UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

RAFAEL DA SILVA PIMENTA

DESENVOLVIMENTO DE JOGOS PARA A PLATAFORMA ANDROID UTILIZANDO CORONA SDK

QUIXADÁ
2013

RAFAEL DA SILVA PIMENTA

DESENVOLVIMENTO DE JOGOS PARA A PLATAFORMA ANDROID UTILIZANDO CORONA SDK

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Graduação em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Orientador Prof. Jefferson de Carvalho Silva

QUIXADÁ
2013

Dados Internacionais de Catalogação na
Publicação
Universidade Federal do
Ceará
Biblioteca do Campus de
Quixadá

P697d Pimenta, Rafael da Silva
Desenvolvimento de jogos para a plataforma android utilizando corona SDK / Rafael da Silva Pimenta. – 2013.
40 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2013.
Orientação: Prof. Msc. Jefferson de Carvalho da Silva
Área de concentração: Computação

1. Android 2. Jogos 3. Linguagens de programação (computadores) I. Título.

CDD 005.3

RAFAEL DA SILVA PIMENTA

DESENVOLVIMENTO DE JOGOS PARA A PLATAFORMA ANDROID UTILIZANDO CORONA SDK

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Graduação em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Aprovado em: _____ / julho / 2013.

BANCA EXAMINADORA

_____
Prof. Jefferson de Carvalho Silva (Orientador)
Universidade Federal do Ceará-UFC

_____
Prof. João Marcelo Uchôa de Alencar
Universidade Federal do Ceará-UFC

_____
Prof. Marcio Espínola Freire Maia
Universidade Federal do Ceará-UFC

À minha família...

RESUMO

Este trabalho irá abordar assuntos relevantes para o desenvolvimento de jogos, como ciclo de desenvolvimento, mercado e estrutura do sistema operacional Android. O trabalho tem como objetivo realizar um estudo sobre a ferramenta de desenvolvimento Corona SDK e seu uso para o desenvolvimento de jogos para plataforma Android, descrevendo seus componentes e estrutura, destacando as vantagens e desvantagens. Depois será realizado um estudo de caso onde um jogo simples será desenvolvido para uma análise posterior. O estudo de caso utilizará as informações obtidas e requisitos previamente estabelecidos, como valor de licença, ferramentas disponíveis, documentação, funcionalidades oferecidas e linguagem de programação utilizada.

Palavras chave: Corona SDK. Android. Jogos. Dispositivos Móveis.

ABSTRACT

This work will address issues relevant to the development of games, such as the development cycle, market and structure of the Android operating system. The work aims to conduct a study on the development tool Corona SDK and its use to develop games for the Android platform, describing its components and structure, highlighting the advantages and disadvantages. After will be carried a study of the case where a simple game will be developed for an analysis posterior. The case study will use the information obtained and requirements previously established, as the value of license, available tools, documentation, and functionality offered programming language.

Keywords: Corona SDK. Android. Games. Mobile Devices.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Dispositivos móveis como telefones celulares, *smartphones* e *tablets* apresentaram constante evolução principalmente em seu processamento, memória RAM e armazenamento. Com o avanço do hardware veio também a sofisticação dos sistemas operacionais, como *Windows Mobile* da *Microsoft*, IOS da *Apple*, e *Android* do Google, tendo esse último se tornado o mais popular recentemente, segundo pesquisa da empresa *Gartner* (2012). Estes avanços permitiram que os dispositivos móveis conseguissem suportar a execução de softwares cada vez mais complexos, despertando a atenção da indústria de desenvolvimento de software e especialmente da indústria de jogos.

Jogos eletrônicos deixaram de ser uma simples brincadeira e se transformaram em superproduções envolvendo equipes de centenas de profissionais altamente qualificados e utilizando o que há de mais avançado em técnicas de programação. Assim, os desenvolvedores voltaram sua atenção para os dispositivos móveis, devido a sua popularidade e evolução do hardware, o que favorece o desenvolvimento de jogos mais elaborados e consequentemente mais atrativos aos usuários.

De acordo com Nogueira (2004), desenvolver jogos para dispositivos móveis envolve muitos fatores, devido à grande variedade de modelos e as diferentes características de hardware e sistema operacional que possuem. Nesses casos deve ser usado um game *engine* (motor de jogo), que serve como uma camada de abstração entre o sistema operacional e o hardware, facilitando assim o processo de desenvolvimento para vários dispositivos diferentes. Mas ainda persiste um problema, que seria a escolha da *engine* ideal dentre tantas que existem. Muitas delas possuem características bem peculiares e algumas são mais adequadas para projetos com objetivos específicos.

Existem trabalhos e livros que abordam o tema de desenvolvimento de jogos, mas é difícil encontrar trabalhos relacionados com *game engines* para *Android*. Nogueira (2004), por exemplo, faz um estudo comparativo entre as principais plataformas de desenvolvimento de jogos para celulares da época, mas, por ser um trabalho do final de 2004 não se aplica as tecnologias atuais. Um trabalho realizado mais recentemente, e com foco em jogos para Android foi feito por Noronha (2010), que descreve o desenvolvimento de um jogo, com foco em engenharia de software e na utilização do SDK, mas sem a utilização de uma *engine* específica para jogos.

Este trabalho tem como objetivos realizar um estudo sobre o desenvolvimento de jogos para dispositivos móveis com foco no Corona SDK voltado para o desenvolvimento para o sistema operacional Android, destacando suas vantagens e desvantagens. Além disso, iremos implementar um estudo de caso para posterior análise e conclusão sobre as funcionalidades facilidade de uso e recursos oferecidos para proporcionar melhores condições de desenvolvimento e se oferece os requisitos necessários para o desenvolvimento de jogos para Android.

2 REVISÃO BIBLIOGRÁFICA

Jogos digitais são aplicações compostas por sons, imagens, enredo e desafios. Um tipo atrativo de mídia principalmente pela imersão e interatividade que podem oferecer aos jogadores. Assim como qualquer aplicativo o desenvolvimento de jogos passa por vários passos até ser concluído.

2.1 Jogos Digitais

O primeiro passo para a criação de um jogo é a sua concepção, a ideia básica, a motivação para seu desenvolvimento, essa motivação pode ter as mais variadas origens, seja ela por necessidade de uma empresa em lucrar com o jogo ou para realização pessoal. (Junior, 2002).

Ainda na concepção será escolhido o tipo de jogo, os jogos podem ser classificados primeiramente em dois tipos, jogos 3D e 2D. Jogos 3D são jogos onde o jogador vê tudo em perspectiva, esse tipo de jogo proporciona uma maior imersão ao jogador; Jogos 2D são utilizadas figuras e imagens para representar personagens e cenários, diferente de jogos 3D que utilizam modelos vetoriais. (Battaiola, 2001 *apud* Zimmermann, 2006, p.33).

Outra classificação atribuída a jogos é em relação à quantidade de jogadores: os jogos podem ser *multiplayer*, onde dois ou mais jogadores podem jogar simultaneamente, seja *off-line* ou online, jogando em um time ou contra os outros usuários; e pode ser *singleplayer,* onde o jogador joga sozinho. Além dos tipos já citados os jogos também podem ser classificados de acordo com suas características, como jogos de aventura, estratégia, esporte, RPG, simulação, luta entre outros.

Outro passo importante é a escolha do público alvo, nos dias atuais a idade média dos principais consumidores de games varia de 16 a 25 anos, mostrando que games não é “coisa de criança”. Entre esses consumidores existem diferentes tipos de jogadores, os ditos *hardcore,* que gostam de jogos complexos com alto grau de dificuldade, e os jogadores casuais que preferem jogos mais simples que priorizam a diversão. (Junior, 2002).

Além dos já citados outro fator importante no processo de desenvolvimento de jogos é a escolha da plataforma para qual o jogo será desenvolvido. Apesar de existirem jogos que estão disponíveis para várias plataformas, na maioria das vezes eles são desenvolvidos para uma plataforma primeiramente e só depois são adaptados para outras. Os tipos de plataformas são muitas, como consoles de mesa, portáteis, computadores e dispositivos móveis como *tablets* e *smartphones.*

O desenvolvimento de jogos aborda várias áreas de conhecimento sendo assim uma área multidisciplinar dentro da computação, utilizando conhecimentos das áreas de matemática, física, geometria, análise de sistema, artes gráficas dentre outras. Segundo Junior (2002) o desenvolvimento de jogos é um processo quase artístico ou artesanal.

De acordo com Amazonas (2007) e Perucia (2005), para desenvolver jogos complexos e de qualidade é necessário uma equipe especializada e diversificada, esses profissionais são: programadores, artistas, projetistas de níveis/fases, projetistas de jogos, planejador de *software,* arquiteto-chefe, gerente de projeto e testadores.

2.2 Ciclo de Desenvolvimento

O processo de desenvolvimento de jogos pode ser semelhante em alguns pontos ao desenvolvimento convencional de software, utilizando técnicas de engenharia de software, como análise de projeto, implementação e testes. Por outro lado a natureza peculiar dos jogos os torna diferentes dos demais softwares. Isso levou ao desenvolvimento de processos específicos para games que podem ser usados em conjunto com os processos tradicionais.

Segundo Perucia (2005), o ciclo de desenvolvimento de jogos possui as seguintes etapas:

- *Braimstorming* consiste em uma reunião com toda a equipe de desenvolvimento para discutir sobre as características do projeto, como o tipo de jogo, público-alvo e as especificações técnicas.
- *Game Design*: são descritas as características informações cruciais do projeto como: jogabilidade, golpes, personagens, inimigos, fases e outras informações importantes.
- *Design Document* é um documento onde as informações dos passos anteriores são descritas com mais detalhes, onde todas as características do jogo são abordas.
- Cronograma: com as etapas de desenvolvimento já definidas, o gerente de projeto já tem uma noção do tamanho do projeto e pode fazer uma estimativa de quanto tempo ele irá durar.

- *Level Design*: descreve todas as missões e desafios que o jogador precisa completar para prosseguir no jogo. Este documento serve de referencia para os artistas que irão desenvolver os cenários e o visual.
- *Beta*: são lançadas várias versões *alphas* até chegar à versão *beta*, que é uma versão completa do jogo, com todas as fazes. Terminada a versão *beta* começam os testes para encontrar erros, popularmente chamados de *bugs*, bem como coletar informações que possam melhorar o jogo.
- Versão Final: é o produto final pronto para ser vendido, é o resultado do processo de correção de erros e refinamentos ocorridos na fase *beta*.

A Figura 1 mostra a ordem das etapas de desenvolvimento de um jogo.

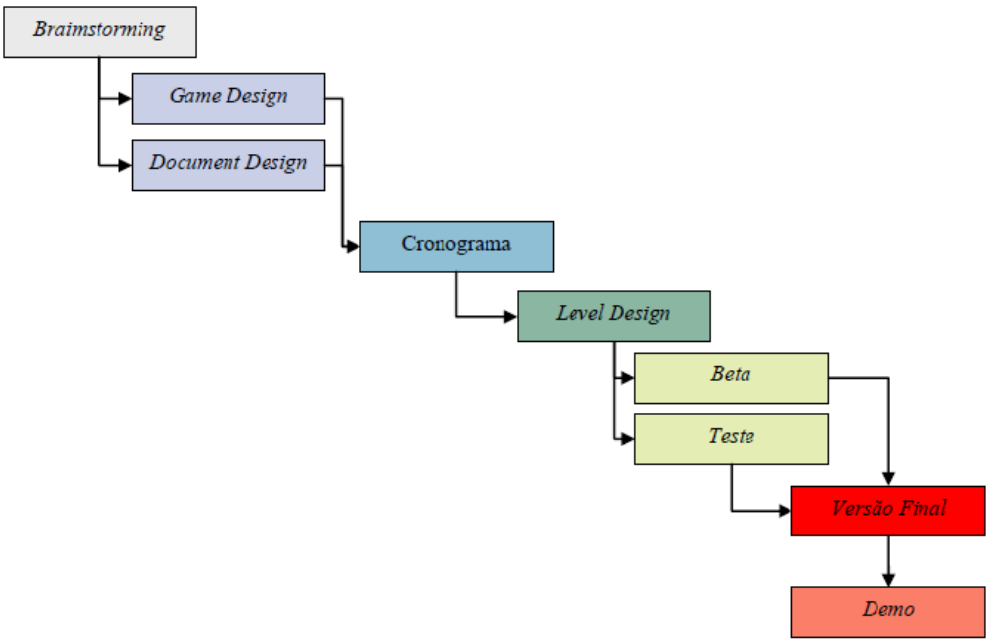

Figura 1: ordem das etapas de desenvolvimento de um jogo – Fonte: Amazonas (2007).

Em projetos pequenos o ciclo de desenvolvimento é reduzido, em alguns casos apenas o *Design Document* é suficiente para que o projeto seja executado de forma satisfatória (Perúcia, 2005). Esse trabalho tem como um de seus objetivos o desenvolvimento de um jogo simples, com a finalidade de demonstrar as principais funcionalidades e características das ferramentas em estudo, tornando-se desnecessário a utilização de um ciclo de desenvolvimento ou qualquer tipo de documentação. O conteúdo desse tópico tem como finalidade servir como uma fonte de pesquisa para desenvolvedores com interesse em aprofundar seus conhecimentos no desenvolvimento de jogos.

2.3 Mercado de jogos no Brasil

Em uma pesquisa realizada pela *Newzoo* (2011), mostrou que o número de jogadores ativos no Brasil é de 35 milhões, sendo que 47% gastam dinheiro com jogos. A Figura 1 mostra uma estimativa de quanto foi gasto em jogos no Brasil, fazendo uma distinção por plataforma.

Figura 2 – Dinheiro gasto com jogos - Fonte: Newzoo (2011)

Na pesquisa da *Newzoo* (2011), os tipos de plataformas foram classificados da seguinte forma:

- Jogos em redes sociais: jogos disponibilizados em redes sociais como Facebook e Orkut.
- Jogos casuais: são jogos hospedados em site de jogos que podem ser jogados diretamente no site ou baixados.
- Jogos em dispositivos móveis: celulares, *smartphones, Ipod Touch e tablets*.
- Jogos MMO (*Massive Multiplayer Online Role Playing*): traduzindo para português, de Interpretação de Personagem On-line e em Massa Para múltiplos Jogadores.
- Jogos para consoles (incluindo portáteis): *Playstation* 1/2/3, PSP, Xbox, Xbox360, *Nintendo* Wii, Ds, Dsi, 3Ds entre outros.
- Jogos PC/Mac em embalagens: jogos comprados em mídia física CD/DVD e instalados em PC, Laptop ou Mac.

- Jogos de PC e Mac por download: jogos que são baixados para poder serem instalados no PC, Laptop ou Mac (MMOs, jogos casuais pagos em sites não entram nessa categoria).

A Figura 3 mostra o número de jogadores por plataforma.

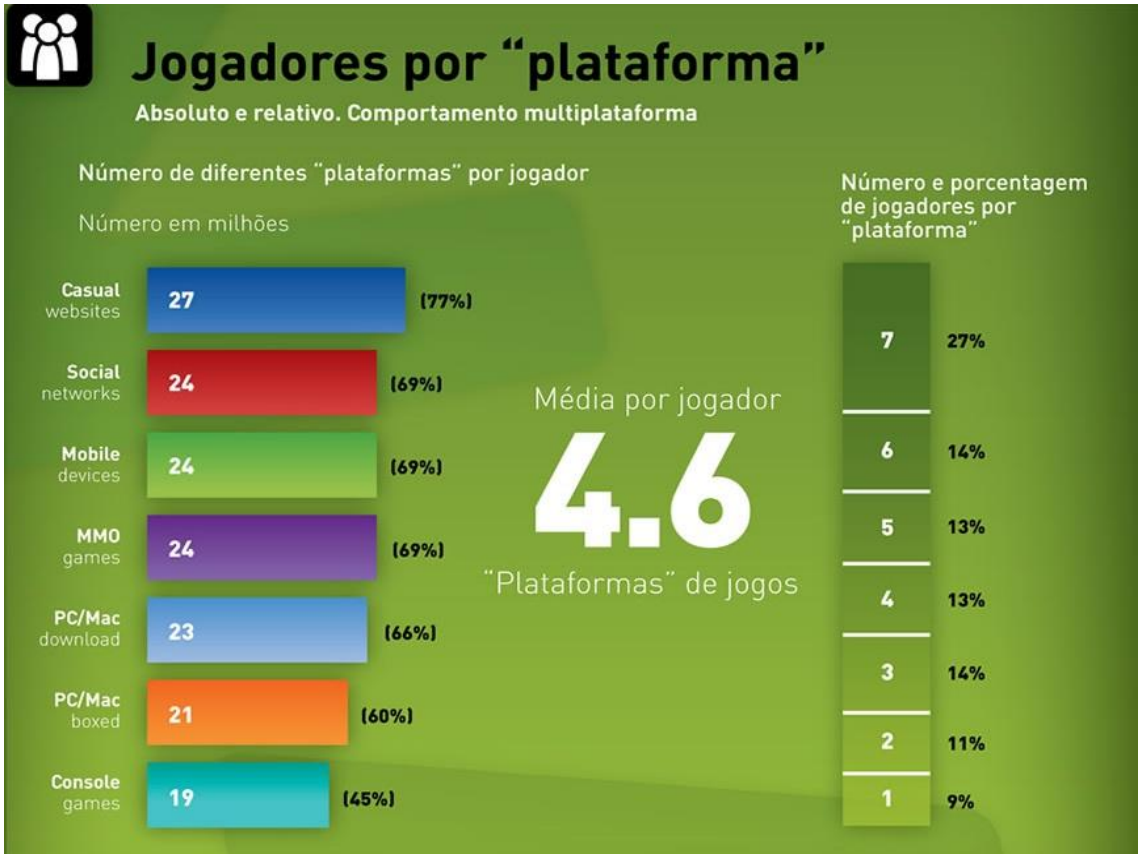

Figura 3 - Jogadores por plataforma – Fonte: Newzoo (2011)

Através das informações obtidas na pesquisa é possível observar a importância dos jogos na indústria de softwares e entretenimento, principalmente dos jogos para dispositivos móveis, que será o foco desse trabalho e que apresentou maior número de jogadores que plataformas tradicionais como consoles de mesa e computadores. Daí surge à necessidade de se conhecer melhor a produção desse tipo jogo e quais ferramentas são usadas para tal atividade.

2.4 Engine

No desenvolvimento de jogos uma das partes mais importantes é a definição da arquitetura. Sua finalidade é proporcionar uma visão geral do projeto, para que o desenvolvimento possa fluir da melhor maneira possível. A especificação da arquitetura é definida no *design document* e está dividida em duas partes: abstração de hardware e abstração de jogo. (Moreira, 2011 e Perucia, 2005).

A figura abaixo ilustra a organização da arquitetura de um jogo em camadas.

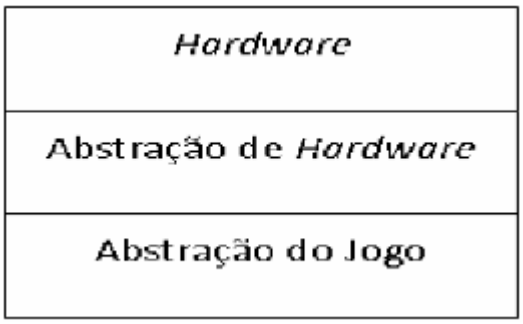

Figura 4 – Organização da arquitetura de um jogo em camadas – Fonte: Moreira (2011).

A função da camada de abstração de hardware é facilitar o uso de recursos de entrada/saída, vídeo, áudio, rede, entre outros. A camada de abstração de jogo é responsável pelo gerenciamento dos subsistemas.

As ferramentas de desenvolvimento responsáveis por oferecer esses recursos de abstração são chamadas de *game engine* (motor de jogo), trata-se de um programa ou conjunto de bibliotecas com finalidade de facilitar e abstrair o desenvolvimento de jogos eletrônicos. Dentro de um motor de jogo existem vários subsistemas, dentre eles os principais são: motor gráfico, motor de física, motor de som, motor de inteligência artificial. (Perucia, 2005).

O motor gráfico é o responsável pela renderização das imagens e resposta visual interativa. Ele é um gerador de imagens a partir de um modelo, O motor de Física simula física real como gravidade, elasticidade, fricção, bem como calcula movimento e colisão entre corpos. Motor de som é encarregado da produção dos sons e efeitos sonoros. E por ultimo o motor de inteligência artificial, responsável por definir o comportamento dos personagens não jogáveis. (Amazonas 2007).

Além das já citadas, existem ainda outras funcionalidades oferecidas pelos motores de jogo, como por exemplo, permitir o desenvolvimento de vários jogos usando a mesma ferramenta; capacidade de reaproveitar código de projetos anteriores; abstração de hardware e de sistema operacional; conexão com a Internet; suporte à animação.

A estrutura de um motor de jogo pode ser observada na Figura 5.

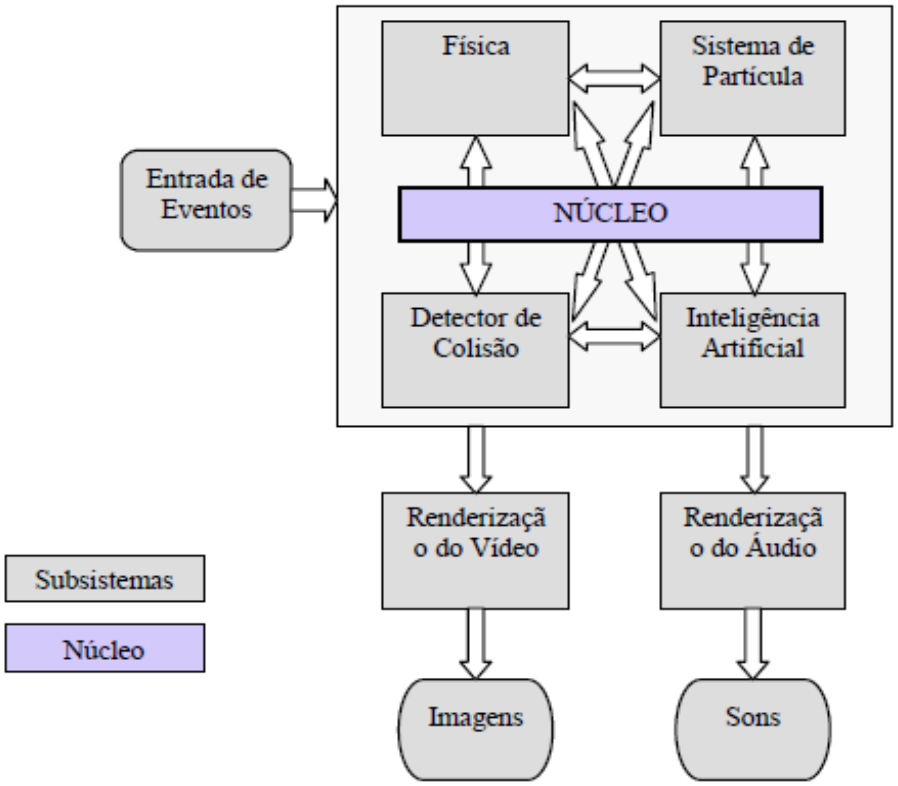

Figura 5: estrutura de um motor de jogo – Fonte: Gamasutra (2005 *apud* Amazonas, 2007, p.16).

Na figura a entrada de eventos representa os três estados possíveis para um botão: pressionado, liberado ou constantemente pressionado, as ações geradas são decididas pelo desenvolvedor. Na segunda camada se encontra o núcleo, principal componente e responsável por gerenciar a *engine*, está diretamente relacionado com os subsistemas. (Amazonas, 2007).

2.5 Android

Android é um sistema operacional de código aberto voltado para dispositivos móveis, foi desenvolvido pelo Google e mantido pelo *Open Handset Alliance* (OHA), o sistema funciona sobre um núcleo baseado no Linux e a maioria dos aplicativos é programado em Java o que facilita o processo de desenvolvimento, pois a linguagem Java é amplamente difundida entre os programadores.

Segundo *Android Developers* (2012), o Android se tornou rapidamente o sistema operacional para dispositivos móveis que mais cresce, principalmente pela abertura do código fonte do mesmo, que o tornou popular entre consumidores e desenvolvedores, impulsionando o crescimento do desenvolvimento e consumo de aplicativos, representando um contingente de

aproximadamente 200 milhões de aparelhos com o sistema operacional, e mais de 550.000 novos dispositivos ativados diariamente.

De acordo com Pereira e Silva (2009, *apud* LOPEZ, 2011, p.13) a quantidade de telefones celulares vem aumentando massivamente. Atualmente o celular é o produto de consumo considerado como mais utilizado no mundo, cuja quantidade de aparelho corresponde à metade da população do mundo, cerca de 3,3 bilhões (dados extraídos de 2007). Estima-se que até o final de 2013, esse número poderá chegar a 5,6 bilhões.

Diante do exposto, percebem-se inúmeras possibilidades de negócio, em especial no desenvolvimento de software e jogos voltados a atender essa demanda, assim, o desenvolvimento de jogos para plataforma Android desponta como um dos mais atrativos ramos de mercado, já que está voltado principalmente para dispositivos móveis, os quais podem ser considerados a nova moda de consumo de tecnologias voltadas ao entretenimento.

Segundo (LOPEZ, 2011), um desenvolvedor que queira programar em Android, seja aplicações comerciais, entretenimento ou jogos, deve conhecer bem a estrutura dessa plataforma.

Para efeito ilustrativo, a figura abaixo resume os mais importantes componentes da plataforma Android.

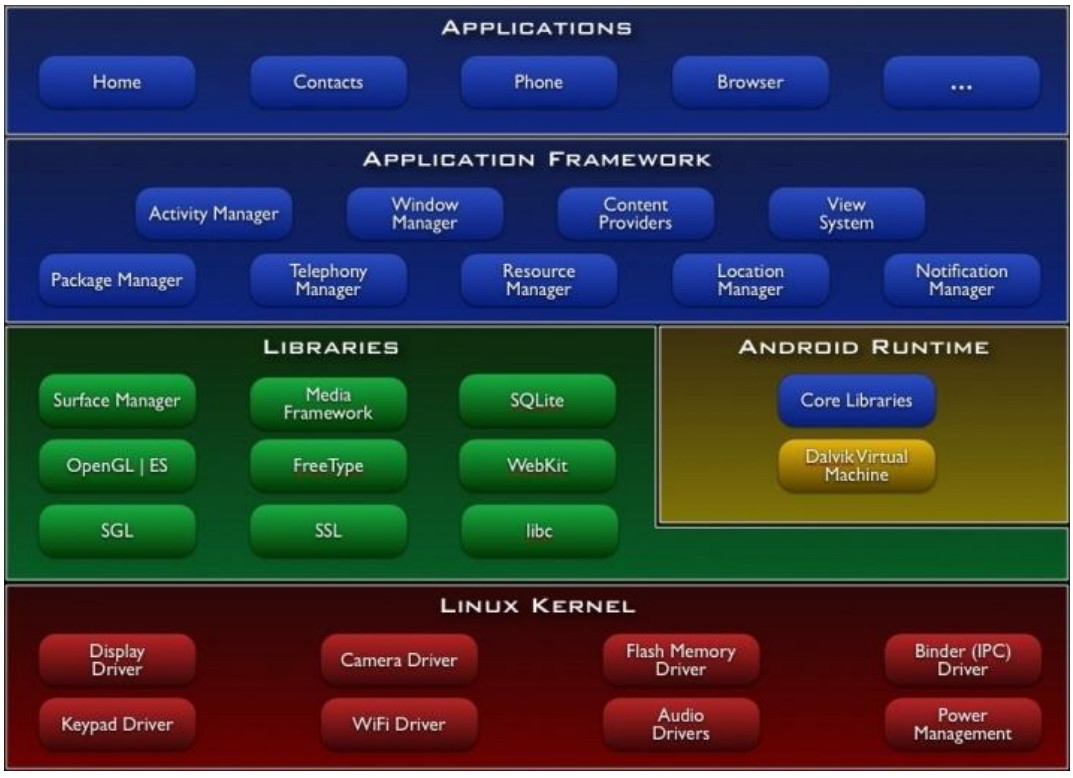

Figura 6 – Arquitetura Geral do Android - Fonte: Google Inc (2012)

No que tange às aplicações, o Android, de acordo com Google Inc (2012), caracteriza-se como um sistema operacional que possui um conjunto de aplicações nativas, sendo as

aplicações destinadas à cliente de e-mail, programas de SMS, agenda, mapas, navegador e contatos, as principais. Todas desenvolvidas em Java.

Outro aspecto relevante no Android são as bibliotecas, das quais, estão inclusos um conjunto de bibliotecas C/C++ que são utilizadas por muitos componentes do sistema. Essas bibliotecas estão acessíveis aos desenvolvedores através do SDK do Android, (Google Inc, 2012).

O Android possui um conjunto de bibliotecas (Runtime), que oferecem a maioria das funcionalidades das principais bibliotecas da linguagem Java. As aplicações Android rodam em seus próprios processos, com sua própria instância da máquina virtual (VM) *Dalvik*. Esta por sua vez, foi desenvolvida para executar várias máquinas virtuais de forma eficiente, utilizando arquivos.dex, que são otimizados para consumir pouca memória. A VM do Android roda classes compiladas na linguagem Java que são transformadas em arquivos.dex, usando a ferramenta "dx" do SDK, (Google Inc, 2012).

O Android utiliza a versão 2,6 do kernel do Linux para tarefas centrais do sistema, como segurança, gestão de memória e gestão de processos. O kernel também tem a função de fazer abstração entre o hardware e o resto do software, (Google Inc 2012).

2.6 Linguagem Lua

Lua é uma linguagem de programação interpretada, de tipagem dinâmica, poderosa, rápida, leve e possui sintaxe simples o que facilita o aprendizado. Possui gerenciamento automático de memória com coleta de lixo incremental. Lua é de código aberto sob licença MIT e assim como Java roda sobre uma máquina virtual, que gera um *bytecode* permitindo que o código seja interpretado em uma grande variedade de hardware, (DAVILA, 2012 e PUC-RIO, 2013).

A linguagem Lua foi criada no Brasil por uma equipe do Grupo de Tecnologia em Computação Gráfica da PUC-Rio (Pontifícia Universidade Católica do Rio de Janeiro). É amplamente utilizada no desenvolvimento de jogos (e.g., *World of Warcraft e Angry Birds)* e outros *softwares* como *Apache HTTP Server*, *Adobe Photoshop Lightroom,* (PUC-RIO, 2013).

> Lua é distribuída via um pequeno pacote e compila sem modificações em todas as plataformas que têm um compilador C padrão. Lua roda em todos os tipos de Unix e Windows, e também em dispositivos móveis (usando Android, iOS, BREW, Symbian, Windows Phone), em microprocessadores embutidos (como ARM e Rabbit, para aplicações como Lego MindStorms), e até mainframes IBM, (PUC-RIO, 2013).
>
> Lua é uma *engine* rápida e pequena que você pode facilmente embutir na sua aplicação. Lua tem uma API simples e bem documentada que permite uma integração forte com

> código escrito em outras linguagens. É simples estender Lua com bibliotecas escritas em outras linguagens. Também é simples estender programas escritos em outras linguagens com Lua. Lua é usada para estender programas escritos não só em C e C++, mas também em Java, C#, Smalltalk, Fortran, Ada, Erlang, e mesmo outras linguagens de script, como Perl and Ruby, (PUC-RIO, 2013).

Lua é uma linguagem que permite certo controle aos designers de games sem uma grande curva de aprendizagem. Lua destaca-se em desenvolvimento de jogos quando é usada em conjunto com uma linguagem mais poderosa de baixo nível como C++, além disso, a linguagem permite a criação rápida de protótipos de situações do jogo ou até jogos completos, (SCHUYTEMA, 2008).

2.7 Corona SDK

O Corona SDK foi desenvolvido pela Ansca Mobile fundada em 2008, a plataforma foi lançada em 2010 e atualmente se encontra na versão 2013.1076. O SDK gera aplicações para quatro plataformas: IOS, Android, Kindle Fire e Nook. Ele usa *Lua* como linguagem de programação, a função do Corona é traduzir essa aplicação para o iOS ou o Android. (DAVILA, 2012).

O Corona foi escolhido entre outras ferramentas mesmo sendo paga, devido ao suporte oferecido pela empresa e quantidade de material disponível, o que facilita o trabalho de desenvolvedores inexperientes.

O Corona SDK promete diminuir consideravelmente o ciclo de criação de jogos para Android, porque elimina a dificuldade de programar em Java. Isso é alcançado usando a linguagem de programação *Lua*, que é amplamente usada no desenvolvimento de jogos digitais. (DAVILA, 2012).

2.7.1 Requisitos Mínimos

Segundo a Ansca Mobile, o mínimo necessário para instalar o Corona é:

No Macintosh:

- Mac OS X 10.6 ou superior
- Processador Intel
- CPU 64 bits
- Sistema gráfico OpenGL 1.4 ou superior
- Xcode para Mac OS X

- Assinatura no Apple Developer Program (necessário para testar seus aplicativos no seu dispositivo)
- Editor de Texto

No Windows:

- Windows 7, Vista ou XP
- Processador de 1GHz (recomendado)
- 38 MB de disco livre (mínimo)
- 1 GB de memória RAM (mínimo)
- Sistema gráfico OpenGL 1.3 ou superior (disponível nas versões mais recentes do Windows)
- Java 6 SDK
- Editor de Texto

2.7.2 Licenciamento

O Corona possui uma licença do tipo assinatura, é paga anualmente e todas as atualizações futuras podem ser baixadas sem custo adicional. Até a presente data, existem 3 modelos de assinatura: *Starter*, *Pro* e *Enterprise*. A assinatura é necessária caso o programador queira publicar o aplicativo nas lojas oficiais, (DAVILA, 2012).

Assinantes têm prioridade na hora de fazer o *build* da aplicação, essa *build* possui um tamanho otimizado, além disso, assinantes tem acesso a fóruns exclusivos.

2.7.3 Estrutura de um Projeto

Um projeto no Corona SDK tem apenas um arquivo principal, chamado *main.lua*. Outros arquivos podem compor um projeto, os principais são: *config.lua*, e *build.settings*. Outros arquivos podem ser adicionados para deixar os projetos mais organizados.

A figura 7 a seguir mostra a estrutura básica de um projeto.

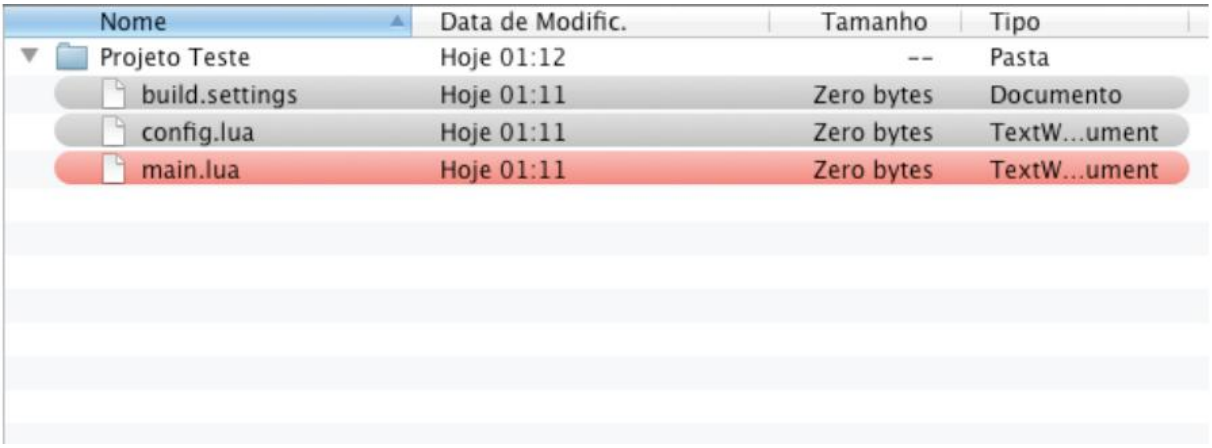

Figura 7- Estrutura de um projeto no Corona SDK – Fonte: DAVILA (2012)

O arquivo *config.lua* contém informações referentes ao *runtime*. A principal informação contida nesse arquivo é referente ao tamanho da tela e redimensionamento. O arquivo *build.settings* apesar de ter a extensão diferente dos restantes, esse arquivo usa a sintaxe da linguagem *Lua* e nele devem ser inseridas instruções para serem usadas na hora do *build*, (DAVILA, 2012).

3 Elementos Gráficos: *DisplayObjects*

Todos os desenhos que aparecem na tela são criadas através de *DisplayObject*. Os objetos não podem ser criados diretamente. São criados tipos especiais de *DisplayObject*, figuras como retângulos, círculos, imagens, texto, etc, (CORONA, 2013).

Você pode criar um *DisplayObject* chamando um tipo especial de função, chamada de construtor. Este implicitamente adiciona o objeto acima de todos os outros objetos que você criou. Existem construtores para cada tipo específico de *DisplayObject* , por exemplo *display.newRect* ( ) cria um *VectorObject,* (CORONA, 2013).

Todas as instâncias de *DisplayObject* tem comportamento semelhante ao de tabelas Lua normais. Isso significa que você pode adicionar suas próprias propriedades para o objeto, enquanto eles não entrem em conflito com as propriedades e nomes de métodos abaixo, (CORONA, 2013).

3.1 **Principais Propriedades**

Abaixo estão as propriedades comuns compartilhadas por todos os objetos de exibição que utilizam o operador ponto de acordo com CORONA, 2013.

- ***object.alpha*** é a opacidade do objeto. Um valor de 0 é transparente e 1.0 é opaco. O valor padrão é 1.0.

Note-se que esta é uma propriedade diferente alfa do que o componente opcional "alfa" encontrado em cada especificação da cor da forma (r, g, b, [a]). Neste último caso, a faixa alfa é 0-255, assim como as faixas de vermelho, verde e azul, porque o conjunto de valor resultante especifica uma cor de 32 bits com 8 bits por canal.

- ***object.height*** é em coordenadas locais

- ***object.isVisible*** controla se o objeto é visível na tela. **Verdade** é visível e **falso** não é.

- ***object.isHitTestable*** permite que um objeto continue a receber eventos de sucesso , mesmo que não seja visível. Se **verdadeiro** , os objetos vão receber eventos de sucesso, independentemente da visibilidade, se **falsa** , os eventos são enviados apenas para objetos visíveis.

- ***object.length*** [somente leitura] é preterido em favor de *group.numChildren* . Para o grupo de objetos, este é o número de filhos; 0 caso contrário.
- ***object.parent*** [somente leitura] Retorna o grupo pai do objeto.

- ***object.rotation*** é o ângulo de rotação da corrente (em graus).

- ***object.contentBounds*** é uma tabela com propriedades xMin , xMax , yMin , yMax em coordenadas de tela.Geralmente usado para mapear o objeto em um grupo para as coordenadas de tela.
- ***object.contentHeight*** é a altura em coordenadas de tela.

- ***object.contentWidth*** é a largura em coordenadas de tela.

- ***object.width*** é em coordenadas locais

- ***object.x*** especifica a posição x (em coordenadas locais) do objeto em relação ao pai - a origem do pai para ser mais preciso. Especificamente, ele fornece a posição-x do ponto de referência do objeto em relação ao pai. Mudar o valor deste irá mover o objeto na direção x. Alterar ponto de referência do objeto mudará x do objeto e os pontos y sem mover o objeto.

- ***object.xOrigin*** especifica a posição-x da origem do objeto em relação à origem dos pais. É em coordenadas locais do objeto. Mudar o valor deste irá mover o objeto na direção x.

- ***object.xReference*** define a posição-x do ponto de referência em relação ao local de origem do objeto. É em relação a outro ponto do objeto, e não a seu pai. Conceitualmente, o ponto de referência é o local sobre o qual a escala e rotação de ocorrer. Por vezes, este é referido como o ponto de registro. Para a maioria dos objetos de exibição, este valor padrão é 0, ou seja, a posição-x da origem eo ponto de referência são os mesmos. Isso limita a definir o ponto de referência, de modo a mudar o valor dessa propriedade não altera a posição do objeto.

- ***object.xScale*** obtém ou define o fator de escala X. Um valor de 0,5 irá dimensionar o objeto a 50% no sentido X.

- ***object.y*** especifica a posição-y (em coordenadas locais) do objeto em relação ao pai - a origem do pai para ser mais preciso. Especificamente, ele fornece a posição y do ponto de referência do objeto em relação ao pai. Mudar o valor deste irá mover o objeto na direção y. Alterar ponto de referência do objeto mudará x do objeto e os pontos y sem mover o objeto.

- ***object.yOrigin*** especifica a posição y da origem do objeto em relação à origem dos pais. É em coordenadas locais do objeto. Mudar o valor deste irá mover o objeto na direção y.

- ***object.yReference*** define a posição y do ponto de referência em relação ao local de origem do objeto. É em relação a outro ponto do objeto, e não a seu pai. Conceitualmente, o ponto de referência é o local sobre o qual a escala e rotação de ocorrer. Por vezes, este é referido como o ponto de registro. Para a maioria dos objetos de exibição, este valor padrão é 0, ou seja, a posição y da origem e o ponto de referência são os mesmos. Isso limita a definir o ponto de referência, de modo a mudar o valor dessa propriedade não altera a posição do objeto.

- ***object.yScale*** obtém ou define o fator de escala Y. Um valor de 0,5 irá dimensionar o objeto a 50% no sentido Y.

3.2 Principais Métodos

Lua pode chamar métodos de objeto de duas formas: usando o operador ponto ou dois pontos. A figura 8 a seguir mostra exemplos de como são usadas às duas formas.

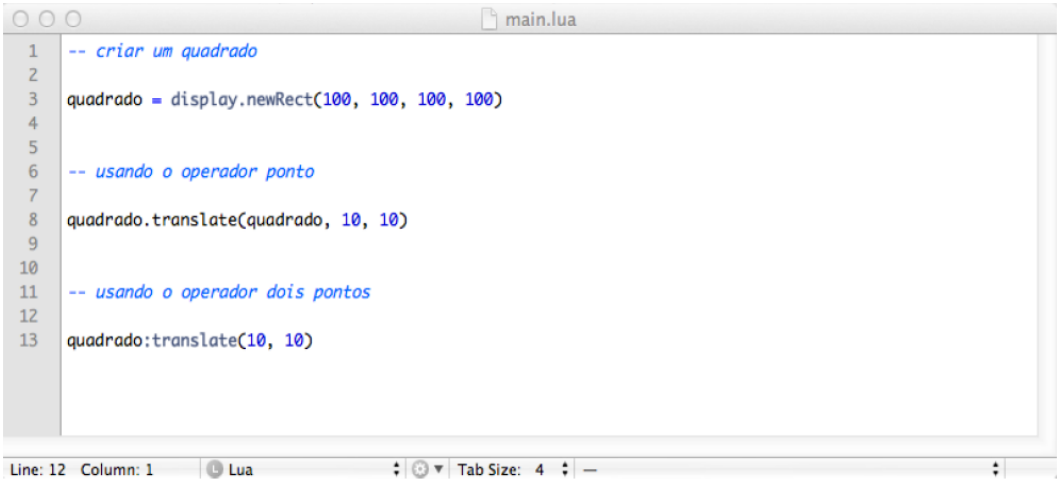

Figura 8 – Exemplo de chamada de método – Fonte (DAVILA, 2012).

Abaixo será descrito os principais métodos nativos dos *DisplayObjects, (*Fernandez, 2012 *apud* DAVILA, 2012).

- ***object:rotate(deltaAngle):*** adiciona uma quantidade *delta* em graus ao objeto para rotacionar.
- ***object:scale(sx, sy)*:** multiplica *XScale* e *yScale* propriedades por *sx* e *sy* respectivamente.
- ***object:setReferencePoint(referencePoint)*:** define o ponto de referência tanto para o centro do objeto (padrão) ou para um dos vários pontos convenientes ao longo da caixa delimitadora do objeto. O *referencePoint* argumento deve ser uma dos parâmetros abaixo:
  - o *display.CenterReferencePoint*
  - o *display.TopLeftReferencePoint*
  - o *display.TopCenterReferencePoint*
  - o *display.TopRightReferencePoint*
  - o *display.CenterRightReferencePoint*
  - o *display.BottomRightReferencePoint*
  - o *display.BottomCenterReferencePoint*
  - o *display.BottomLeftReferencePoint*
  - o *display.CenterLeftReferencePoint*

- ***object:translate(deltaX, deltaY)*:** acrescenta *deltaX* e *deltaY* para as propriedades *x* e *y*, respectivamente. Isto irá mover o objeto de sua posição atual.
- ***object:removeSelf()*:** remove o objeto da exibição na tela e libera o espaço usado na memória, assumindo que não há outras referências a ele.

3.3 Manipulação de Imagens

Adicionar imagens no Corona é muito simples, esta ação é feita através do método construtor *display.newImage(filename, [,base directory] [, left, top])*, onde *filename* é o nome da imagem, *base directory* representa o caminho onde se encontra o arquivo e *top*, *left* são dois

parâmetros de coordenadas que devem ser informados juntos, se informado, a imagem será posicionada no local informado, caso não seja informado, a imagem será centrada sob sua origem. Corona suporta dois formatos de imagens JPG e PNG, demais formatos como BMP, TIFF e GIF não são suportados, (DAVILA, 2012).

Quando se adiciona uma imagem maior que a área de exibição no Corona, a imagem é redimensionada automaticamente para que se ajuste ao tamanho da tela, porém, nem sempre é necessário que a imagem seja redimensionada, por isso, existe um indicador booleano na lista de parâmetros que altera essa propriedade: *display.newImage([parentGroup,] filename [, baseDirectory] [, x, y] [,isFullResolution,* (DAVILA, 2012).

> O parâmetro *parentGroup* indica o grupo *pai* do cenário ao qual pertence o objeto e *isFullResolution* é o parâmetro responsável por permitir o redimensionamento ou não. O valor padrão é *false*. Caso seja indicado *true* a imagem é carregada na sua resolução total. (DAVILA, 2012).

3.3.1 Lidando com Tamanhos Diferentes de Tela

Algumas configurações podem ser adicionas no *config.lua* para a hora do *runtime*. Sabe-se que o Android é um sistema operacional bastante fragmentado, com uma grande quantidade de aparelhos e tamanhos de tela. O Corona SDK ajuda na manutenção da uniformidade no desenvolvimento de jogos, são utilizados parâmetros no *runtime* que determinam automaticamente o tamanho do cenário e reorganiza os elementos na tela de uma maneira satisfatória, (DAVILA, 2012).

O Corona possibilita alternar entre imagens de baixa e alta resolução sem que seja necessário alterações no código, este é um recurso muito útil levando em consideração a grande quantidade de dispositivos com diferentes resoluções de tela.

Para fazer uso dessa funcionalidade é necessário editar o arquivo *config.lua* e incluir a propriedade de tabela *imageSuffix.* Na tabela *imageSuffix os* índices representam a *string* adicionada no final do nome do arquivo e qual o fator multiplicador daquele sufixo um exemplo pode ser visto na Figura 9 abaixo, (DAVILA, 2012).

config.lua

```
application =
{
    content =
    {
        width = 320,
        height = 480,
        scale = "letterbox", -- para o Android é melhor o zoomStrech
        imageSuffix =
        {
            ["@2x"] = 2 -- determina o sufixo da imagem, por exemplo no
                        -- iPhone 3gs seria moon.png e pro iphone 4 moon@2x.png
        }

        xAlign = "left",
        yAling = "top"
    }
}
```

Figura 9 – Exemplo de redimensionamento de imagem dinâmico. – Fonte: (DAVILA, 2012).

Para carregar imagens de resolução dinâmica usa-se o comando *display.newImageRect([parentGroup,] filename [, basedirectory], w, h),* onde w e h representam a largura e altura respectivamente, (CORONA, 2013).

3.4 Texto

Para inserir um texto usa-se o método *display.newText([parentGroup,] string, x, y, font, size),* onde *font* determina o tipo de fonte, *size* indica o tamanho da fonte. Para o parâmetro *font* pode ser utilizado algumas constantes no lugar de uma *string* como *native.systemFont* ou *native.systemFontBold*.

Abaixo segue alguns métodos que alteram propriedades do texto:

- **object.size**: define o tamanho do texto.
- ***obect:setTextColor(r, g, b [, a]):*** altera a cor de acordo com o código RBG, e define a opacidade que varia de 0 a 255, o padrão é 255.
- ***object.text:*** é a string que contem o texto.

A Figura 10 abaixo mostra um exemplo de utilização de alguns dos métodos citados.

```
locais meuTexto = display.newText (  "Olá, mundo" , 0 , 0 , "Helvetica" , 16  )
meuTexto: SetTextColor ( 0 , 0 , 255 )

- Alterar o valor do meuTexto
mytext.text =  "Olá, universo!"
```

Figura 10 – Exemplo de utilização de métodos de modificação de texto. Fonte – (Corona, 2013).

3.5 Inserção de áudio

Como foi dito anteriormente jogos digitais possuem muitos componentes, entre eles um dos principais são os efeitos sonoros, que proporcionam uma maior imersão aos jogadores, e muitas vezes criam uma identificação única para os jogos, por exemplo: Super Mario Bros, *Bomber Man* e *Angry Birds*.

O *Corona Audio System* usa recursos *OpenAL*. Os formatos suportados são:

- Todas as plataformas suportam arquivos de 16 bits
- iOS e o simulador para Mac suportam .mp3, .caf, .aac
- O simulador para Windows suporta . mp3 e .wav
- Android suporta .mp3 e .ogg

A diferença entre os formatos suportados pelas diferentes plataformas se deve ao licenciamento de cada formato. O Android possui uma limitação no que se refere ao nome do arquivo de áudio, ele simplesmente ignora a extensão do arquivo, então uma boa forma de contornar esse problema é adicionar o nome do formato junto com o nome do arquivo, por exemplo: audio_mp3.mp3.

O Corona oferece duas maneiras de inserir áudio, por *stream* ou carregamento completo do arquivo na memória. A inserção de áudio utilizando *stream* economiza memória, pois o som é carregado aos poucos, o método utilizado é o *audio.loadStream.* Na inserção por carregamento completo o áudio é carregado uma única vez e pode ser usado várias vezes, o método utilizado é o *audio.loadSound*.

Sintaxe:

- *audio.loadStream(audiofilename [, baseDir])*
- *audio.loadSound(audiofilename [, baseDir])*

Parâmetros:

- *audioFilename* – especifica o nome do arquivo de áudio
- *baseDir* – é opcional e deve ser usado quando o local onde se encontra o arquivo não esteja no diretória raiz do projeto.

3.5.1 Controle de áudio

O Corona oferece funcionalidades básicas de controle de áudio como *play, stop, pause, resume e rewind.* Os controles de áudio são facilmente implementados, mas cada arquivo de áudio está associado a um único canal, sendo assim um áudio não pode ser pausado em si, e o controle é sobre o canal, (DAVILA, 2012).

3.5.2 audio.play

O método audio.play executa arquivo de áudio especificado por *audioHandle,* se um canal não for especificado o Corona escolherá um.

Sintaxe:

audio.play(audioHandle [, {[channel=c] [, loops=l] [, duration=d] [, fadein=f] [, onComplete=o]}])

Parâmetros:

- *audioHandle:* carrega o objeto de áudio com um dos métodos de *load*.
- *channel*: varia de 1 a 32, caos não seja informado um canal será escolhido automaticamente.
- *loops*: indica quantas vezes o áudio será repedido, se for indicado -1 será executado infinitamente.
- *duration:* tempo em que o áudio será executado em milisegundos.
- *fadein:* tempo em milisegundos em que o áudio irá começar a ser executado do volume mínimo até o normal.
- *onComplete:* recebe o nome de método que pode vir a ser executado depois do áudio.

4 API de Physics

O Corona SDK utiliza o motor de física Box2D, que permite a manipulação de física com poucas linhas de código. Todos os *DisplayObject* podem ser manipulados pela *Physics* API porque o Corona trata cada *PhysicObject como uma extenção de DisplayObject*, (DAVILA, 2012).

Por padrão o Box2d trabalha com o sistema métrico de medida, mas a API faz todas as conversões necessárias. A escala padrão é de 30 pixels por metro e todas as medidas são feitas em graus para manter a consistência.

Existem 3 métodos que alteram o status de um objeto, são eles:

- *physics.start()* – reinicia ou resume o ambiente físico.
- *physics.pause()* – para o motor de física temporariamente.
- *physics.stop()* – destrói o ambiente físico completamente.

4.1.1 Gravidade

A gravidade pode ser tratada bidirecionalmente ou em apenas uma direção, para definir a gravidade, usa-se o método p*hysics.setGravity(x, y),* onde *x* e *y* são o par ordenado do eixo cartesiano. O valor padrão para essa função é *x = 0* e *y =9.8*, onde 9,8 representa a gravidade da terra.

Outra maneira oferecida pelo SDK de manipular a gravidade de um objeto é baseá-la de acordo com a inclinação do dispositivo. A implementação é muito simples, para isso é apenas necessário que a API do acelerômetro esteja aplicada.

A Figura 11 mostra um exemplo de utilização do acelerômetro:

Figura 11 – influência do acelerômetro na gravidade, Fonte (DAVILA, 2012).

Podemos observar que o método *onTilt()* é passado como parâmetro quando um *listener* para o acelerômetro é criado, quando o dispositivo se mover, o acelerômetro será chamado e o evento disparado.

4.1.2 Sensores

Sensores não interagem diretamente com outros corpos, mas produzem eventos de física quando outros corpos passam por eles. Um exemplo seria a caçapa de uma mesa de sinuca.

Para tornar um objeto como sensor, é necessário ao criar o objeto, passar o parâmetro *isSensor* como *true: physics.addBody(rect, {isSensor: true})* (DAVILA, 2012).

4.1.3 Colisões

Na detecção de colisões, o objeto deve ter um *listener* ligado ao chamado *collision*. O evento "*collision*" tem duas fases, "*began*" e "*ended*", que são respectivamente os momentos de contato inicial e final durante a colisão. Essas fases funcionam tanto para a colisão simples entre dois corpos ou sensores. Existem mais dois eventos além *collision,* que podem ser usados com corpos, não sensores, (DAVILA, 2012).

- *preCollision*: é disparado antes dos objetos começarem a interagir, depende da lógica aplicada ao jogo, talvez seja necessário detectar este evento e sobrescrever a colisão.
- *postCollision*: é ativado após a colisão. Este é o único evento em física é aplicada.

4.1.4 Corpos Físicos

Corpos físicos são uma extensão de *DisplayObject*, ou seja, quando o construtor *physics.addBody* não retorna um novo objeto, mas apenas estende o *DisplayObject* com propriedades físicas. Porém, uma vez adicionadas as propriedades físicas do objeto, não é possível removê-las, apenas é possível remover o objeto como um todo, (DAVILA, 2012).

4.1.5 O construtor physics.addBody

O comando *physics.addBody* adiciona propriedades físicas a um objeto existente.

Sintaxe:

- *physics.addBody(object, [bodyType,] {density=d, friction=f, bounce=b [,radius=r] [,filter=f]})*
- *physics.addBody(object, [bodyType,] {density=d, friction=f, bounce=b [,shape=s] [,filter=f]})*

Parâmetros:

- *density*: representa a densidade do objeto, possui valor padrão 1
- *friction:* valor de fricção, o padrão é 0.3, podendo ser um valor negativo. Se o valor for 0 o corpo não possui fricção se for 1 possui fricção forte.

- *bounce:* é um propriedade interna do Box2d conhecida como “restituição”, determina a velocidade do objeto após as colisões, o valor padrão é 0.2 que indica que o objeto perde velocidade, se o valor for 1.0 o objeto não perde velocidade, se for maior que 1.0 o objeto ganha velocidade, (DAVILA, 2012).

> Por padrão, todos os objetos criados tem forma retangular, corpos circulares exigem um parâmetro a mais, o *radius*, no caso esse parâmetro recebe o raio. Não é indicado para corpos ovais, pois o Box2D apenas trata de colisões circulares, retangulares e poligonais, (DAVILA, 2012).
>
> Corpos poligonais precisam do parâmetro *shape.* Esse campo é uma tabela de pontos coordenados que formam as arestas do polígono. O máximo de pontos indicados permitidos por colisão é de oito. Uma importante observação é que todo polígono criado deve ser convexo, caso seja necessário criar um corpo côncavo é necessário criar múltiplos objetos e anexar uns aos outros, (DAVILA, 2012).
>
> Os corpos físicos também podem ser classificados de três maneiras: *static*, *dynamic* e *kinematic*. O *bodyType* padrão é o *dynamic.* Geralmente objetos que se movem são do tipo *dynamic*, e objetos que não se movem ou não são afetados por gravidade, de *static.* Objetos *kinematic* são afetados por forças, porém a gravidade em nada interfere, no caso, se o evento de *drag* (arrastar) for agir nesse objeto, seu *bodyType* deve ser esse, ao menos enquanto o evento de *drag* estiver em ação, (DAVILA, 2012).

5 Recursos Oferecidos

A Ansca mantém um portal onde oferece vários serviços e informações para os desenvolvedores que utilizam o SDK, a página inicial é www.coronalabs.com/. As principais informações se encontram nas opções *Community e Resources.*

Na página *Community* estão algumas opções importantes para auxiliar os usuários a tirar dúvidas e melhorar o entendimento sobre técnicas de programação, as opções que mais se destacam são o fórum e *Code Exchange* onde os membros da comunidade compartilham seus códigos para ajudar outros programadores.

Em *Resources* se encontram mais opções para ajudar os desenvolvedores, as principais são:

- *Corona Plugin Directory:* diretório que contem *plugins* que dão acesso a funcionalidades adicionais e serviços de terceiros.
- *Corona SDK Video Tutorials:* tutoriais em vídeo.
- *Case Studies:* exemplos práticos de aplicativos e jogos baseados em estudos de caso compartilhados pela comunidade.
- *Corona SDK Books and Courses*: Loja virtual contendo livros físicos e virtuais (sendo alguns em português) além de cursos disponibilizados em DVD ou via download.

- *Corona Tech Tutorials*: Coleção dos melhores tutoriais coletados da internet, desde o nível básico ao avançado.
- *Documentation*: descrição completa da API do Corona.
- *3rd Party Tools and Services: lista de* ferramentas desenvolvidas por terceiros que complementam as funcionalidades do SDK.

5.1 SDK

Iremos abordar agora as funcionalidades e recursos oferecidos pela Corona SDK, abaixo poderá ser observado a tela inicial da ferramenta.

Figura 12 – Tela inicial do SDK – Fonte: o autor

Podemos observar que algumas opções são iguais do site Corona, são elas *Tutorials, Share Code, Forums, APIs*, que levam direto para páginas do site. Destacamos outra opção que muito importante, *Sample Apps,* onde existem várias aplicações e jogos prontos que podem ser tomados como exemplo para desenvolvedores iniciantes, os códigos fonte estão no diretório no qual o SDK se encontra instalado.

Na opção *Demo* são encontrados mais exemplos práticos que introduzem técnicas básicas de programação utilizando o Corona, em *Demo* encontramos três tutoriais abordando imagens e texto, eventos envolvendo toques e sons, e outro mostrando a utilização de física.

A opção *Dashborad* é uma área destinada a assinantes que recebem suporte personalizado, *Simulator* é onde são feitas as simulações das aplicações. Em *Newproject* são criados novos projetos.

5.2 Geração da *build*

Gerar uma *build* para o *Android* é muito simples, já que não é preciso nenhum tipo de chave para gerar os executáveis para testes. Caso o objetivo seja a publicação no Google Play, é necessário ter uma *Keystore* ou uma *Key Alias*.

- Abra o simulador.
- Clique em File > Open e selecione o arquivo main.lua do projeto.
- Depois acesse a opção File > Build for Android, a janela a seguir surgirá:

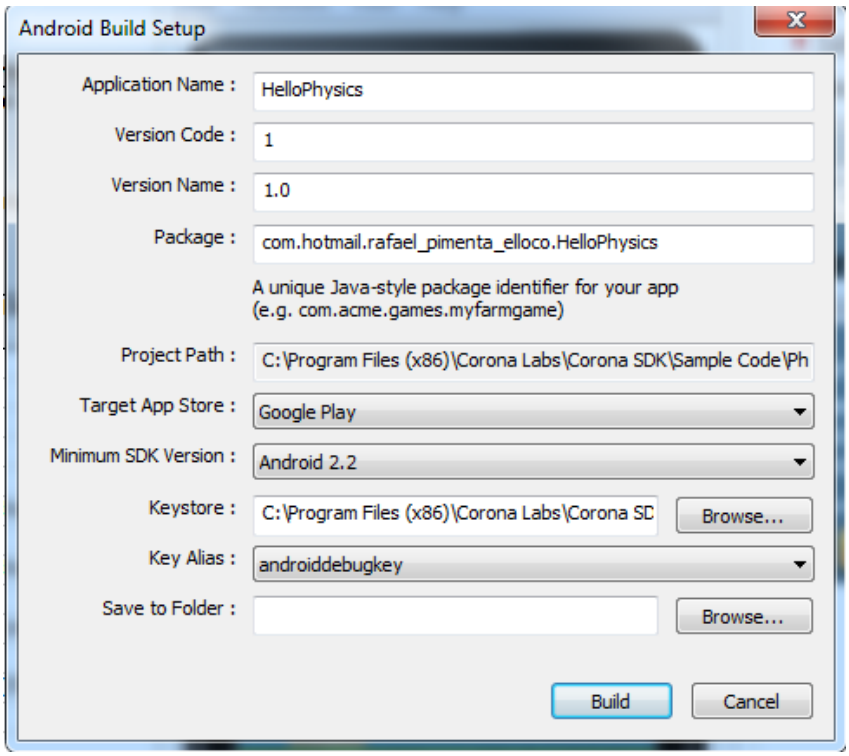

Figura 13 – Geração da build para Android- Fonte: o autor

- Devemos preencher os campos *Application Name, Version Code, Version Name e Package.* Selecionar a versão do Android desejada (por padrão utiliza 2.2 já que existe retrocompatibilidade com versões posteriores), para efeito de testes em dispositivos e não publicação em lojas, selecione Debug para a Keystore e androiddebugkey para a Key Alias, em seguida selecione a pasta destino para o arquivo.

Após a geração da *build* utilize um gerenciador de arquivos para transferir o arquivo apk para seu dispositivo.

5.3 Monetização

Alem da venda dos aplicativos e jogos nas lojas virtuais os desenvolvedores podem gerar renda de outras formas, para isso são utilizados vários plugins adicionais que são desenvolvidos por terceiros e disponibilizados no portal do Corona. Esses plugins oferecem meios de arrecadação através de publicidade de marcas e outros aplicativos. Na página referente a essas extensões ainda são encontrados descrição básica da sintaxe e links com exemplos de código.

5.4 Limitações

Um dos maiores problemas do Corona se deve ao fato de seu isolamento, ele não permite chamadas direta às APIs do Android ou do iOS, por isso alguns recursos não podem ser implementados. Outra limitação é em relação ao *Android*, pelo fato do SDK ter sido desenvolvido baseado na versão 2.2 do sistema operacional, então novos recursos oferecidos pelas versões posteriores não estão disponíveis para os desenvolvedores que utilizem o Corona. Por outro lado as aplicações desenvolvidas funcionam normalmente em versões recentes devido à retro compatibilidade. (DAVILA, 2012).

5.5 Pontos Positivos

As principais vantagens em relação ao uso do Corona SDK são a API simples e a utilização da linguagem de programação Lua, uma linguagem simples e poderosa, e a possibilidade de desenvolver jogos e aplicações multiplataforma.

Toda documentação está disponível para consulta no site da *Ansca*, ela é bastante completa e conectada a diversos artigos com dicas e tutoriais que são periodicamente publicados no blog hospedado no mesmo site. O site também conta com um fórum, que apesar de disponível só para assinantes, é bastante ativo.

Outro destaque são os exemplos distribuídos juntamente com a SDK, pois com eles é possível ver e aprender como determinadas tarefas podem ser realizadas. Além disso, no site da *Ansca* estão disponíveis para download modelos completos de jogos que você pode utilizar para aprender ou até mesmo como base para não ter que começar seu jogo do zero.

6 Estudo de caso

O objetivo do estudo de caso é desenvolver um protótipo de um jogo simples chamado gamebox, ele foi desenvolvido a partir de informações e tutoriais disponibilizados pela comunidade e pelo SDK. Tomamos como base um exemplo encontrado na opção *Demo* do SDK, uma animação simples que utiliza simulação de física que consiste em uma caixa caindo sobre o chão. Algumas alterações foram feitas, ao invés de uma única caixa caindo serão várias caixas caindo de forma aleatória.

Foi usada a IDE *Lua Development Tools* baseada na IDE eclipse, que pode ser baixada gratuitamente no link http://www.eclipse.org/koneki/ldt/ .

O projeto possui dois arquivos, o principal obrigatoriamente com o nome de *main.lua*, e o outro *config.lua* que possui informações básicas como resolução de tela e escala utilizada e a quantidade de frames por segundo, a figura a baixo mostra o código referente ao arquivo *config.lua*.

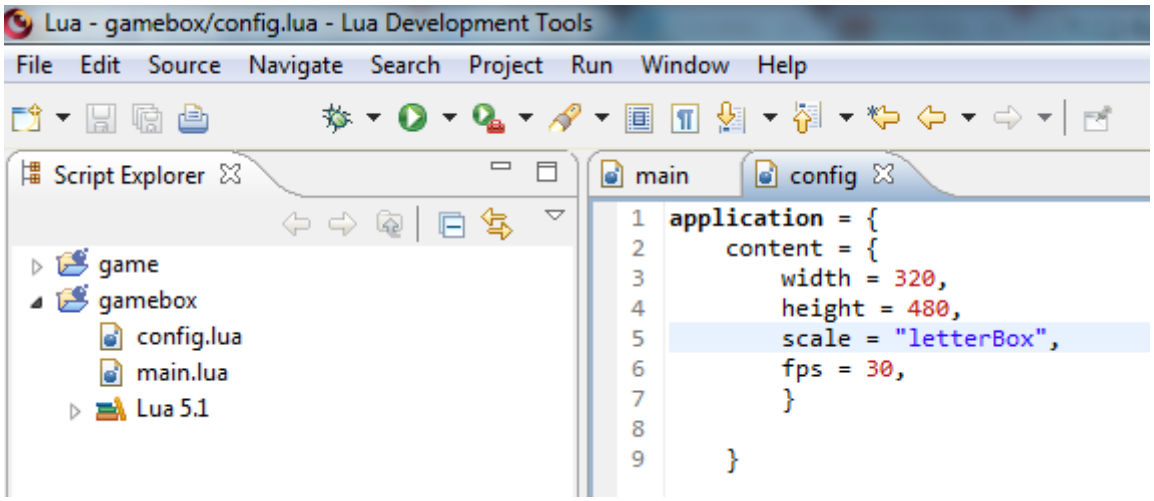

Figura 14 – arquivo *config.lua* – Fonte: o autor

No *main.lua* está toda a implementação de fato, onde é feito o carregamento dos elementos que compõe o cenário, atribuição de características aos objetos como densidade, fricção e elasticidade. A seguir o código comentado do arquivo main.lua.

```
local physics = require "physics" --requisita uso da biblioteca physics
physics.start() --inicia o ambiente físico

local sky = display.newImage("background.jpg") --carrega imagem que servirá como plano
de fundo
local grass = display.newImage("grass.png") -- carrega imagem que representará o solo
grass.y = 439 -- posição onde se encontrará a imagem
```

```
physics.addBody(grass, "static", {friction=0.3}) -- atribuição de propriedades físicas
ao solo
physics.setGravity(0, 9,8) -- atribuindo ao ambiente a gravidade da Terra

function newCrate()

	local crate = display.newImage("crate.png") --carrega a imagem que será o objeto
	em destaque na cena
	crate.x = math.random(320) -- posição aleatória onde surgirá o objeto no eixo x
	crate.y = -100 -- o valo -100 indica que o objeto surgirá completamente acima da
	tela
	crate.rotation =10 --rotação do objeto
	physics.addBody(crate,{density=2.0, friction=0.5, bounce=0.3})
	--atribui propriedades físicas, densidade, fricção e elasticidade
end
--temporizador
local dropCrates = timer.performWithDelay(500, newCrate, 50)
--[[ Chama uma função após um atraso, 500 é o tempo em milesegundos do atraso,
 newCrate a função que será executada e 50 o número de iterações. --]]
```

A Figura 15 mostra o jogo em funcionamento.

Figura 15 – Jogo em execução – Fonte: o autor

6.1 Testes

Testamos a instalação em quatro dispositivos com diferentes especificações de hardware e versão do SO, a tabela abaixo exibe as configurações dos aparelhos.

Samsung Galaxy 5	LG Optimus L5 II	Samsung Galaxy SII	LG Optimus 4x HD
Processador: 600 Mhz Núcleos: 1 Memória Ram: 256 MB Android: 2.2	Processador: 1 Ghz Núcleos: 1 Memória Ram: 512 MB Android: 4.2	Processador: 1,2 Ghz Núcleos: 2 Memória Ram 1 GB Android: 4.0	Processador: 1,5 Ghz Núcleos: 4 Memória Ram 1 GB Android: 4.1

Tabela 1 – Especificações dos dispositivos utilizados nos testes – Fonte: o autor

A instalação foi bem sucedida em três dispositivos, apenas no Galaxy 5 não foi possível instalar o protótipo do jogo, o motivo não foi identificado pois a *build* do jogo foi gerada para funcionar em aparelhos que possuam *Android* a partir da versão 2.2, que é a utilizada no aparelho em questão. Nos dispositivos restantes o resultado foi satisfatório o jogo funcionou sem travamentos ou lentidão.

Um problema observado foi em relação ao tamanho da *build* gerada pela versão *free* do SDK, o projeto com 363 KB se transformou em um arquivo de 4,36 MB, que após a instalação passou a ocupar 7,93 MB de espaço em disco, deixando uma aplicação muito simples com um tamanho relativamente grande, comprovando que a versão grátis não oferece uma geração otimizada de *build*. Em contrapartida o consumo de memória RAM se mostrou mais eficiente consumindo 10,8 MB. A Figura 15 exibe o consumo de memória do jogo.

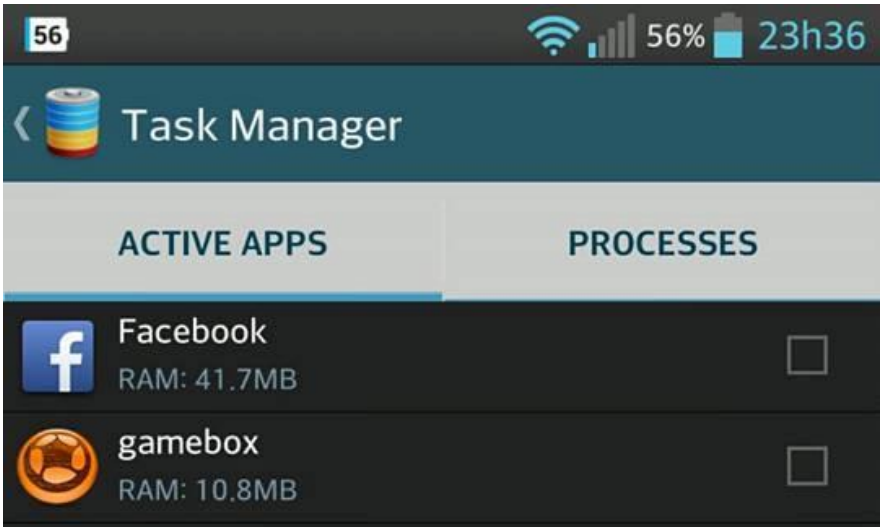

Figura 16 – Consumo de memória RAM do jogo – Fonte: o autor

7 Conclusão

O Corona SDK como uma *engine* de desenvolvimento de jogos para dispositivos móveis se mostra bastante eficiente, o estudo de caso serviu para comprovar isso, com poucas linhas de código foi possível criar uma aplicação simples, em pouco tempo e envolvendo utilização de física. Apesar das limitações encontradas e o fato de ser uma ferramenta paga, os benefícios ainda pesam a favor do SDK. Somando API simples, uma boa documentação, muitos exemplos práticos, tutoriais, uma comunidade ativa e a simplicidade da linguagem Lua, tornam a ferramenta muito atrativa para desenvolvedores, também atribuída ao desenvolvimento multiplataforma alem de poupar tempo no desenvolvimento e consequentemente reduzindo custos, compensando dessa forma o pagamento da assinatura.

REFERÊNCIAS

AMAZONAS, Daniel Souza, **Desenvolvimento de Jogos 3D em Java com a Utilização do Motor Gráfico *Irrlicht.*** Faculdade Lourenço Filho – FLF, Fortaleza CE, 2007. Disponível em <http://www.flf.edu.br/revista-flf/monografias-computacao/monografia_jogo3d.pdf> Acesso em: 12 de março de 2012.

ANDROID DEVELOPERS. **Android Developers**. Disponível em: <http://www.android.com/developers/> Acesso em 29 de março de 2012.

CORONA. **Corona Labs.** Disponível em: <http://developer.coronalabs.com/content/display-objects#Content_Size_Properties> Acessado em: 12 de março de 2013.

DAVILA, Gabriela, **Uma Introdução no Desenvolvimento Móvel de Jogos com Framework Multi-Plataforma.** Faculdade de Tecnologia de Americana - Curso Superior de Tecnologia em Desenvolvimento de Jogos Digitais – Americana SP, 2012. Disponível em < http://www.academia.edu/1237811/Corona_SDK_Uma_Introducao_no_Desenvolvimento_Movel_de_Jogos_com_Framework_Multi-Plataforma> Acesso em 10 de janeiro de 2013.

GARTNER, **Venda de Dispositivos Móveis.** Disponível em: <http://www.gartner.com/it/page.jsp?id=1848514 > Acesso em 13 de abril de 2012.

GOOGLE INC. **Dev Guide.** Disponível em: <http://developer.android.com/guide/basics/what-is-android.html> Acesso em 15 de março de 2012.

JUNIOR, Ademar De Souza Reis, et al.. **Um Estudo Sobre os Processos de Desenvolvimento de Jogos Eletrônicos (Games)**. Departamento de Informática – UFPR, Curitiba – PR, 2002. Disponível em< http://www.ademar.org/texts/processo-desenv-games.pdf > Acesso em: 13 de março de 2012.

LOPEZ, Caroline Oliveira. **Uma abordagem do mercado Android para desenvolvimento de jogos.** Monografia de conclusão de curso. Faculdade de Tecnologia de Americana. Americana – SP, 2011. Disponível em:< http://www.uberdan.com.br/site/sap/carol.pdf >. Acesso em 16 de Setembro de 2012.

Mil anos a Dez Tecnologia, Desenvolvimento e Banco de Dados. **Corona SDK seis meses desenvolvendo jogos e APPs para IOS e Android.** Disponível em: < http://blog.milanosadez.com.br/2012/04/corona-sdk-seis-meses-criando-jogos-e.html > Acesso 27 de abril de 2013.

MOREIRA, Fábio Eduardo, **Desenvolvimento de Jogos Eletrônicos Multiusuários para Dispositivos Móveis.** Faculdade Farias Brito – FFB, Fortaleza – CE, 2011. Disponível em <http://www.ffb.edu.br/sites/default/files/tcc-20102-fabio-eduardo-moreira.pdf> Acesso em: 20 de abril de 2012.

NEWZOO INC. **Infográfico Brasil.** Disponível em: <http://www.newzoo.com/ENG/1607-Infograph_BR_Portugese.html> Acesso em 8 de abril de 2012.

NOGUEIRA, Wallace Franco, et al.. **Plataformas para Desenvolvimento de Jogos para Celulares.** Departamento de Tecnologia da Informação – UFAL, Tabuleiro dos Martins AL, 2004. Disponível em < http://www.dcc.ufla.br/infocomp/artigos/v4.1/art07.pdf > Acesso em 20 de fevereiro de 2012.

NORONHA, Andréia Michelle da Cunha**,** et al.. **Jogo para Aplicativos Móveis Utilizando o Android,** Universidade Católica de Brasília - UCB, Brasília DF 2010. Disponível em < https://www.google.com.br/search?rlz=1C1AVSH_enBR467BR467&sugexp=chrome,mod=17&sourceid=chrome&ie=UTF-8&q=Jogo+para+Aplicativos+M%C3%B3veis+Utilizando+o+Android#hl=pt-BR&rlz=1C1AVSH_enBR467BR467&sclient=psy-ab&q=Jogo+para+Aplicativos+M%C3%B3veis+Utilizando+o+Android+(3)%09Noronha%2C+Andr%C3%A9ia+Michelle+da+Cunha&oq=Jogo+para+Aplicativos+M%C3%B3veis+Utilizando+o+Android+(3)%09Noronha%2C+Andr%C3%A9ia+Michelle+da+Cunha&aq=f&aqi=&aql=&gs_l=serp.3...31781.32651.0.33304.2.2.0.0.0.0.241.475.2-2.2.0...0.0.DkldoE_krt4&pbx=1&bav=on.2,or.r_gc.r_pw.r_cp.r_qf.,cf.osb&fp=b76163a64e539bee&biw=1280&bih=709 > Acesso em 20 de fevereiro de 2012.

PERUCIA, Alexandre Souza, et al.. **Desenvolvimento de jogos eletrônicos: teoria e prática**. São Paulo : Novatec, 2005.

PUC-RIO. **Lua** Disponível em: <http://www.lua.org/portugues.html> Acesso em 10 de abril de 2013.

SCHUYTEMA, Paul, **Design de games: uma abordagem prática.** São Paulo: Cengage Learning, 2008.

TECMUNDO, **O Sistema Operacional Móvel que Conquistou o mundo.** Disponível em: < http://www.tecmundo.com.br/infografico/9010-android-o-sistema-operacional-movel-que-conquistou-o-mundo.htm > Acesso em 13 de Abril de 2012.

ZIMMERMANN, João Carlos, **Desenvolvimento de jogos para celulares,** Centro Universitário Ritter dos Reis, Porto Alegre - RS, 2006. Disponível em: < http://www.uniritter.edu.br/graduacao/informatica/sistemas/downloads/Desenvolvimento_de_Jogos_para_Celulares.pdf > Acesso em 20 de agosto de 2012.