(this);
26          this.registerUpdateHandler(this.physicsHandler);
27          this.randon = new Random();
28          this.velocidadeX = getVelocidadeX();
29          this.velocidadeY = getVelocidadeY();
30          this.larguraTela = plargura;
31          this.alturaTela = paltura;
32          this.passarinho = ppassarinho;
33          this.activity = pactivity;
34
35          this.physicsHandler.setVelocity(-this.velocidadeX,
36          this.velocidadeY);
37          this.setPosition(this.larguraTela, this.getPosicaoInicial());
38          Float escala = larguraTela / 1000f;
39          if(escala <0){
40              escala = 0f;
41          }
42          this.setScale(escala);
43      }
```

FIGURA 84 – MÉTODO INIMIGOS

FONTE: O Autor (2015).

Os códigos do método *onManagedUpdate*(), são responsaveis por passar ao aplicativo a velocidade, com um valor aleatório que é gerado por *private float getVelocidadeX*() e por *private float getVelocidadeY*(), utilizados nas variáveis velocidadeX e velocidadeY, em relação ao movimento e a direção. Juntamente, é verificado que chegando no limite da tela, a pontuação é incrementada, e o laço de vereficação é constante, atualizando o *physicsHandler* para evitar desperdício de objetos, que não são destruídos ao chegar no final da tela e, sim, reutilizados novamente permanecendo em um ciclo, na *activity* (FIGURA 85).

```
56
57      @Override
58      protected void onManagedUpdate(float pSecondsElapsed) {
59          if (this.mY < 0) {
60              this.physicsHandler.setVelocity(-this.velocidadeX,-this.velocidadeY);
61          } else if (this.mY + this.getHeight() > this.alturaTela) {
62              this.physicsHandler.setVelocity(-this.velocidadeX,-this.velocidadeY);
63          }
64          if (this.mX < 0) {
65              this.setPosition(this.larguraTela, this.getPosicaoInicial());
66              this.velocidadeX =  getVelocidadeX();
67              this.velocidadeY = getVelocidadeY();
68              this.physicsHandler.setVelocity(-this.velocidadeX, this.velocidadeY);
69              this.activity.atualizaPontuacao();
70          }
71          // AQUI DEFINE A COLISAO = GAMEOVER
72          if(this.collidesWith(this.passarinho)){
73              this.activity.gameOver();
74          }
75          super.onManagedUpdate(pSecondsElapsed);
76      }
```

FIGURA 85 – MÉTODO ONMANAGEDUPDATE

FONTE: O Autor (2015).

Os códigos do método getVelocidadeY() são responsáveis por manter uma dinâmica aleatória no movimento vertical dos objetos Inimigos (FIGURA 86).

```
44
45      private float getVelocidadeY() {
46
47          return (this.random.nextBoolean() ? this.random.nextFloat()
48          * 50 : this.random.nextFloat() * (-50));
49
50      }
```

FIGURA 86 – MÉTODO GETVELOCIDADEY

FONTE: O Autor (2015).

Os códigos do método getVelocidadeX() são responsáveis por manter uma dinâmica aleatória no movimento horizontal dos objetos Inimigos (FIGURA 87).

```
51
52      private float getVelocidadeX(){
53
54          return (this.randon.nextInt(2)+1) * 50f;
55
56      }
```

FIGURA 87 – MÉTODO GETVELOCIDADEX

FONTE: O Autor (2015).

Por fim os códigos do método getPosicaoInicial() são responsáveis por voltar à posição das dinâmicas aleatória, dos movimentos verticais e horizontais dos objetos Inimigos à posição inicial (FIGURA 88).

```
57
58      public float getPosicaoInicial(){
59
60          return this.randon.nextFloat() * this.alturaTela;
61
62      }
```

FIGURA 88 – MÉTODO GETPOSICAOINICIAL

FONTE: O Autor (2015).

CONSIDERAÇÕES FINAIS

O objetivo deste trabalho de conclusão de curso foi demonstrar a realização do desenvolvimento de um jogo 2D para a plataforma Android.

Desta forma observa-se que para a escolha do desenvolvimento de um projeto é necessário optar por uma solução que traga melhor benefício, em termos de eficiência, custo e tempo de desenvolvimento para a sua construção. Foi através deste conhecimento que foi usada a plataforma Android SDK.

Para a conscientização do conhecimento do Sistema Android, houve a necessidade de expor um pouco sobre o histórico, definição, arquitetura da plataforma e sobre as ferramentas utilizadas tais como: Dia, Gimp, Eclipse, Android SDK, Java JDK, SQLite, ADT plugin e a AndEgine (GLS2).

Foi verificado que nos testes de funcionalidade o jogo apresentou um bom desempenho de performance e os botões como também as telas, ficaram dentro das expectativas de configuração. Ainda foi observado que falta implementar alguns ajustes e com possíveis melhorias futuras, é possivel tornar este trabalho melhor e mais completo.

Ao fim deste trabalho, advindo do desenvolvimento deste aplicativo (Jogo), pretende-se disseminar e incentivar o estudo e a criação de jogos às pessoas que desejam iniciar no desenvolvimento de jogos, demonstrando o conhecimento obtido no decorrer deste curso.

REFERÊNCIAS

ANDROID ARQUITECTURE. Disponível em: http://code.google.com/android/what-is-android.html. Acesso em 22 de abril de 2015.

ANDROID. Disponível em: http://www.android.com/about/ Acesso em 25 de abril de 2015.

ABLESON, W.; COLLINS C.; SEN R. **Unlocking Android – A Developer's Guide.** Manning, 2009.

ABRAGAMES. **Plano Diretor da Promoção da Indústria de Desenvolvimento de Jogos Eletrônicos no Brasil: Diretrizes Básicas.** 2004. Disponível em: http://www.abragames.br. Acesso: 26 maio de 2015.

AQUINO, J. F. S. **Plataforma de Desenvolvimento para Dispositivos Móveis**. 14f. Trabalho de Pós Graduação.(Especialização em Introdução a Computação Móvel). Pontifícia Universidade Católica do Rio de Janeiro – PUC – Rio, Rio de Janeiro, 2007.

ARANHA, G. O **processo de consolidação dos jogos eletrônicos como instrumento de comunicação e de construção de conhecimento.** 2004. Ciências & Cognição; Ano 01, Vol 03, p. 21-62.

AMORIN, A. **A origem dos jogos eletrônicos.** USP, 2006.

BATTAIOLA, A. L.; DOMINGUES, R. d. G. DILZA, B. F. **Desenvolvimento de Jogos em Computadores e Celulares.** Departamento de Computação - UFSCar. 2004.

BONGIOLO, C. E. F.; BRAGA, E. R.; SILVEIRA, M. S. **Subindo e Escorregando: Jogo para introdução do conceito de adição de números inteiros.** IV Congresso RIBIE, Brasilia, 1998.

BORDIN, M.V. **Introdução a Arquitetura Android.** Setrem, Três de Maio, Rio Grande do Sul, 2012. Disponível em:

http://sites.setrem.com.br/stin/2012/anais/Maycon.pdf. Acesso em 23 de maio de 2015.

COUTINHO, D. MOBILEGAMER. **Melhores jogos para celular de 2014.** Disponível em: http://www.mobilegamer.com.br/2015/01/melhores-jogos-para-celular-de-2014-android-ios-e-windows-phone.html. Acesso em 05 de junho de 2015.

DANTE, S. R. S.; BARONE, A. C. **Jogos Educativos Computadorizados Utilizando a Abordagem de Algoritmos Genéticos.** Universidade Federal do Rio Grande do Sul - UFRGS - Curso de Curso de Pós-Graduação. 1998.

DEVELOPERS. **ADT Plugin Release Notes.** Disponível em: http://developer.android.com/tools/sdk/eclipse-adt.html. Acesso em 19 de maio de 2015.

DEVELOPER JUNIOR. **Tecnologia JAVA!.** Disponivel em: https://devjr.wordpress.com/tecnologia-java/. Acesso em 24 de maio de 2015.

DEVMEDIA. **Artigo WebMobile 18 – Android: um novo paradigma de desenvolvimento móvel.** Disponivel em: http://www.devmedia.com.br/artigo-webmobile-18-android-um-novo-paradigma-de-desenvolvimento-movel/9350. Acesso em 26 de maio de 2015.

ECLIPSE FOUNDATION. Disponível em: http://www.eclipse.org. Acesso em 24 de abril de 2015.

FERNANDES, A. **A Comunicação Medida por Interfaces Digitais: a interação humana com os jogos digitais em celulares.** 232f. Tese (Doutorado em Comunicação Social). Programa de Pós-Graduação em Comunicação Social. Universidade Metodista de São Paulo, São Bernardo do Campo, 2007.

GNOME.ORG. **Dia.** Disponível em: https://wiki.gnome.org/Apps/Dia. Acesso em 26 de maio de 2015.

GOMES, R. C.: FERNANDES, J. A. R.; FERREIRA, V. C. **Sistema Operacional Android.** 29f. Trabalho de Graduação(Engenharia em Telecomunicação). Sistema de Computação. Universidade Federal Fluminense, 2012.

JOHAN, 2003, Johan, **Dicky. Take Control of Your Properties**, Eclipse Technical Articles, 2003.

LINUX. **O Gimp – GNU Image manipulation program.** Disponivel em: http://diegocarol.blogspot.com.br/2009/11/o-gimp-gnu-image-manipulation-program-e.html. Acesso em 24 de maio de 2015.

MARTINS, G.R.; RAFALSKI, J. P.; RESENDE, R. F. **Protótipo de rede tolerante a falhas e desconexões utilizando comunicação via Bluetooth entre dispositivos movies Android.** 70f. Trabalho de Graduação(Bacharel em Ciência da Computação). Curso de Ciência da Computação. Centro Universitario Vla Velha, Vila Velha, 2011.

ORACLE. **JAVA SE Development Kit 7 Downloads.** Disponivel em: http://www.oracle.com/technetwork/pt/java/javase/downloads/jdk7-downloads-1880260.html. Acesso em 24 de maio de 2015.

KORJENIOSKI, M. **Desenvolvimento de Jogos 2D com Android.** 42f. Trabalho de Pós Graduação. (Especialização em Tecnologia Java). Universidade Tecnologica Federal do Paraná, Curitiba, 2011.

SUN MICROSYSTEM. **JAVA TECHNOLOGY**. Disponível em: http://java.sun.com Acesso em 15 de abril de 2015.

SOUZA, M. V. O. ROCHA, V. M. **Um estudo sobre o desenvolvimento de jogos eletrônicos.** Unipê, João Pessoa. Dezembro/2005. 123 páginas.

STANG, B. **Game Engines features and Possibilities by Bendik Stang - IMM DTU 2003.** Institute of Informatics and Mathematical Modeling - The Technical University of Denmark.2003.

VOGELL, Lars. **Android SQLite Database: tutorial**. 2011. Disponivel em: HTTP://www.vogella.de/articles/AndroidSQLite/article.html. Acesso em 20 de abril de 2015.

GONÇALVES, Eduardo Corrêa. **SQLite, muito prazer!.** 2011. Disponível em: http://www.devmedia.com.br/sqlite-muito-prazer/7100. Acesso em 16 de abril de 2015.

LECHETA, Ricardo R. **Google Android:** aprenda a criar aplicações para dispositivos móveis com Android SDK. 2. ed. São Paulo: Novatec, 2010.

LECHETA, Ricardo R. **Google Android:** aprenda a criar aplicações para dispositivos móveis com Android SDK. 3. ed. São Paulo: Novatec, 2013.

LEMAY, Laura, **Aprenda em 21 Java 2.** Rio de Janeiro: Editora Campus, 1999.

LIMA, L. A.; NETO, W.P.F.; VINICIUS, G.; FECHINE, J. M. **Eclipse Toos – Ferramenta para auxílio à composição dinâmica de software.** 11f. DSC/UFCG, Campina Grande, PB, 2015.

BOOCH, Grady. **UML: guia do usuário.** Rio de Janeiro: Elsevier, 2006.

MEDEIROS E. S., **Desenvolvendo Software com UML 2.0: definitivo.** São Paulo: Pearson Makron Books, 2004.

MOTA, Kleber, **Instalando e atualizando o plugin ADT no Eclipse.** Disponível em: http://www.klebermota.eti.br/2010/02/08/installing-and-updating-adt/. Acesso em 19 de maio de 2015.

LEE, V.; SCHNEIDER, H.; SCHELL, R., **Aplicações móveis:** arquitetura, projeto e desenvolvimento. São Paulo: Pearson, 2005.

PETTINGA, V. C. a. **O Jogador-Autor: A Comunicação Ativa dos Jogos para Computador.** Universidade Federal da Bahia. 2002.

PEREIRA, G. A. **Projeto e desenvolvimento de jogos computacionais.** 153f Trabalho de Graduação (Bacharel em Ciência da Computação). Universidade do Estado de Santa Catarina – UDESC. Joinville, 2006.

SQLITE. Disponível em: http://www.sqlite.org/. Acesso em 25 de maio de 2015.