



# **Apostila de Android**

## **Programando Passo a Passo**

**Desenvolvimento de Jogos (Edição Completa)**



# **Apostila de Android**

## **Programando Passo a Passo**

**Desenvolvimento de Jogos (Edição Completa)**

**(VENDA E DISTRIBUIÇÃO PROIBIDA)**

**De : Luciano Alves da Silva (lucianopascal@yahoo.com.br)**

**[www.apostilaandroid.net](http://www.apostilaandroid.net)**

**Rio de Janeiro**

**Janeiro 2013**



## Aviso sobre esta apostila

Antes de iniciar a leitura deste material, veja esses avisos importantes:

Esse material **NÃO PODERÁ SER DISTRIBUÍDO**, em hipótese alguma, em outros sites da Internet ou através outros processos/meios .

Esse material , em hipótese alguma, **NÃO PODE SER COMERCIALIZADO** tanto pela Internet ou de forma impressa.

Se por acaso você encontrar este material sendo distribuído em outro site ou sendo comercializado (sem ser pelo site oficial da apostila), por favor, entre em contato com o autor (ver e-mail na primeira página).



## **Sobre o Autor da Apostila**

Luciano Alves da Silva é Bacharelado em Ciência da Computação pela UNISUAM e Pós-Graduado em Docência do Ensino Superior pelo Instituto A Vez do Mestre (Universidade Cândido Mendes - UCAM). Possui conhecimento e domínio das linguagens de programação Pascal, Java, C/C++, C#, Visual Basic, Delphi, PHP e HTML. Já criou Ambientes de Desenvolvimento Integrado (conhecidos como IDE) como o MakeWare (que trabalha com as linguagens Pascal, C++ e Java) e o AlgoWare (interpretador de algoritmos).

É autor também dos seguintes livros, pela editora AGBOOK

- Aprenda Passo a Passo a Programar em Android – Guia Essencial para Desenvolvedores
- Desenvolvendo Jogos com a Plataforma XNA – Guia para Desenvolvedores.
- Desenvolvendo Jogos com a Ferramenta RPG Maker VX– Guia do Usuário.



## **Apresentação**

Este material é dedicado para aqueles que desejam desenvolver jogos em 2D voltados para a plataforma Android, através da ferramenta de desenvolvimento Eclipse com o Android SDK (usando o Android Developer Tools).

Este material é completo, não é necessário adquirir nenhuma Apostila de Android disponível no site para este aprendizado sobre jogos. O usuário que tenha interesse em desenvolver jogos para essa plataforma, precisa possuir conhecimentos básicos sobre linguagens de programação (de preferência Java ou suas derivadas C++/C#) e algoritmos.



## Índice analítico

<b>Capítulo 1</b>	<b>Introdução ao desenvolvimento de jogos .....</b>	<b>7</b>
<b>Capítulo 2</b>	<b>Visão geral sobre o Google Android .....</b>	<b>9</b>
2.1)	Introdução .....	9
2.2)	Estrutura Geral da plataforma Google Android .....	10
2.2.1)	A arquitetura do Android.....	12
2.2.2)	Aplicações.....	12
2.2.3)	Android Runtime .....	13
2.2.4)	Linux Kernel.....	14
2.3)	Versões da plataforma Android.....	14
2.3.1)	Qual versão do Android utilizamos para desenvolver os jogos ? .....	14
<b>Capítulo 3</b>	<b>Instalando e Configurando o Android Developer Tools .....</b>	<b>16</b>
3.1)	Efetuando o download e configurando o ADT.....	17
<b>Capítulo 4</b>	<b>Começando a desenvolver jogos para Android .....</b>	<b>25</b>
4.1)	- Conhecendo a estrutura geral do projeto Android para jogos.....	29
	O diretório “src” (source) .....	29
	O diretório “res” (resources) .....	37
	Os diretórios “drawable-hdpi”, “drawable-mdpi”, “drawable-ldpi” e “drawable-xhdpi”. .....	37
	O diretório “values” .....	38
	Definindo o nome de um projeto .....	39
	Definindo o ícone da aplicação (jogo) .....	41
	Executando o jogo .....	43
<b>Capítulo 5</b>	<b>Visualizando imagens no jogo (A classe Image) .....</b>	<b>47</b>
<b>Capítulo 6</b>	<b>Movendo elementos do jogo pela tela .....</b>	<b>55</b>
<b>Capítulo 7</b>	<b>Trabalhando com animação de sprites (A classe AnimationSprites) .....</b>	<b>71</b>
<b>Capítulo 8</b>	<b>Detectando colisões entre objetos no jogo .....</b>	<b>76</b>
<b>Capítulo 9</b>	<b>Criando elementos dinâmicos durante a execução do jogo ..</b>	<b>85</b>
<b>Capítulo 10</b>	<b>Trabalhando com as classes Scene e Character .....</b>	<b>95</b>
10.1)	A classe Scene .....	95
10.2)	A classe Character.....	99

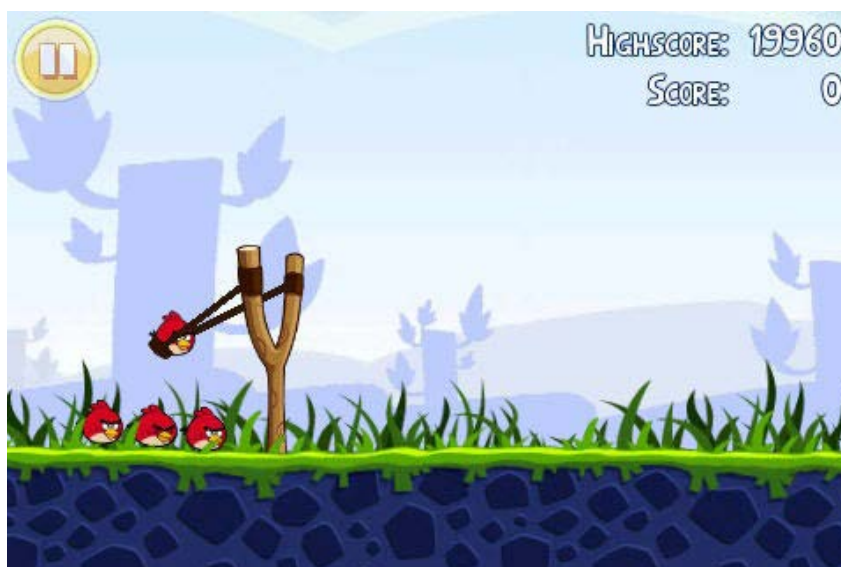


<b>Capítulo 11 Trabalhando com HUD .....</b>	<b>113</b>
11.1) Criando uma HUD simples.....	113
11.2) Criando uma HUD personalizada .....	118
<b>Capítulo 12 Programação com Inteligência Artificial (I.A) .....</b>	<b>129</b>
12.1) Definindo movimentos aleatórios para um elemento de jogo.....	129
12.1.1) Melhorando o código : Não deixar ele sair da tela .....	136
12.1.2) Melhorando o código : Gerando inimigos de cores aleatórias .....	139
<b>Capítulo 13 Músicas e sons no jogo .....</b>	<b>142</b>
13.1) A classe MediaPlayer .....	142
13.2) A classe SoundPool.....	142
13.3) Desenvolvendo o jogo .....	143
<b>Capítulo 14 Trabalhando com mais de uma tela no jogo .....</b>	<b>155</b>
<b>Capítulo 15 Publicando seus jogos Android no Google Play .....</b>	<b>170</b>
<b>Apêndice A Sites e ferramentas para a criação de sprites e sons de efeitos.....</b>	<b>173</b>
<b>Apêndice B Guia de referência da biblioteca GameUtil .....</b>	<b>176</b>
<b>Conclusão a respeito do material .....</b>	<b>184</b>



## **Capítulo 1**      **Introdução ao desenvolvimento de jogos**

Antes de começarmos a desenvolver os nossos jogos para a plataforma Android, precisamos ter em mente o que vem a ser um jogo (de um modo geral). Um jogo nada mais é do que uma aplicação (programa), desenvolvida e destinada somente para o entretenimento. Atualmente encontramos jogos dos mais diversos gêneros para os mais diversos públicos. Vejamos agora alguns títulos conhecidos de jogos que existem hoje em dia:



**Angry Birds (Android)**





# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)



**Devil Ninja (Android)**

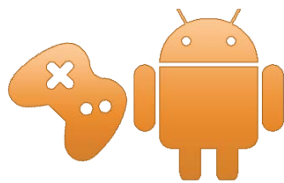


**Rayman Jungle Run (Android)**

Para desenvolvermos um jogo, naturalmente, precisamos ter algumas coisas em mente : seu início, meio e fim (história do jogo de modo geral, não importando seu gênero) ; as ferramentas que vamos utilizar para desenvolver os jogos e programadores/artistas que estarão envolvidos no projeto do jogo.

Aqui nesta apostila estaremos utilizando como ferramenta para o desenvolvimento do nosso jogo, a ferramenta Eclipse juntamente com o Android SDK (usando o Android Developer Tools).





## **Capítulo 2** Visão geral sobre o Google Android

**C**omo vamos desenvolver jogos para o Android, neste capítulo irei falar a respeito da plataforma e de sua arquitetura de forma geral, pois desta forma teremos conhecimento e base suficiente para desenvolver jogos para o Android, avaliando hardware e versões da plataforma.

### **2.1) Introdução**

O Android é uma plataforma desenvolvida pela Google voltada para dispositivos móveis, totalmente aberta e livre (Open Source), que foi divulgada em 5 de novembro de 2007. Inicialmente o sistema Android foi desenvolvido pelo Google e atualmente essa plataforma é mantida pela OHA (Open Handset Alliance. Visite o link : <http://www.openhandsetalliance.com>), um grupo constituído por aproximadamente 84 empresas as quais se uniram para inovar e acelerar o desenvolvimento de aplicações e serviços, com o objetivo e trazer aos consumidores uma experiência mais rica em termos de recursos, menos dispendiosa em termos financeiros para o mercado móvel.

Um dos primeiros Smartphones que ofereceu suporte a esse sistema operacional foi o G1 da empresa T-Mobile. Confira na imagem seguinte:



**G1 - T-Mobile**



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

Atualmente o sistema Android se encontra hoje disponível tanto em SmartPhones quanto nos famosos Tablets. Confira abaixo alguns dos dispositivos encontramos hoje no mercado com o sistema operacional Android:



**SmartPhone Samsung Galaxy**



**Tablet Motorola XOOM**



## 2.2) Estrutura Geral da plataforma Google Android

O Android SDK é uma ferramenta de desenvolvimento que disponibiliza um conjunto de APIs necessárias para desenvolver aplicações para a plataforma Android, utilizando a linguagem Java.

Vamos conhecer os recursos encontrados nessa plataforma:

- **Application framework:** Permite a reutilização e substituição de componentes ;
- **Dalvik virtual machine:** É uma Máquina Virtual Java (JVM) voltada para dispositivos móveis ;
- **Browser Integrado** baseado no webkit engine ;
- **Gráficos Otimizados** O Android é constituído por bibliotecas 2D e 3D baseada na especificação OpenGL ES 1.0 ;
- **SQLite:** Sistema Gerenciador de Banco de Dados (SGBD) já embutido no Android para guardar dados ;
- **Suporte multimídia:** A plataforma já oferece para áudio, vídeo e formatos de imagem (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF) ;
- **Telefonia GSM** (dependente de hardware) ;
- **Bluetooth, EDGE, 3G, e WiFi** (dependente de hardware) ;
- **Câmera, GPS, compasso, e acelerômetro** (dependente de hardware) ;
- **Rico ambiente de desenvolvimento** , incluindo um emulador de dispositivo, ferramentas de depuração, memória, performance e um plugin para o Eclipse (ADT) ;

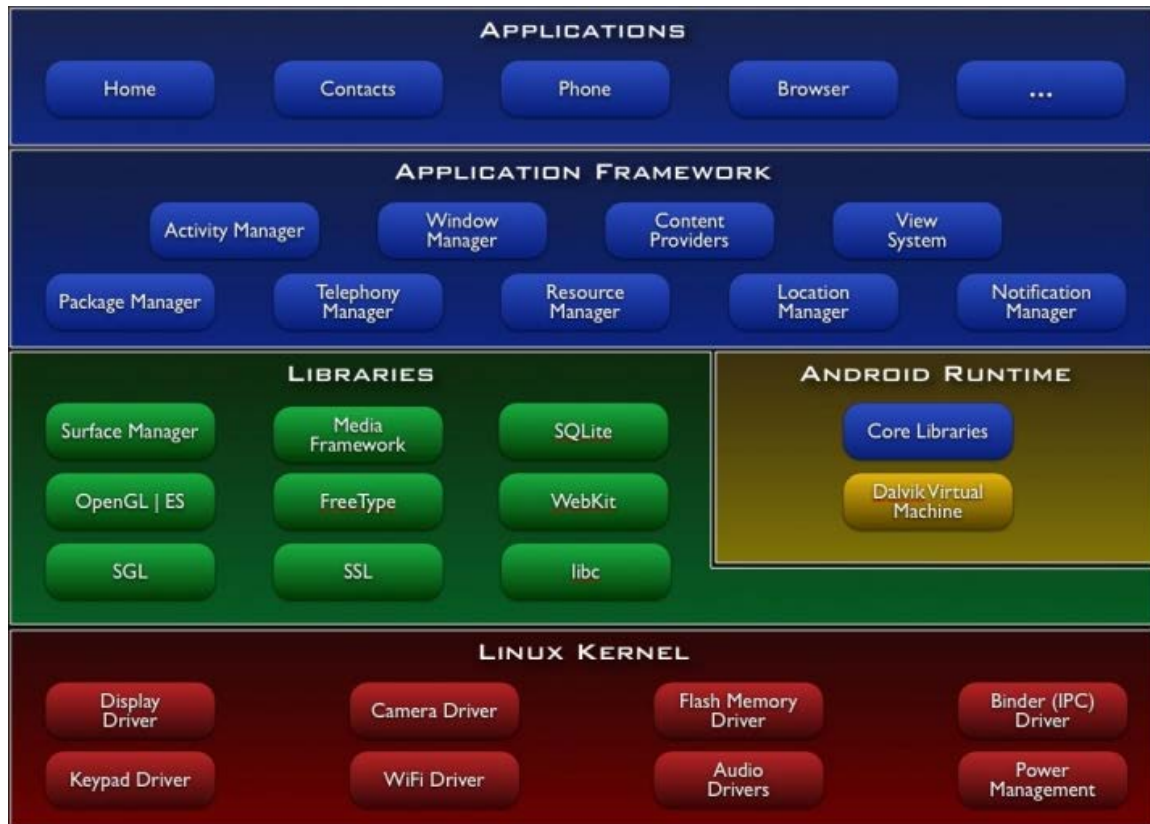


# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

### 2.2.1) A arquitetura do Android



**Arquitetura geral da plataforma**

### 2.2.2) Aplicações

O Android nos fornece um conjunto de aplicações fundamentais, são elas:

- um cliente de e-mail;
- programa de SMS;
- agenda;
- mapas;
- navegador;
- contatos entre outros.

Todos os aplicativos acima presentes no Android foram desenvolvidos na linguagem de programação Java.



O Android nos fornece um conjunto de bibliotecas C/C++ utilizadas por vários componentes do sistema. Veja algumas das bibliotecas abaixo:

- **System C library:** Consiste em uma implementação derivada da biblioteca C padrão baseado no sistema (libc) do BSD sintonizada para dispositivos rodando Linux.

- **Media Libraries:** Baseado no PacketVideo's OpenCORE; são as bibliotecas que suportam os mais diversos formatos de áudio e vídeo, incluindo também imagens.

- **Surface Manager:** Responsável pelo acesso ao subsistema de exibição bem como as múltiplas camadas de aplicações 2D e 3D;

- **LibWebCore:** Consiste em um web browser engine utilizado tanto no Android Browser quanto para exibições web.

- **SGL – o engine de gráficos 2D**

- **3D libraries:** Uma implementação baseada no OpenGL ES 1.0 APIs; As bibliotecas utilizam aceleração 3D via hardware (quando disponível) ou o software de renderização 3D altamente otimizado incluído no Android.

- **FreeType** – Biblioteca responsável pela renderização de fontes bitmap e vector;

- **SQLite** – Conforme já mencionado, consiste no sistema gerenciador de banco de dados (SGBD) relacional disponível para todas as aplicações.

### 2.2.3) Android Runtime

O Android é constituído por um conjunto de bibliotecas que fornece a maioria das funcionalidades disponíveis nas principais bibliotecas da linguagem Java. Toda aplicação Android roda em seu próprio processo, com sua própria instância da máquina virtual Dalvik. O Dalvik foi escrito de forma a executar várias VMs eficientemente. Ele executa arquivos .dex, que é otimizado para consumo mínimo de memória. A VM é baseada em registros e roda classes





# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

compiladas pela linguagem Java que foram transformadas em arquivos .dex, através da ferramenta “dx” incluída no SDK.









O Dalvik VM foi baseado no kernel do Linux para funcionalidades subjacentes como o encadeamento e a gestão de baixo nível de memória.

#### 2.2.4) Linux Kernel

O Android foi projetado em cima da versão 2.6 do kernel do Linux para os serviços centrais do sistema, tais como segurança, gestão de memória, gestão de processos, etc. O kernel também atua como uma camada de abstração entre o hardware e o resto do software.

#### 2.3) Versões da plataforma Android

Atualmente no mercado e nas lojas são vendidos os mais diversos dispositivos (Smartphones e Tablets) Android nas suas mais variadas versões. Vamos conhecer agora algumas versões do Android existentes abaixo:

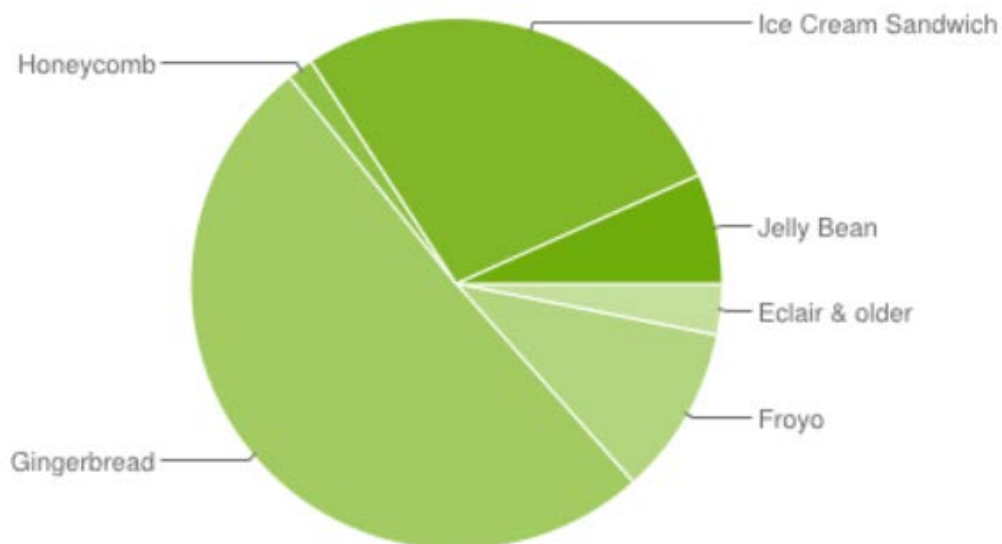
 <b>Android 1.5</b> <b>(Cupcake)</b> <b>(Abril - 2009)</b>	 <b>Android 1.6</b> <b>(Donut)</b> <b>(Setembro - 2009)</b>	 <b>Android 2.1</b> <b>(Eclair)</b> <b>(Janeiro - 2010)</b>	 <b>Android 2.2</b> <b>(Froyo)</b> <b>(Maio - 2010)</b>
 <b>Android 2.3.x</b> <b>(GingerBread)</b> <b>(Dezembro- 2010)</b>	 <b>Android 3.x</b> <b>(HoneyComb)</b> <b>(Janeiro - 2011)</b>	 <b>Android 4.0</b> <b>(Icecream sandwich)</b> <b>(Outubro - 2011)</b>	 <b>Android (4.1 / 4.2)</b> <b>(Jelly Bean)</b> <b>(Junho/Outubro- 2012)</b>

##### 2.3.1) Qual versão do Android utilizamos para desenvolver os jogos ?

No tópico anterior vimos que existem várias versões do Android presente na maioria dos dispositivos que encontramos hoje em dia. A pergunta que faço agora é : Para qual versão do Android irei desenvolver o meu jogo (aplicativo) ?



Para definirmos “de fato” para qual versão do Android desenvolver, precisamos saber quais as versões do Android são as mais utilizadas e presentes hoje em dia, conforme você pode conferir no gráfico abaixo (a pesquisa do gráfico abaixo foi finalizada em 03/12/2012).

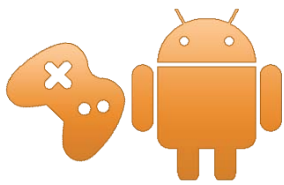


### **Versões mais usadas do Android**

Pelo que podemos ver no gráfico acima, a versão do Android mais utilizada e “predominante” é o Android 2.3 (Ginger Bread). As versões mais existentes do Android hoje em dia são as versões 2.2 (Froyo), 2.3 (Ginger Bread) e 4.0 (Icecream sandwich). Quando desenvolvermos uma aplicação, seja ela qual for (um jogo, utilitário e etc.), devemos pensar para qual versão do Android desenvolver e, saber se todos os recursos (em níveis de programação) que vamos utilizar se encontram “presentes” na versão do Android escolhida.

Neste material estaremos utilizando o Android 2.2 (Froyo) para a construção dos nossos jogos (através de um modelo de projeto de jogo que desenvolvi que pode ser conferido no Capítulo 4).





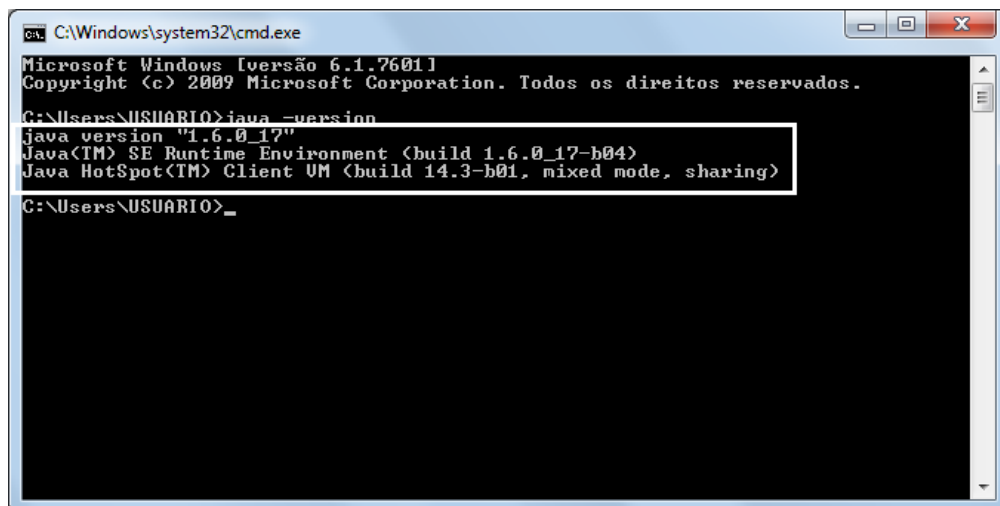
## Capítulo 3 Instalando e Configurando o Android Developer Tools

**P**ara a elaboração desse material eu fiz o uso do Android Developer Tools (que consiste no Eclipse 4.2 para Windows configurado com o Android SDK). Qualquer versão (de preferência superior) do programa citado acima serve. Para que toda essa aplicação funcione é necessário que você tenha instalado, antes de tudo, a Máquina Virtual Java (de preferência a versão 6 ou posterior). Bom, mãos a obra.

Para saber se você possui uma Máquina virtual Java entre no prompt de comando e digite a seguinte linha:

```
java -version
```

Se mostrar algo parecido como demonstra a figura seguinte:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [versão 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\USUARIO>java -version
java version "1.6.0_17"
Java(TM) SE Runtime Environment (build 1.6.0_17-b04)
Java HotSpot(TM) Client VM (build 14.3-b01, mixed mode, sharing)

C:\Users\USUARIO>
```

### Maquina Virtual Java instalada no computador

Significa que você possui uma máquina virtual Java instalada no seu computador, caso contrário, instale o JRE. Você pode fazer o download do Java pelo link abaixo:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

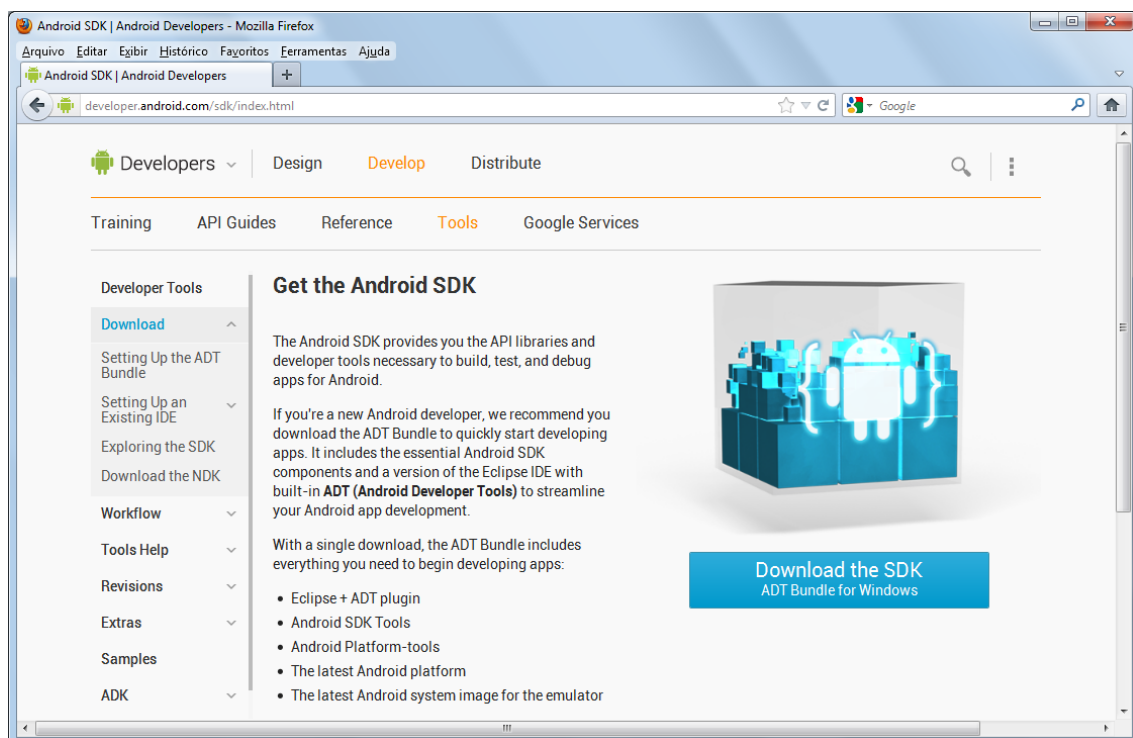


### 3.1) Efetuando o download e configurando o ADT

Conforme havia falado, a ferramenta que iremos utilizar aqui para a construção dos nossos jogos é o ADT (Android Developer Tools), que consiste no Eclipse configurado com o Android SDK. A linguagem que utilizaremos para criar as aplicações será a linguagem Java, na qual a ferramenta dá suporte. Para realizarmos o download desta ferramenta basta visitarmos o seguinte link abaixo:

<http://developer.android.com/sdk/index.html>

Depois de visitar o link será aberta a seguinte página, conforme demonstra a figura seguinte:



**Site do Android – Download do ADT**

Para efetuarmos o download do Android Developer Tools basta clicarmos no botão “Download the SDK”. Feito isso será aberta uma tela de contrato de “termos e condições”, onde nela você vai confirmar os termos de contrato e selecionar a versão sistema operacional (32 bits ou 64 bits). Depois de escolher o sistema operacional, basta clicar no botão “Download the SDK ADT Bundle for Windows” para efetuar o download da ferramenta.

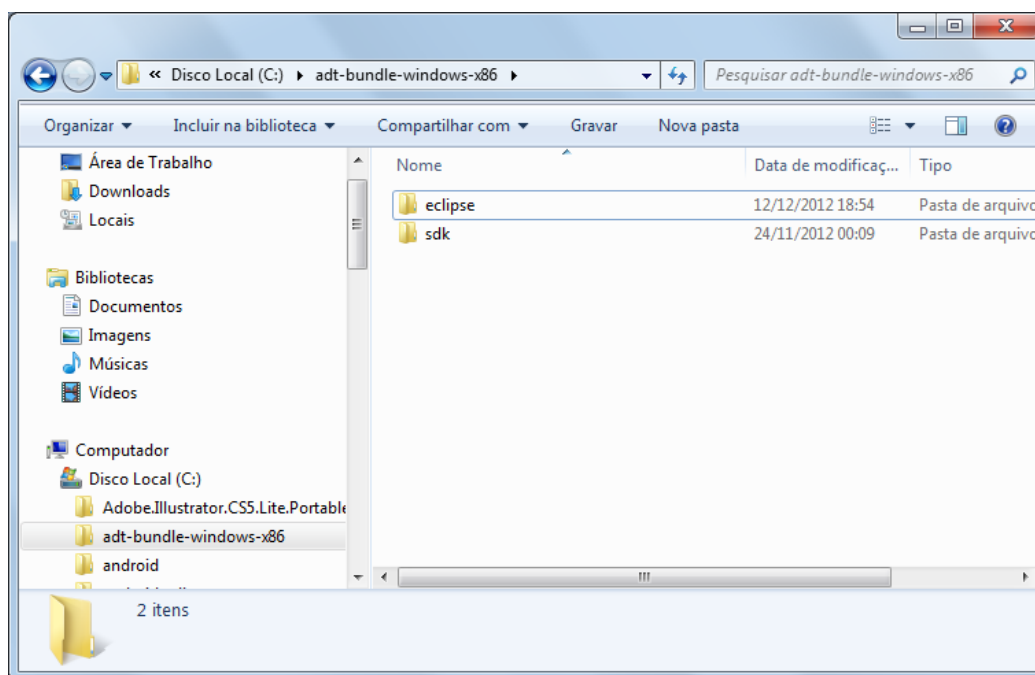


# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

A instalação da ferramenta é bastante simples (como no Eclipse), bastando descompactar o arquivo em um diretório desejado. O diretório de instalação do ADT para esse caso será o diretório “raiz” “C:\”. Ao descompactar o arquivo da aplicação, certifique-se de que foi gerado o diretório “C:\adt-bundle-windows-x86” (ou “C:\adt-bundle-windows-x86\_64”, caso o sistema escolhido seja de 64 bits) e que o mesmo apresenta o conteúdo demonstrado na figura abaixo:



### Conteúdo do diretório “adt-bundle-windows-x86”

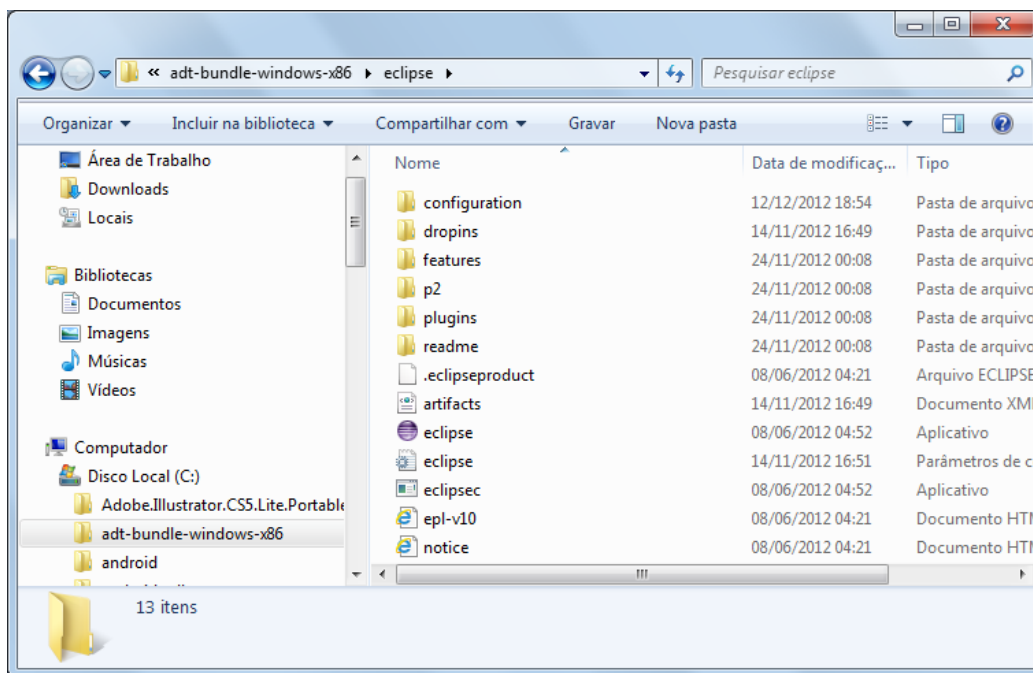
A ferramenta de desenvolvimento se encontra dentro do diretório “eclipse”, basta isso basta acessar o diretório “eclipse”. Feito isso, você verá o seguinte conteúdo:



# Apostila de Android

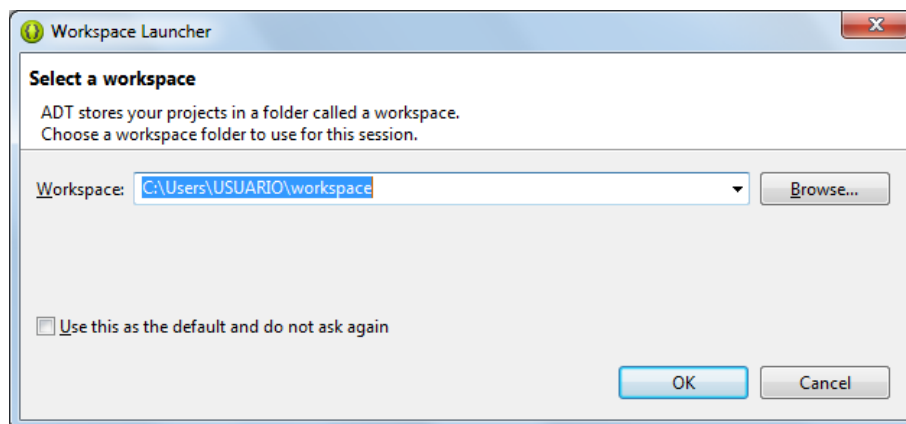
## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



#### Conteúdo do diretório “eclipse”

Para executarmos o Android Developer Tools basta clicar no ícone (de cor “roxa”) escrito “eclipse”. Quando a ferramenta é executada pela primeira vez, será solicitado um diretório de “Workspace” (diretório de trabalho), que é o local no qual o Eclipse vai gerar os projetos, conforme você confere na figura seguinte:



#### Diretório do Workspace

Escolha o diretório desejado para seu “Workspace” e , caso você deseje que ela seja definitiva, marque a opção “Use this as the default and do not ask again”. Depois disso clique em “OK”.

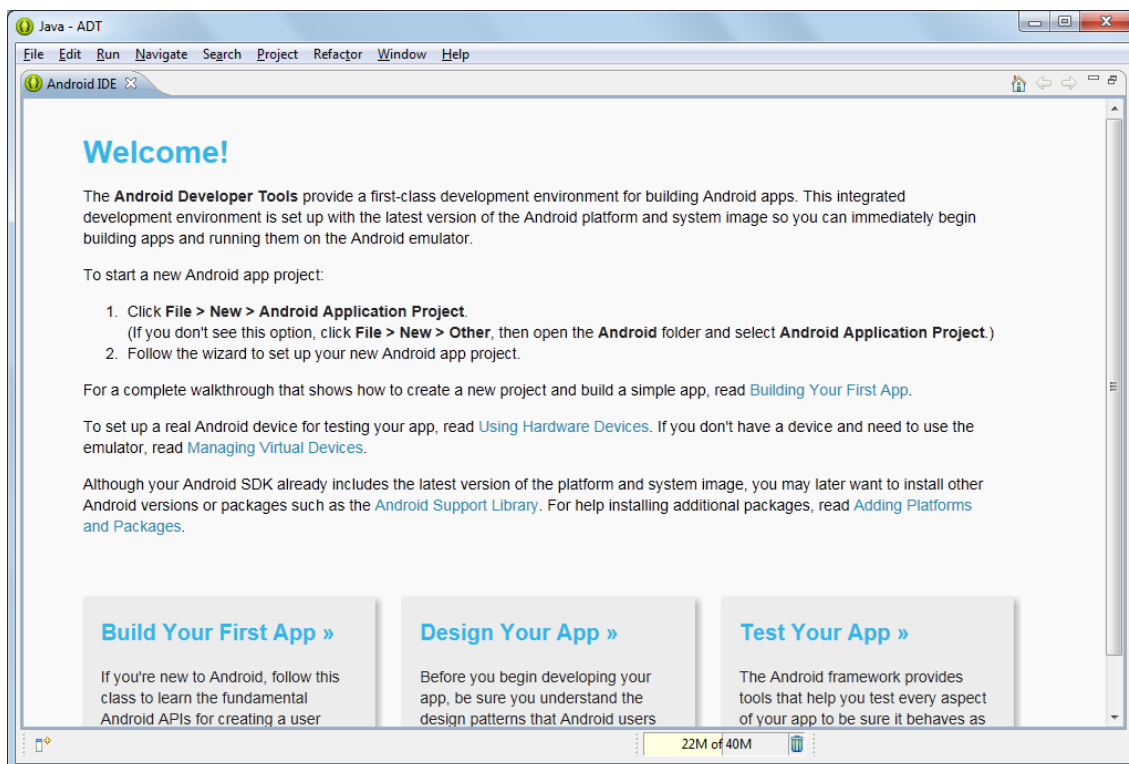
Feito isso, o ambiente de programação Eclipse será carregado, conforme demonstra a figura seguinte:



# Apostila de Android

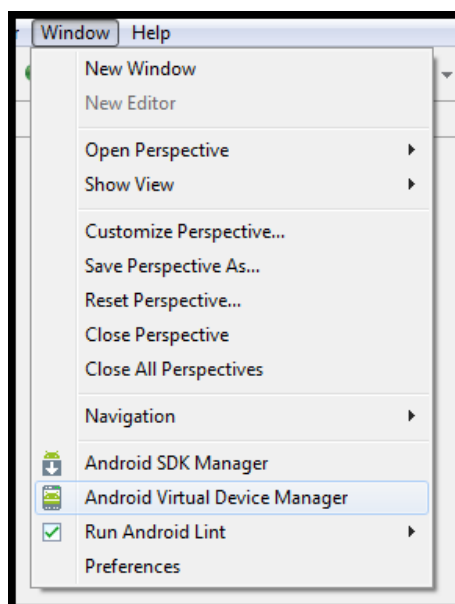
## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



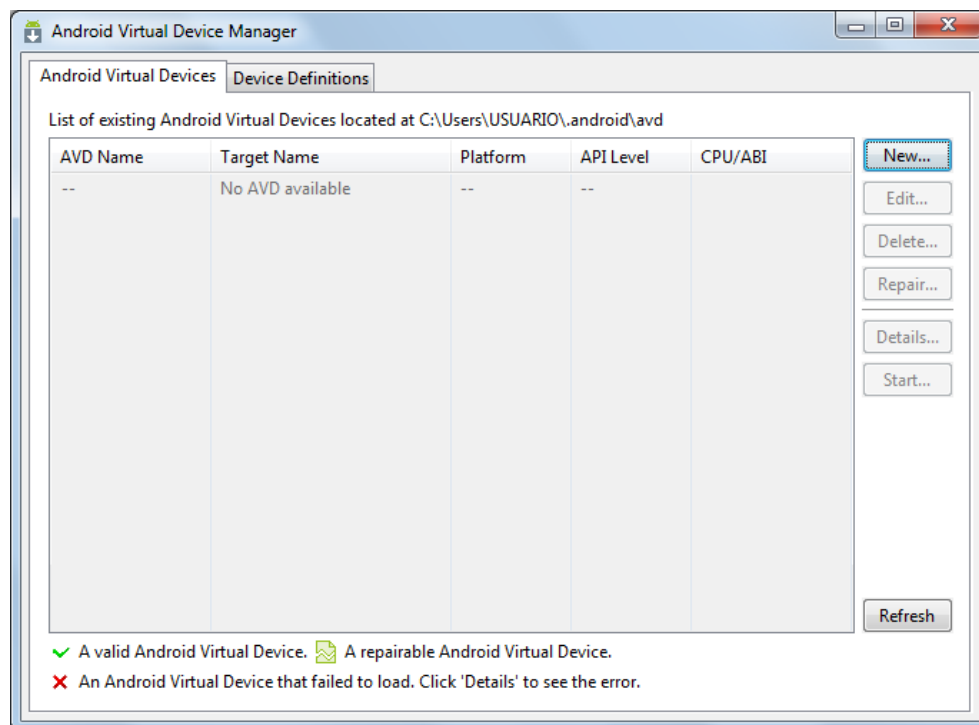
### Ambiente de programação Eclipse (ADT)

Agora vamos fechar a guia “Android IDE” para encerrarmos a tela de “Welcome!” e em seguida vamos configurar um dispositivo virtual (conhecido no Android como AVD ou “Android Virtual Device”) que usaremos para testar e executar as nossas aplicações (jogos). Para definirmos um AVD basta irmos no menu “Windows” / “Android Virtual Device Manager”, como mostra a figura abaixo:



### Android Virtual Device Manager

Feito isso, será aberta a seguinte tela abaixo:



**Caixa de diálogo – Android Virtual Device Manager**

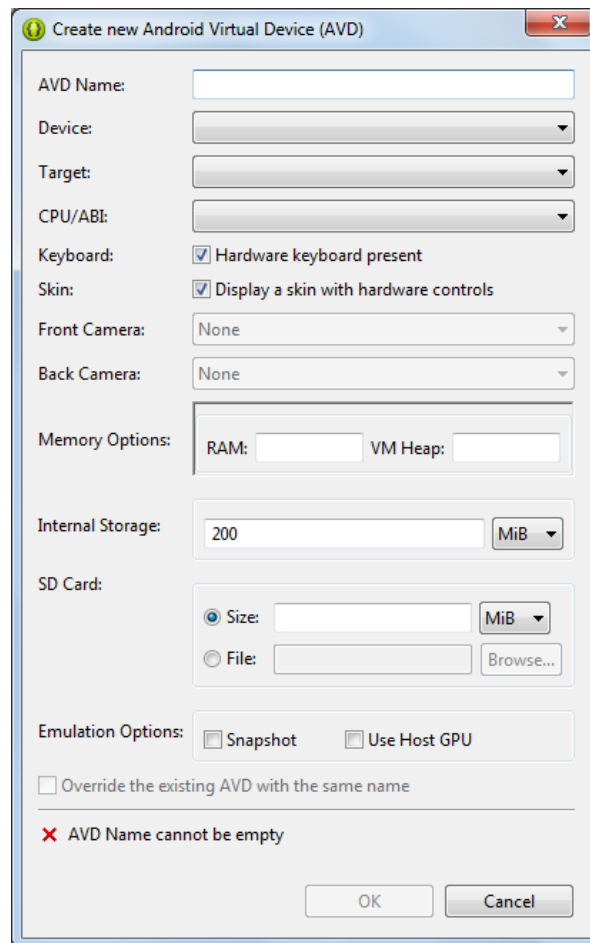
Para criarmos um dispositivo virtual clique no botão “New”, e em seguida será aberta uma tela conforme mostra a figura seguinte:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



**Caixa de diálogo - Create new AVD**

No Android Developer Tools já existe uma plataforma que já acompanha a ferramenta que é o Jelly Bean (Versão 4.2), logo, vamos utilizar ela.

Inicialmente, vamos configurar o básico pra executarmos a nossa aplicação. Em “AVD Name” você define o nome do AVD, vamos chamá-lo de “EmuladorGame”.

Em “Device” escolhemos o tipo de dispositivo e a sua resolução. Atualmente encontramos diversos dispositivos Android com os mais diversos tamanhos (resoluções) diferentes. A resolução que será utilizada para a construção dos nossos jogos será uma resolução “5.1 WVGA” (que é normalmente uma resolução de Tablet). Para o dispositivo vamos escolher a opção “5.1 WVGA (480 x 800 : mdpi)”.

Automaticamente quando selecionamos o tipo de dispositivo (Device), é definida a plataforma que será utilizada (a seção “Target” ). Como havia falado,





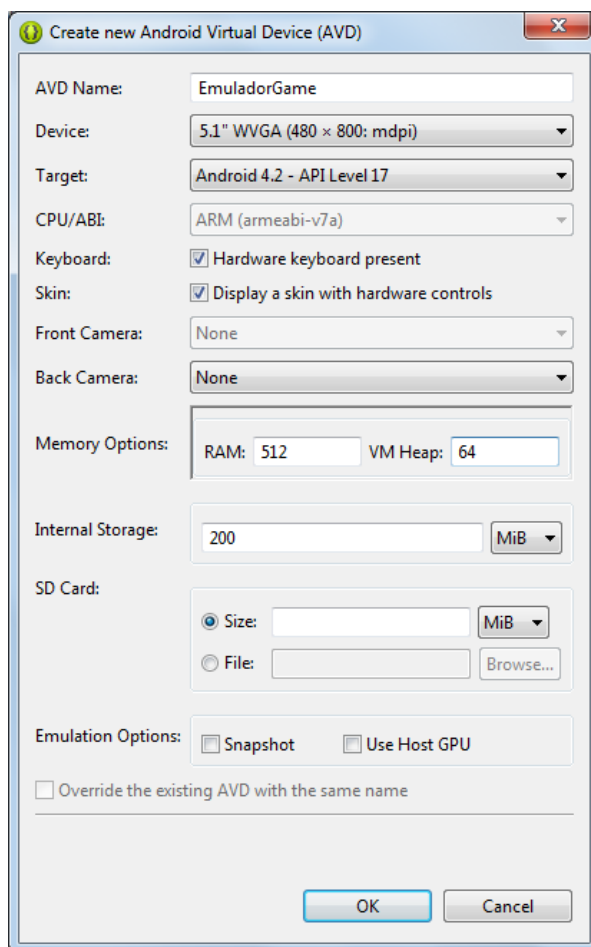
# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

essa versão acompanha a plataforma Jelly Bean 4.2, logo, a plataforma escolhida será a Jelly Bean (Android 4.2 – API Level 17).

Na seção “Memory Options” em “VM Heap” vamos definir o tamanho do “Heap” de 16 (que é o padrão) para 64 (estou definindo esse valor “bem alto” para evitar “estouro de pilha” durante a execução do jogo no emulador). Veja como ficou as configurações na figura abaixo:



**Caixa de diálogo - Create new AVD**

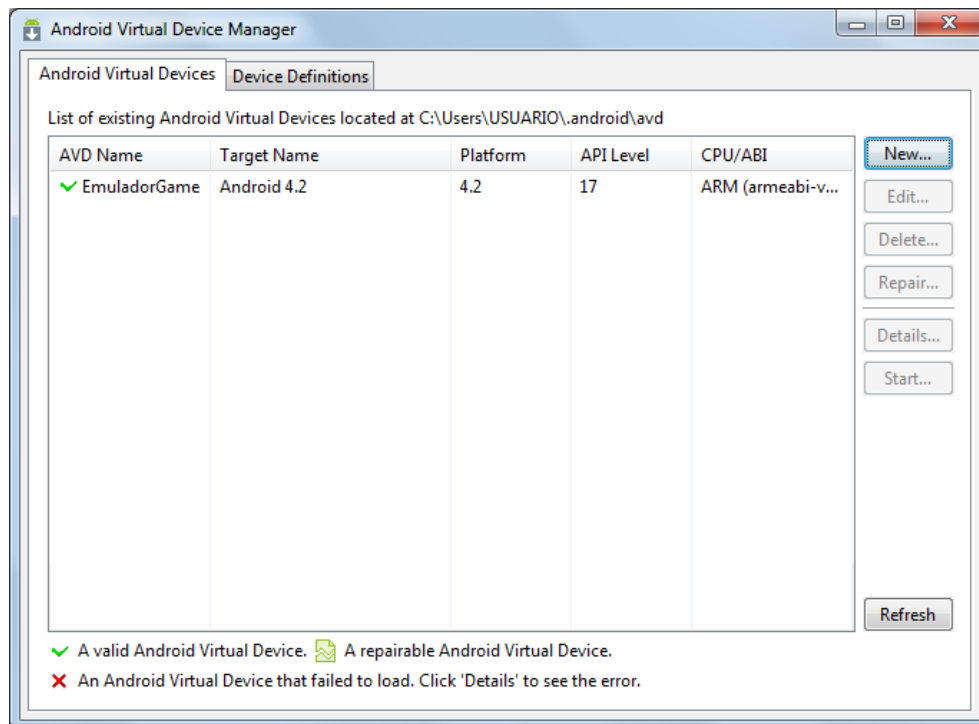
Para criarmos nosso AVD, clique no botão “OK” e pronto. O resultado você confere na figura seguinte:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



**Caixa de diálogo – Android Virtual Device Manager**

Bom, nosso ambiente de desenvolvimento está devidamente configurado. No próximo capítulo vamos aprender passo a passo como criar um projeto para Android voltado para jogos através de um modelo de projeto que eu mesmo desenvolvi.



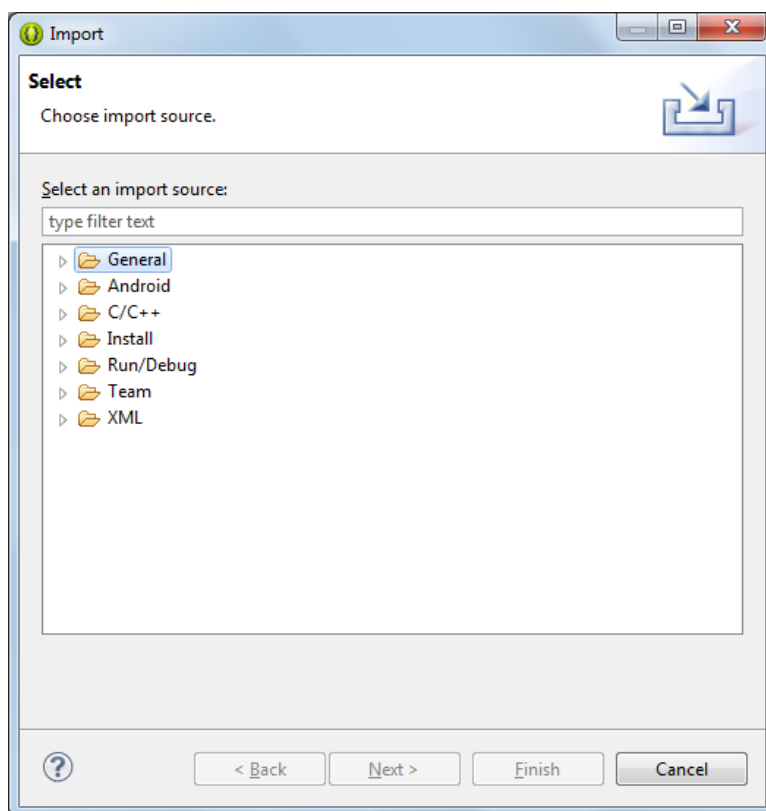
## Capítulo 4 desenvolver Android

## Começando a jogos para

**D**epois de tudo configurado e instalado, conforme foi mostrado no capítulo anterior, vamos a partir de agora começar a desenvolver nossos jogos para a plataforma Android.

Conforme havia falado, para criarmos os nossos jogos aqui no ADT, iremos utilizar um modelo de projeto que eu mesmo desenvolvi que vai facilitar muito o processo de desenvolvimento de jogos. Esse modelo de projeto já acompanha este material (que é o arquivo “Game Android Project V2\_2.zip”).

Para usarmos esse modelo de projeto precisaremos “importar” esse modelo para o ADT, para podermos desenvolver nossos jogos. Para importar o modelo basta irmos no menu “File” / “Import”. Feito isso, deverá ser aberta a seguinte tela conforme mostra a figura seguinte :



**Caixa de diálogo - Import**

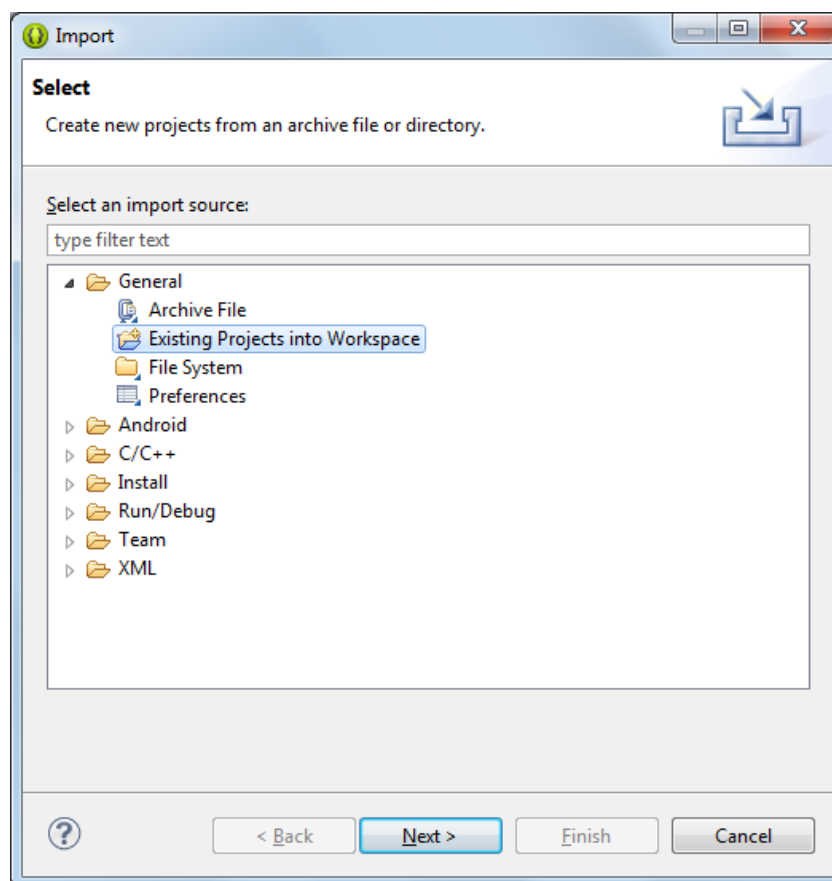


# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

Se observamos a caixa de diálogo acima, existem várias “pastas” onde em cada uma delas existe um conjunto de opções. Para importarmos um projeto vamos “expandir” a pasta “General” e em seguida iremos selecionar a opção “Existing Projects into Workspace”, conforme é exibido na figura abaixo:



**Caixa de diálogo - Import**

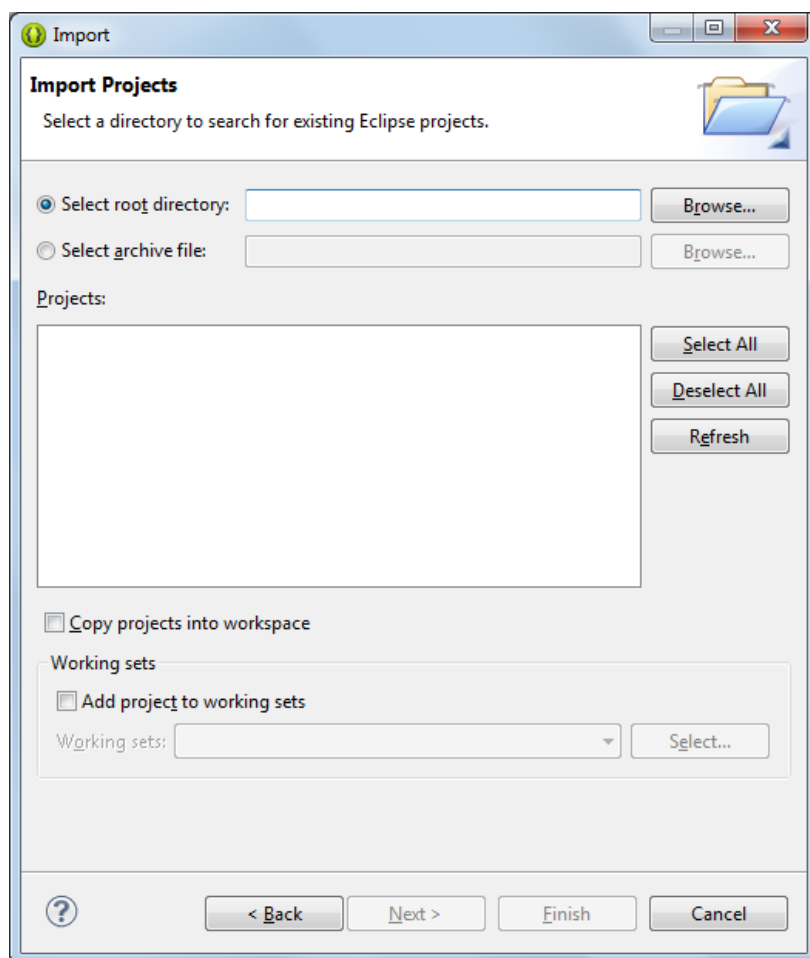
Clicando em “Next” avançamos para a próxima etapa, que consiste na seleção do projeto que será utilizado no ADT, conforme é mostrada na figura seguinte:



# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)



**Caixa de diálogo - Import**

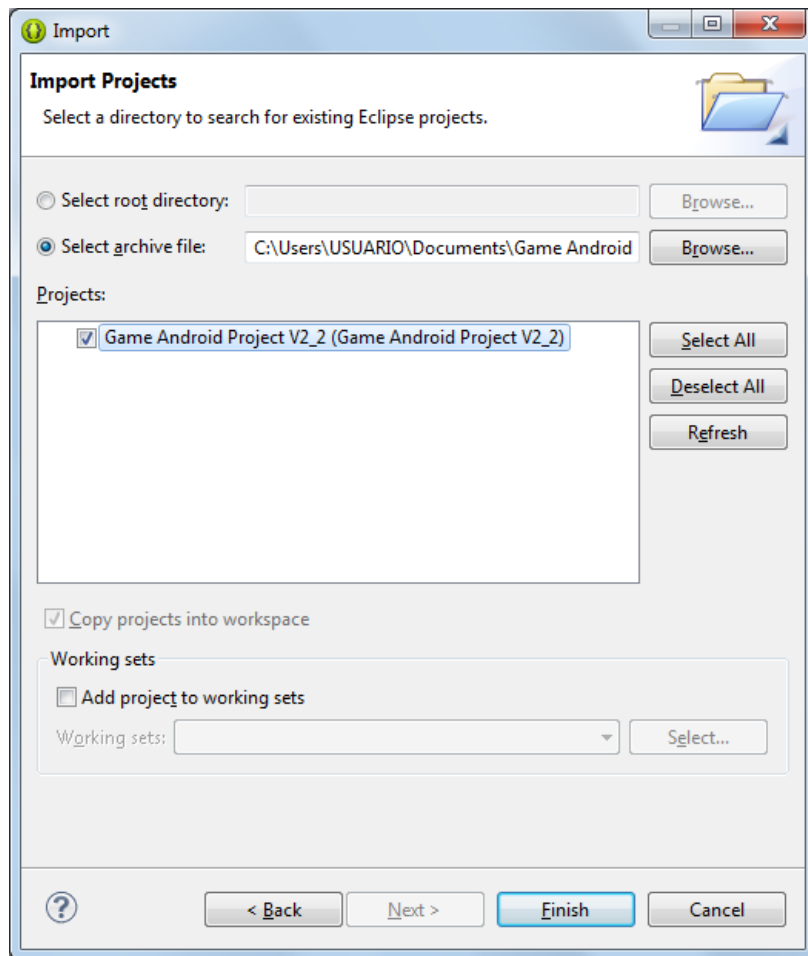
Na caixa de diálogo acima vamos selecionar a opção “Select archive file”, e em seguida vamos clicar no botão “Browse...” para selecionarmos o nosso modelo de projeto (o arquivo “Game Android Project V2\_2.zip”, que acompanha este material). Depois de selecionado o arquivo, na lista “Projects” vai estar sendo exibido o modelo de projeto que será utilizado no ADT, conforme você vê na figura seguinte:



# Apostila de Android

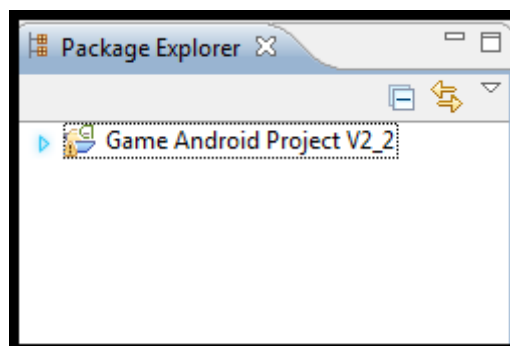
## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



**Caixa de diálogo - Import**

Para importarmos o nosso projeto basta clicarmos no botão “Finish” e pronto, já podemos utilizá-lo para construirmos os nossos jogos. Vejamos o nosso projeto no ADT, conforme mostra a figura seguinte:



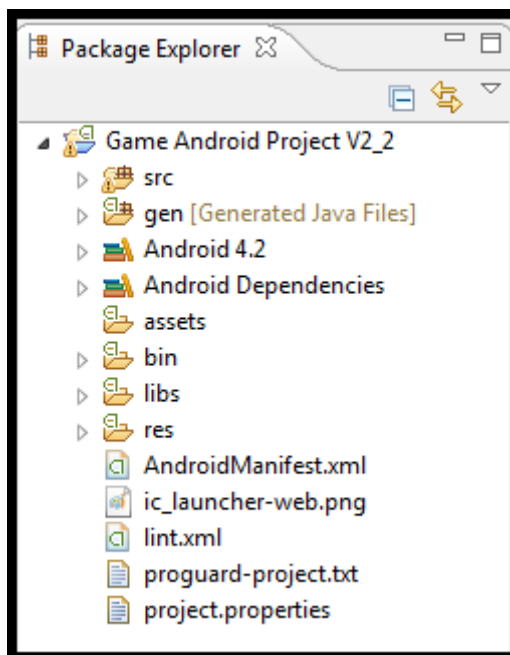
**Projeto importado para o ADT**



### 4.1) - Conhecendo a estrutura geral do projeto Android para jogos

Nós vamos utilizar para a construção dos nossos jogos para Android, um modelo de projeto (que eu mesmo desenvolvi) voltado para o desenvolvimento de aplicativos desta categoria. Vamos conhecer abaixo toda a estrutura (de arquivos e pastas) que compõem este projeto.

Se visualizarmos o lado esquerdo, existe uma seção chamada “Package Explorer”, onde nela existe uma pasta chamada “Game Android Project V2\_2” (que é o nosso projeto), constituído por vários subdiretórios, que por sua vez, possui seus respectivos arquivos. Se expandirmos a pasta do projeto, veremos uma estrutura de diretórios e pacotes conforme demonstra a figura seguinte:



**Package Explorer**

Irei comentar agora toda a estrutura de um projeto Android. Ele é composto por várias pastas e arquivos, cada um com uma finalidade em específico.

#### **O diretório “src” (source)**

Vamos começar conhecendo primeiramente o diretório “src” (source). Dentro desse diretório temos todos os arquivos de extensão “.java” que fazem parte de uma aplicação Android. Se nós expandirmos essa pasta encontraremos uma série de arquivos, todos eles organizações em seus respectivos “pacotes”, conforme você pode conferir na figura seguinte:

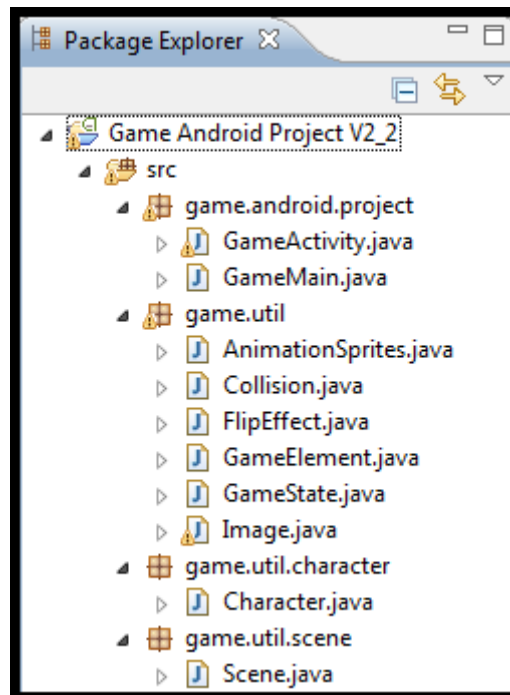




# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



**Package Explorer**

Irei explicar agora a finalidade de cada pacote e de seus respectivos arquivos:

#### O pacote “game.android.project”

Nesse pacote estão constituídos os arquivos principais, nos quais iremos escrever o código do nosso jogo. Irei falar a respeito desses arquivos, são eles:

**GameActivity.java:** Esse é o arquivo responsável por colocar o jogo em execução. Normalmente não mexemos nesse arquivo.

**GameMain.java :** Esse é o arquivo principal do projeto, é nele que entramos com o código do jogo.

Conforme vimos acima, o arquivo no qual iremos mexer para escrever nossos jogos é o arquivo “GameMain.java”. Se clicarmos duas vezes no arquivo, veremos o seguinte conteúdo:

```
package game.android.project;  
  
import android.content.Context;  
import android.graphics.Canvas;
```



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
import android.graphics.Color;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import game.util.*;

public class GameMain extends SurfaceView implements Runnable {

    GameState gameState;

    Thread thread = null;
    SurfaceHolder surfaceHolder;
    volatile boolean running = false;
    Context context;

    //Declare os atributos aqui abaixo

    public GameMain(Context context) {

        //Método construtor, disparado toda vez que o jogo a carregado na memória
        super(context);
        this.context = context;
        surfaceHolder = getHolder();

        //Insira o código aqui

        setFocusable(true);
    }

    @Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh) {
        /* Disparado inicialmente ao carregar o jogo e também
        * quando rotacionamos o dispositivo (retrato/paisagem)
        */
        super.onSizeChanged(w, h, oldw, oldh);

        //Insira o código aqui
    }

    @Override
    protected void onDraw(Canvas canvas) {

        //Desenha todos os elementos do jogo

        super.onDraw(canvas);

        Update();
    }
}
```



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
canvas.drawColor(Color.BLACK);
```

```
//Insira o código aqui
```

```
}
```

```
public void Update() {  
    /*Responsável pelo processamento da lógica do jogo  
    Insira o código aqui*/  
}
```

```
@Override  
public boolean onKeyDown(int keyCode, KeyEvent event) {  
    /* Disparado toda vez quando uma tecla é pressionada  
    *  
    * OBS : A instrução abaixo deste método deve ser a última a ser  
    * executada.  
    * Lembre-se de colocar seu código antes dela.  
    *  
    */  
  
    //Insira seu código aqui  
  
    return super.onKeyDown(keyCode, event);  
}
```

```
@Override  
public boolean onKeyUp(int keyCode, KeyEvent event) {  
  
    /* Disparado toda vez quando uma tecla é liberada  
    *  
    * OBS : A instrução abaixo deste método deve ser a última a ser  
    * executada.  
    * Lembre-se de colocar seu código antes dela.  
    *  
    */  
  
    //Insira seu código aqui  
  
    return super.onKeyUp(keyCode, event);  
}
```

```
@Override  
public boolean onTouchEvent(MotionEvent event) {  
  
    /* Disparado toda vez quando ocorre um toque na tela  
    *  
    * OBS : A instrução abaixo deste método deve ser a última a ser  
    * executada.  
    * Lembre-se de colocar seu código antes dela.  
    *  
    */  
}
```



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
*/  
  
//Insira seu código aqui  
  
return true;  
}  
  
public void onExitGame() {  
  
    /*Disparado quando o jogo é encerrado  
    Insira o código do jogo aqui*/  
  
}
```

O código acima é a estrutura padrão da classe **GameMain** (arquivo “GameMain.java”) no qual iremos escrever o código para o desenvolvimento do nosso jogo. Essa classe é constituída pelos seguintes métodos:

**GameMain:** Esse é o método construtor, executado toda vez que o jogo é executado. São neles que carregamos todos os objetos e iniciamos as variáveis que serão utilizados no jogo. Veja sua sintaxe:

```
public GameMain(Context context)
```

**onSizeChanged:** Esse é o método é executado toda vez que alteramos a orientação do dispositivo (de retrato para paisagem ou vice-versa). Também esse método é executado toda vez que uma aplicação é carregada na memória, na primeira vez que o jogo é executado. Veja sua sintaxe:

```
protected void onSizeChanged(int w, int h, int oldw, int oldh)
```

Irei explicar agora a finalidade de cada parâmetro acima:

w : Retorna a largura atual da tela, em pixels.

h : Retorna a altura atual da tela, em pixels.

oldw : Retorna a antiga largura da tela, antes da mudança da orientação do dispositivo.



**oldh** : Retorna a antiga altura da tela, antes da mudança da orientação do dispositivo.

**Update:** Esse método é executado sempre durante a execução do jogo, e também é o primeiro a ser executado. Ele é responsável por executar a “lógica” de processamento do jogo (movimento, ações e etc.). Vejamos a sua sintaxe:

```
public void Update()
```

**onDraw:** Esse método é executado sempre durante a execução do jogo, e executado “alternadamente” com o método **Update**. Ele é responsável por desenhar a tela do jogo, ou seja, deseja todos os elementos na tela do dispositivo. Vejas a sua sintaxe:

```
protected void onDraw(Canvas canvas)
```

Esse método possui um parâmetro chamado “canvas”, que representa a tela do dispositivo, no qual vamos utilizar para desenhar os elementos do jogo.

**onKeyDown:** Esse método é disparado toda vez que pressionamos uma das teclas do computador (caso estejamos rodando o jogo no emulador do Android) ou quando pressionamos uma tecla do Smartphone/Tablet (caso o mesmo ofereça suporte a esse dispositivo de entrada).

Atualmente os aparelhos Android já não vêm mais com esse periférico, pois, a maioria deles trabalha com o sistema de “touch” (toque). Nessa apostila iremos desenvolver os jogos “focando” o sistema de “touch screen” como periférico. Vejamos a sua sintaxe:

```
public boolean onKeyDown(int keyCode, KeyEvent event)
```

O parâmetro “keyCode” retorna o código da tecla “pressionada” e o argumento “event” guarda todas as informações do evento ocorrido.



**onKeyUp:** Esse método é disparado toda vez que liberamos uma da tecla anteriormente pressionada do computador (caso estejamos rodando o jogo no emulador do Android) ou do Smartphone/Tablet (caso o mesmo ofereça suporte a esse dispositivo de entrada).

Atualmente os aparelhos Android já não vêm mais com esse periférico, pois, a maioria deles trabalha com o sistema de “touch” (toque). Nessa apostila iremos desenvolver os jogos “focando” o sistema de “touch screen” como periférico. Vejamos a sua sintaxe:

```
public boolean onKeyUp(int keyCode, KeyEvent event)
```

O parâmetro “keyCode” retorna o código da tecla “liberada” e o argumento “event” guarda todas as informações do evento ocorrido.

**onTouchEvent:** Esse método é disparado toda vez quando ocorre um “touch screen”, ou seja, quando tocamos a tela do dispositivo. Esse método dispara ou quando “pressionamos” (tocamos) na tela com o dedo, ou quando arrastamos o dedo pela tela ou quando tiramos o dedo do dispositivo. Vejamos a sua sintaxe:

```
public boolean onTouchEvent(MotionEvent event)
```

O parâmetro “event” guarda todas informações do evento, como as coordenadas onde ocorreu o toque na tela do dispositivo, e também o tipo do toque que ocorreu (pressionamento, movimento (drag) ou liberação).

**onExitGame:** Esse método é disparado quando o jogo (aplicativo) é encerrado. Ele é bastante útil para finalizarmos alguma atividade antes do jogo acabar como o encerramento de uma música ou para liberar objetos da memória e etc.



### O pacote “game.util”

Nesse pacote estão contidos as classes básicas que iremos utilizar para a construção dos nossos jogos. Vejamos abaixo quais são essas classes e a finalidade de cada um (que será demonstrada nos capítulos posteriores) :

**Image.java** : Esse arquivo (ou classe) é utilizado quando queremos exibir uma imagem na tela em um jogo.

**AnimationSprites.java**: Esse arquivo (ou classe) é utilizado para realizar animações de sprites em um jogo.

**Collision.java** : Esse arquivo (ou classe) é utilizado para “detectar” colisões entre dois elementos em um jogo.

**FlipEffect.java** : Classe utilizada para “inverter” uma imagem (ou um conjunto de imagens de uma animação de sprites) na “horizontal”.

**GameElement.java**: Essa é a base utilizada para a construção da maioria das classes citadas acima (e de algumas outras classes que serão citadas em seguida), e de bastante utilidade para a construção de outros elementos do jogo por parte do usuário.

**GameState.java**: Essa classe possui somente uma finalidade : guardar o status do jogo, ou seja, se ele está no modo “pausa” ou no modo “play”.

### O pacote “game.util.character”

Esse pacote nada mais é do que uma “extensão” do pacote “game.util”. Dentro dele existe a seguinte classe:

**Character.java** : Esse arquivo (ou classe) é utilizado para a construção de personagens, que iremos utilizar em um jogo.

### O pacote “game.util.scene”

Esse pacote nada mais é do que uma “extensão” do pacote “game.util”. Dentro dele existe a seguinte classe:

**Scene.java** : Esse arquivo (ou classe) é utilizado para representar uma “cena” do jogo, e quais elementos estarão presentes nele.





### O diretório “res” (resources)

Dentro da estrutura do projeto Android existe um diretório chamado “res” (resources), onde dentro dele colocamos todos os recursos que podemos utilizar em um jogo no Android (como imagens, sons, músicas e etc). Vamos conhecer os subdiretórios existentes dentro dessa pasta e a finalidade de cada um deles.

### Os diretórios “drawable-hdpi”, “drawable-mdpi” , “drawable-ldpi” e “drawable-xhdpi”.

A partir da versão 1.6 da plataforma Android foram adicionados dentro da pasta “res” (resources) três diretórios responsáveis pelo gerenciamento de imagens: “drawable-ldpi”, “drawable-mdpi”, “drawable-hdpi”. Nas versões do Android 2.2 em diante , podemos encontrar dentro da pasta do projeto (além dos diretórios citados), também o diretório “drawable-xhdpi” . Todos esses diretórios possuem a mesma finalidade : armazenar imagens que serão utilizadas por uma aplicação em Android. Agora , qual a diferença que existe entre esses diretórios ? Cada um desses diretórios será utilizado de acordo com a resolução do dispositivo Android em uso, ou seja, qual modelo de emulador/dispositivo que você estiver usando.

Quando a gente configura nosso emulador podemos ver que temos disponível várias resoluções : HVGA, QVGA, WVGA e etc. Por exemplo, uma resolução QVGA (320x240) possui uma densidade baixa (low density – ldpi), logo, se você criar uma aplicação para dispositivos desse tipo de configuração, o diretório que você utilizará para guardar suas imagens será o “drawable-ldpi”.

Uma resolução HVGA (320x480) possui densidade média (medium density – ldpi), logo, se você criar uma aplicação para dispositivos desse tipo de configuração, o diretório que você utilizará para guardar suas imagens será o “drawable-mdpi”.

Uma resolução WVGA (800x480 , resolução de um Tablet por exemplo) possui densidade alta (high density – hdpi), porém, há algumas resoluções WVGA que suportam densidade média (medium density – mdpi).

Para os jogos que iremos desenvolver daqui por diante, vamos trabalhar com a resolução WVGA (como definimos na configuração do emulador), com densidade média (mdpi) , logo, todas as imagens que vamos utilizar no jogo serão adicionadas dentro do diretório “drawable-mdpi”.



### O diretório “values”

Um dos recursos que o Android permite que usemos na construção de aplicações são “constantes”. Como bom programador que todos nós somos, sabemos que uma “constante” nada mais é do que um endereço de memória que vai armazenar um determinado valor, que será único até o fim da execução do programa. Mas, o que isso tem a ver com as telas da nossa aplicação ? Vou explicar.

Normalmente usamos constantes para armazenar valores fixos que, naturalmente, estarão presentes na maior parte do seu programa. É muito comum criarmos um título como nome de uma aplicação (como por exemplo : “Alpha Software”), que vai estar presente na maioria das telas (ou em todas) da sua aplicação. Imagine você digitar “manualmente” o nome de seu software em todas as telas da sua aplicação, seria um processo trabalhoso você não acha ? Agora imagine “futuramente” que você precise “mudar” o nome de sua aplicação ? Você terá novamente o mesmo trabalho para digitar em cada tela o nome da nova aplicação, sem dúvida. Usando constantes, você não terá esse trabalho.

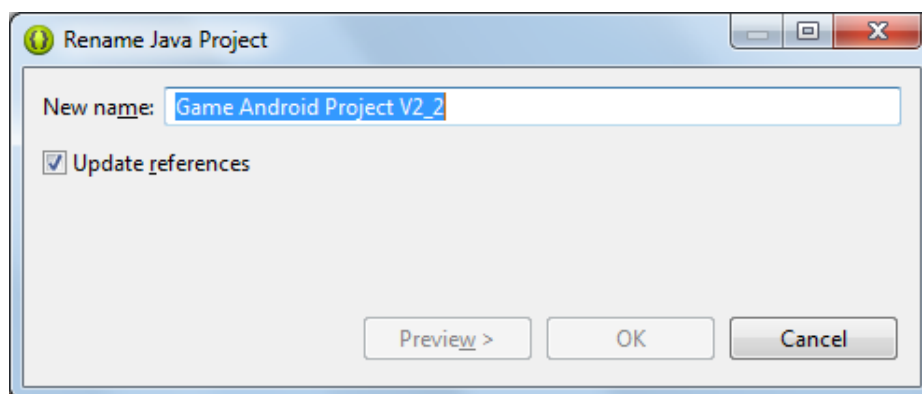
Dentro dessa pasta temos os seguintes arquivos:

- **“strings.xml”** : Esse arquivo guarda constantes relacionadas à nossa aplicação em geral (como o nome da aplicação, o título que vai aparecer na aplicação e etc.).
- **“dimen.xml”** : Esse arquivo guarda constantes relacionadas as dimensões que podemos utilizar em nossa aplicação.
- **“styles.xml”** : Esse arquivo guarda constantes relacionadas à estilos que podemos utilizar em nossa aplicação.



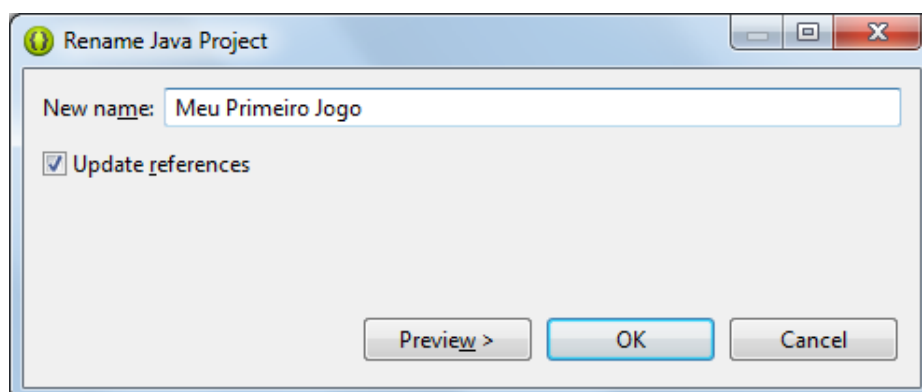
### Definindo o nome de um projeto

Como você percebeu, o nome do nosso projeto no Eclipse (ADT) se chama “Game Android Project V2\_2”, mas, se eu quiser um outro nome para esse projeto (como por exemplo “Meu Primeiro Jogo”), visto que todos os nossos projetos que iremos criar será a partir deste modelo ? Para darmos um novo nome para o projeto, basta selecioná-lo e em seguida pressionar a tecla “F2”, para renomear o projeto com um novo nome, conforme você pode conferir na figura abaixo:



**Caixa de diálogo – Rename Java Project**

Agora vamos renomear o nosso projeto com o seguinte nome : “Meu Primeiro Jogo”. Confira como ficou na figura seguinte:



**Caixa de diálogo – Rename Java Project**

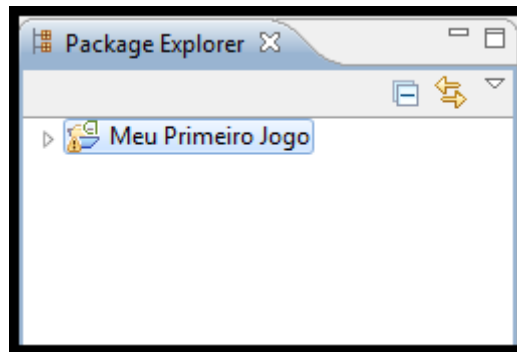
Depois de escrever o novo nome do projeto basta clicar em “OK” para que o projeto seja renomeado. Confira o resultado na figura seguinte:



# Apostila de Android

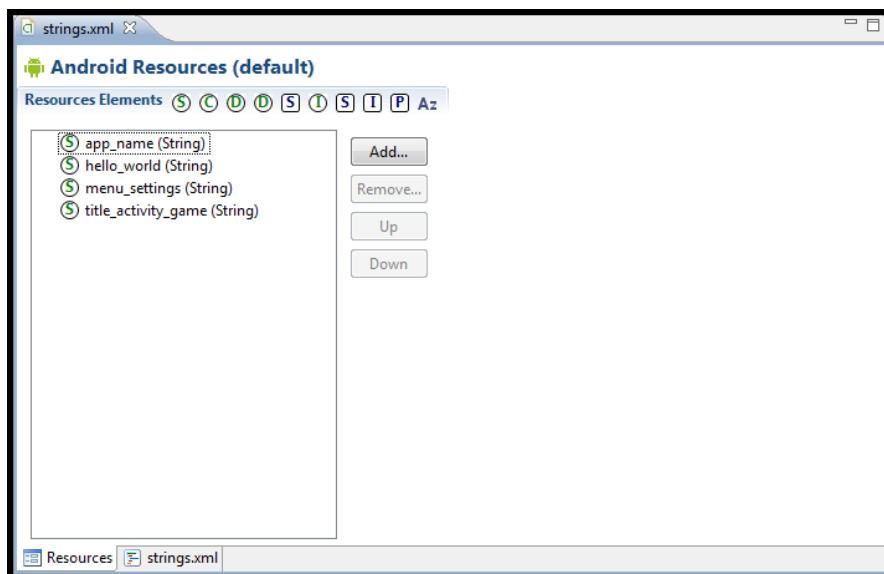
## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



**Projeto renomeado**

Se lembra do arquivo “strings.xml”, que é responsável por guardar constantes de uma aplicação ? Pois bem, uma das constantes desse arquivo armazena o nome do projeto e uma outra constante armazena o nome da aplicação (jogo). Vamos abrir o conteúdo desse arquivo, e o resultado você confere na figura seguinte:



**Conteúdo do arquivo “strings.xml”**

Se nós observarmos acima, temos as seguintes constantes:

**app\_name** : Apesar do nome (“app\_name”, de “Application Name”), essa constante guarda no nome do projeto.

**title\_activity\_game** : Essa constante armazena o “nome da aplicação” (ou melhor, o título da minha atividade que representa a aplicação).

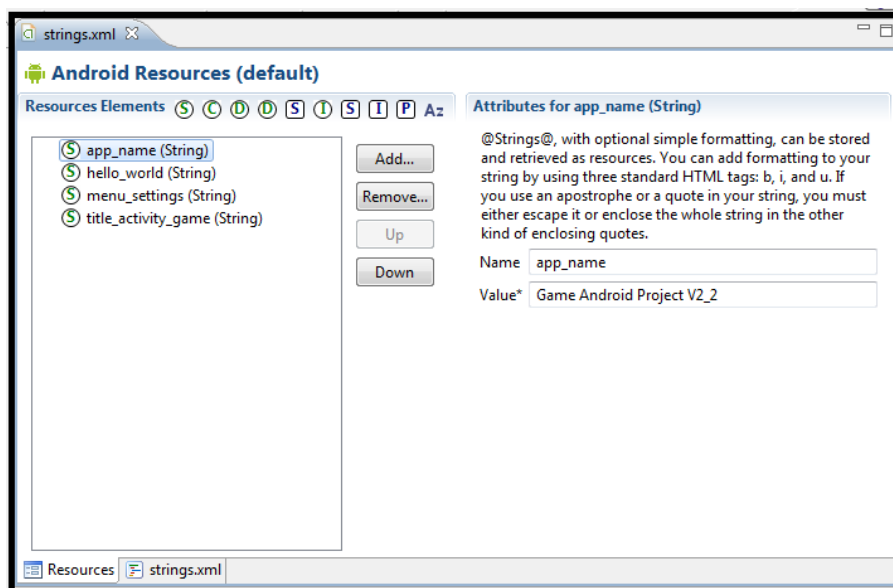
Para visualizarmos o conteúdo de uma das constantes acima, basta selecionar a constante desejada. Veja na figura seguinte:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



### Constante “app\_name” selecionada

Quando selecionamos uma constante, são exibidos dois campos:

Name : Guarda o nome da constante (podendo o mesmo ser alterado)

Value : Armazena o valor que a constante está assumindo (podendo o mesmo ser alterado)

Para essa situação, estaremos alterando o conteúdo que a constante está armazenando. A constante “app\_name” armazena o valor “Game Android Project V2\_2”. Vamos substituir o valor armazenado por essa constante pelo seguinte nome : “Meu Primeiro Jogo”.

Se selecionarmos a constante “title\_activity\_name”, ela estará armazenando o conteúdo “Game Android”. Vamos substituir o valor armazenado por essa constante pelo seguinte nome : “Meu Primeiro Jogo”.

Feito isso, vamos salvar as alterações feitas nesse arquivo (pressionando as teclas CTRL+S).

### Definindo o ícone da aplicação (jogo)

Para definirmos o ícone da aplicação , devemos escolher uma imagem que vá representá-la (normalmente se usa uma imagem de extensão PNG de dimensão 48x48 pixels).

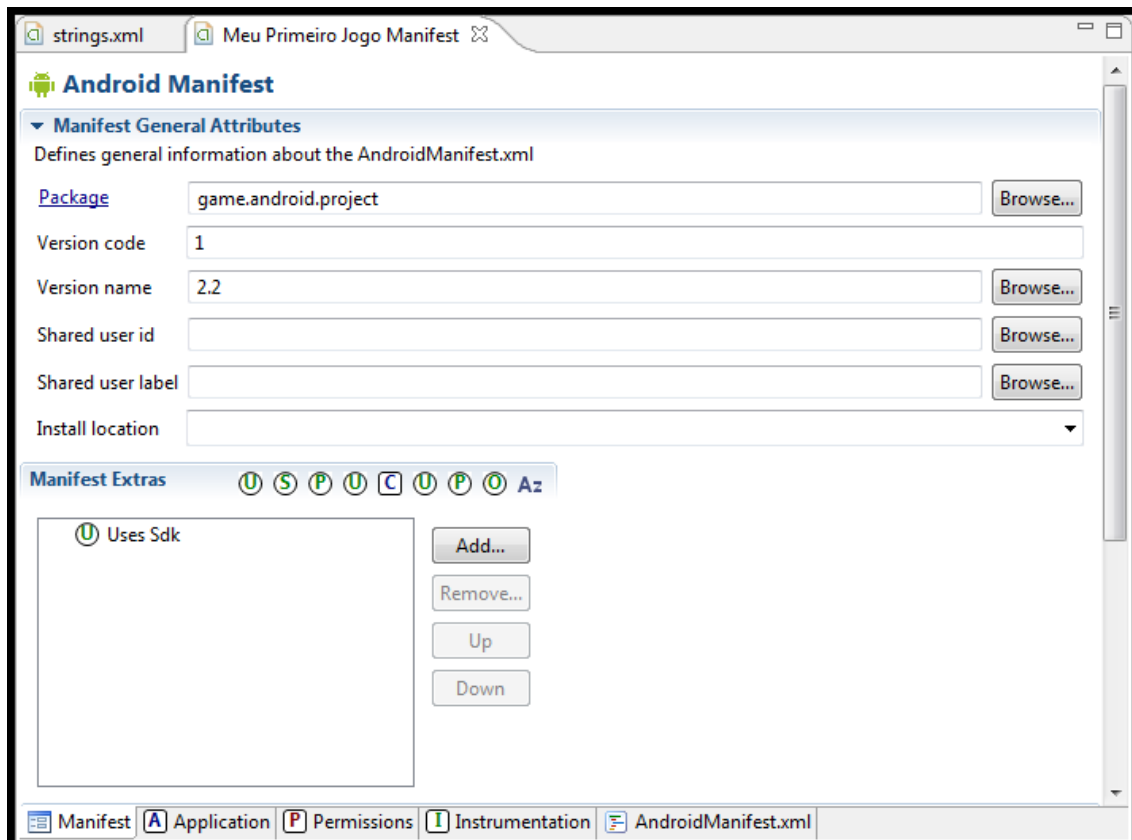


# Apostila de Android

## Programando Passo a Passo

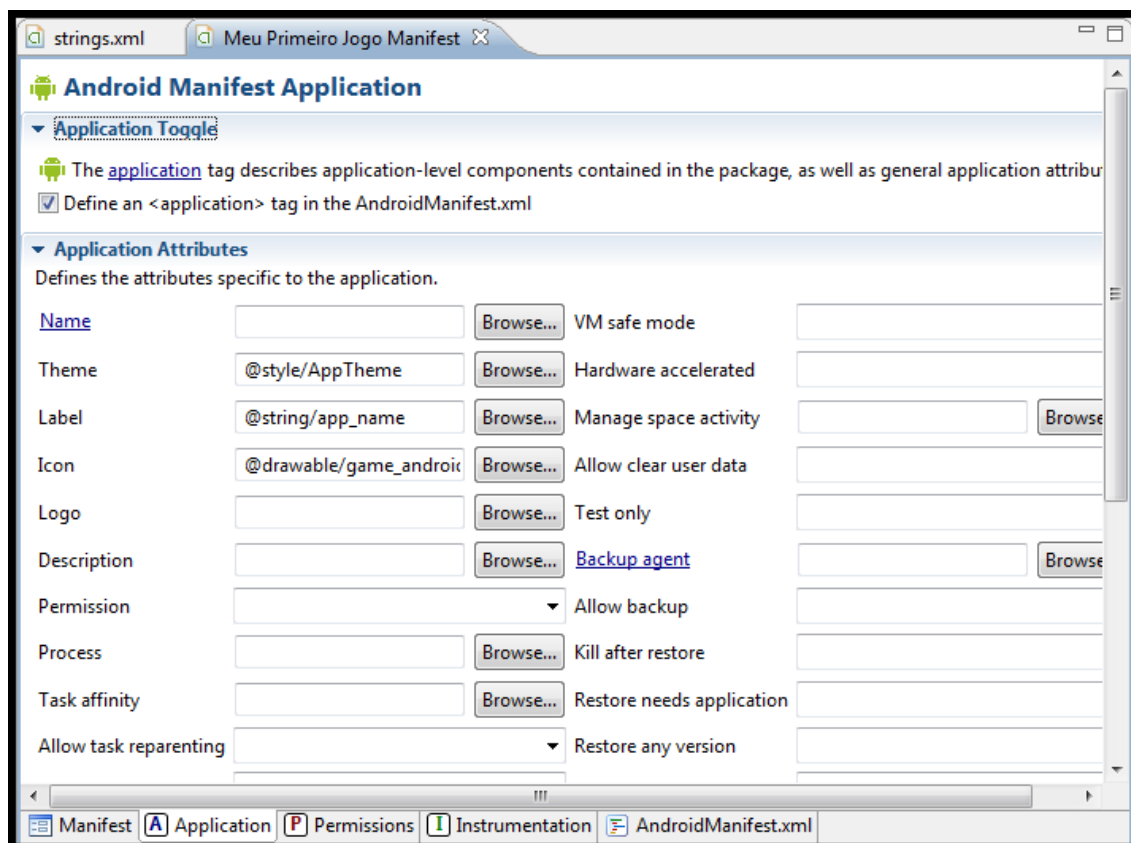
Desenvolvimento de Jogos (Edição Completa)

Por padrão, esse modelo já possui um ícone padrão que vai representar a aplicação, mas, você pode alterar esse ícone. Para definirmos a imagem que vai ser o ícone da aplicação, devemos fazer algumas configurações no arquivo “AndroidManifest.xml” (esse arquivo é considerado o “sistema nervoso” do Android, é nele que definimos as principais características e permissões da nossa aplicação). Vamos dar um duplo clique no arquivo, e o mesmo será aberto conforme mostra a figura seguinte:



**Conteúdo do arquivo “AndroidManifest.xml”**

Se você observar a figura acima, existe uma “guia” chamada “Application”, vamos clicar nela para carregar essa seção, conforme demonstra a figura seguinte:



### Conteúdo do arquivo “AndroidManifest.xml” – Seção “Application”

Se nós observamos acima existe um campo chamado “Icon”, é nele que definimos o ícone da nossa aplicação (jogo). Lembre-se, como o ícone vai ser uma imagem que vamos utilizar, todas as imagens que estaremos utilizando na nossa aplicação devem estar dentro do diretório “drawable-mdpi”. Após definir o ícone da aplicação, salve o arquivo “AndroidManifest.xml” para que as alterações tenham efeito.

Para esse exemplo (e os posteriores), vamos utilizar o ícone padrão do projeto.

### Executando o jogo

Para executarmos o nosso jogo feito no Android, basta clicarmos com botão direito sobre o projeto (Meu Primeiro Jogo) e em seguida selecionar a opção “Run As”, feito isso selecione “Android Application”, conforme demonstra a figura seguinte:

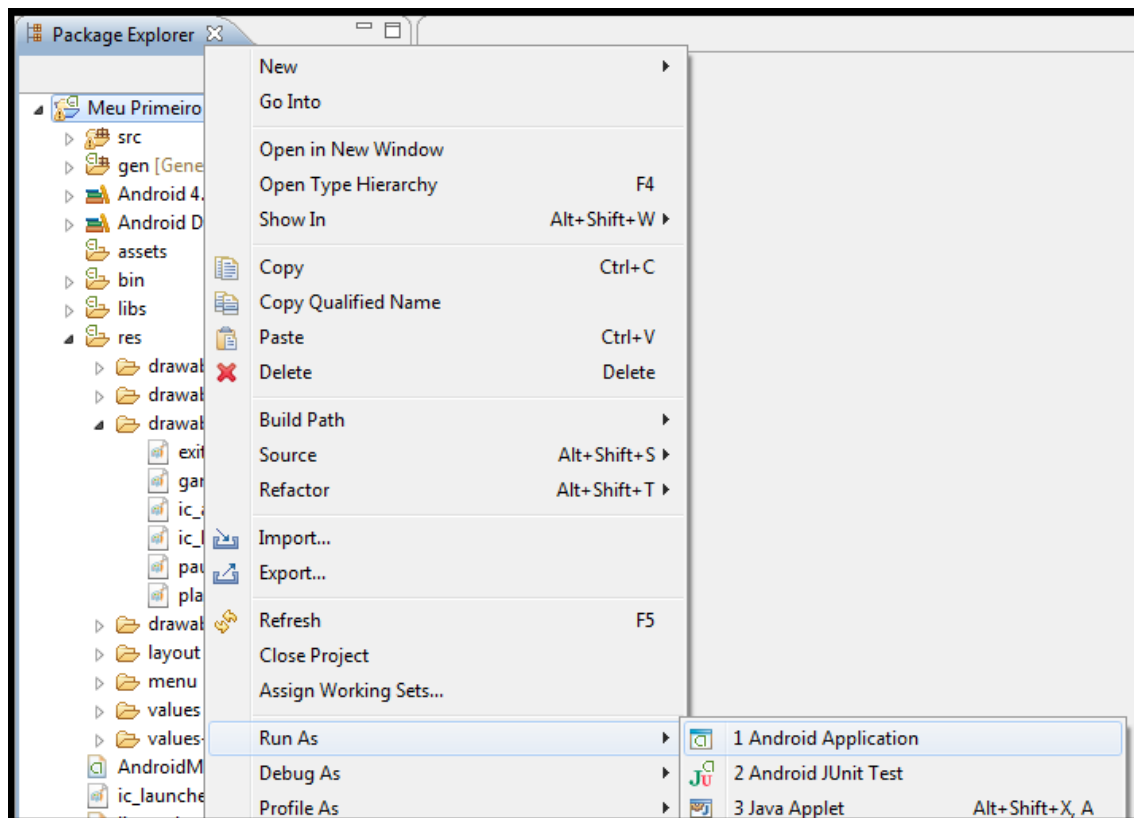




# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



### Executando a aplicação

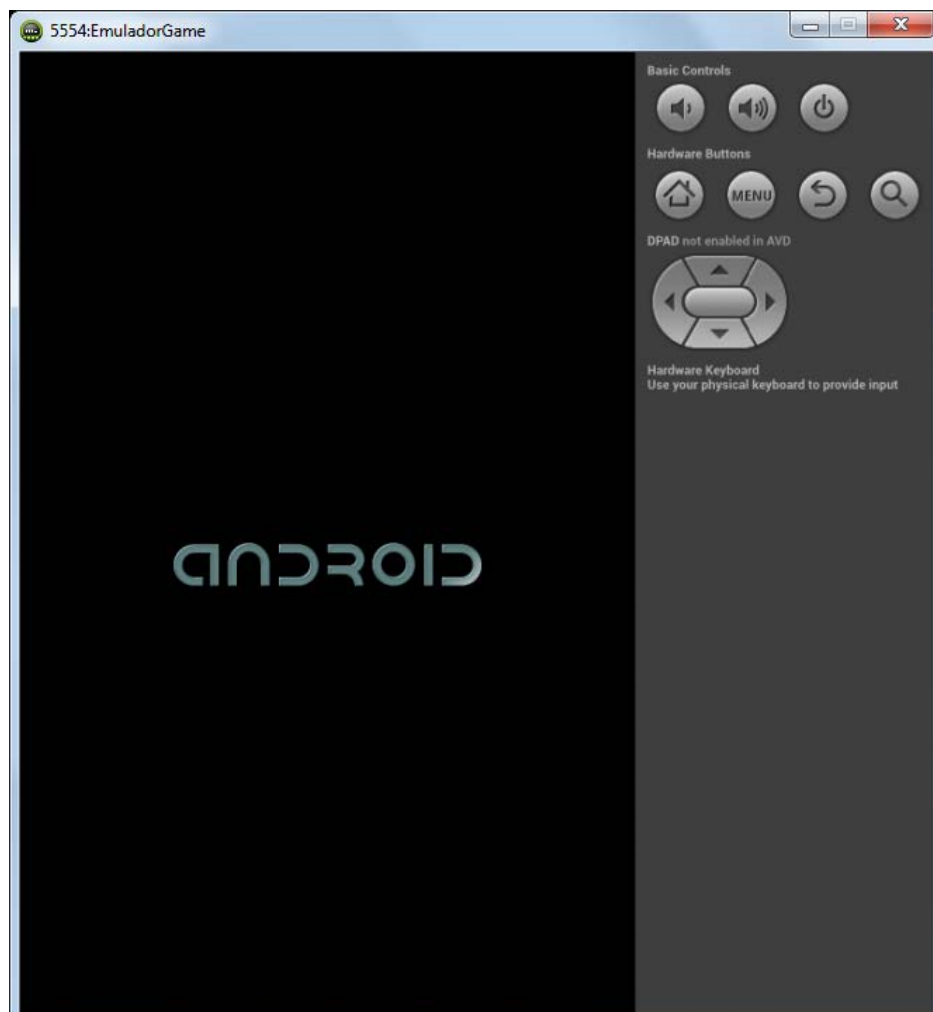
Feito isso o projeto será “compilado” e em instantes será aberto um emulador, onde a nossa aplicação (jogo) “compilada” será executada. Veja na figura abaixo:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

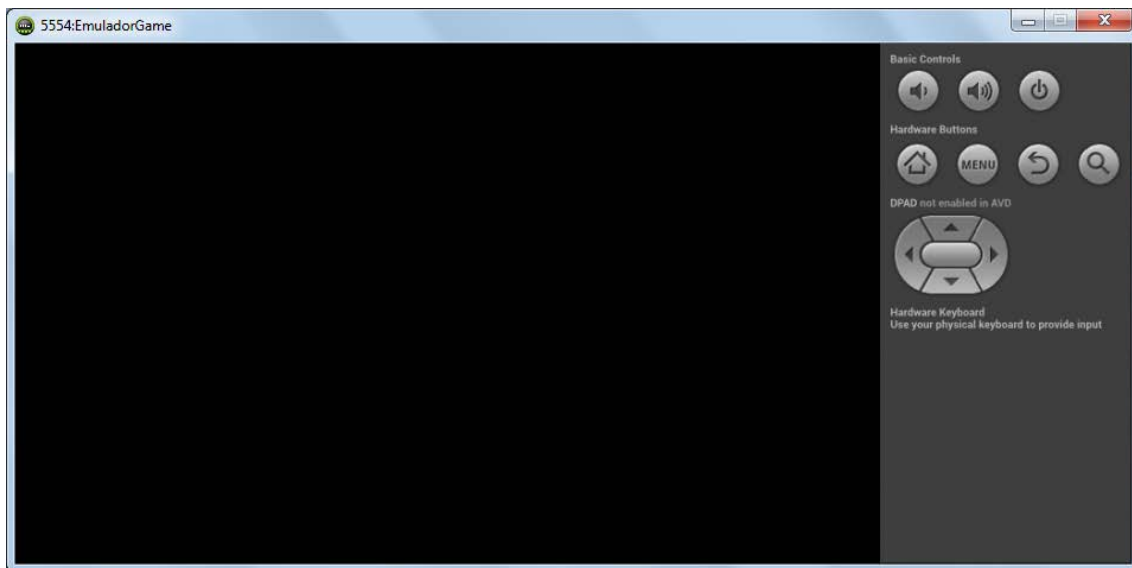


**Emulador Android em execução**

No início da execução do emulador mostra o título Android (de cor cinza), indicando o carregamento dos recursos do emulador. Esse processo normalmente demora em torno de 2 a 10 minutos (dependendo da sua máquina). É recomendável que você tenha no mínimo 512 MB de memória e um processador bem rápido para um bom desempenho da execução) para a aplicação ser exibida, mesmo sendo essa aplicação algo muito simples.

Se você observar, o emulador está rodando em modo “retrato”. Para alterarmos a orientação do emulador para o modo paisagem (que vamos usar daqui para frente), basta pressionar as teclas CTRL+F12.

Após o emulador ser carregado, você verá uma tela totalmente “escura”, sem nada na tela, conforme você pode ver na figura seguinte:



### Emulador do Android em Execução

A tela escura indica que não há nada na tela, que nenhum elemento está presente no jogo, por isso este resultado ocorre.

Se você neste exato momento fechou o emulador após a execução da aplicação (jogo), vou te dizer uma coisa: “Não era para você ter feito isso”. Se você esperou muito tempo para ver essa aplicação em execução, ao executar novamente a aplicação, possivelmente você vai esperar o mesmo tempo. Ao executar pela primeira vez o emulador, e caso vá executar outros programas, minimize o emulador ao invés de fechar, pois se você esperou muito tempo para executar esse programa, com ele minimizado, ao executar outro programa, o Eclipse vai fazer uso do emulador já aberto em vez de abrir outro, com isso a aplicação levará em torno de 7 a 12 segundos em média para ser executada. Nunca esqueça isso!

Neste capítulo conhecemos a respeito do nosso modelo de projeto de jogo para o Android e como executar as futuras aplicações. Nos próximos capítulos vamos começar a desenvolver nossos jogos, passo a passo, através deste material.



## Capítulo 5 Visualizando imagens no jogo (A classe Image)

No capítulo anterior conhecemos e aprendemos a manusear e a executar uma aplicação (jogo) através do modelo de projeto para jogos no Android. Neste capítulo iremos aprender a inserir e visualizar imagens em um jogo no Android, através da classe **Image**, que faz parte do pacote que o modelo de projeto oferece.

Conforme havia falado, todos os nossos projetos que iremos desenvolver, serão baseados no modelo de projeto presente neste material (o arquivo “Game Android Project V2\_2.zip”). Para isso vamos “importar” esse arquivo, conforme já foi mostrado no capítulo anterior.

Agora vamos alterar algumas propriedades do projeto, conforme podemos conferir abaixo:

Nome do projeto : VisualizandolImagensJogo

Constante “app\_name” : Visualizando Imagens Jogo

Constante “title\_game\_activity” : Visualizando Imagens Jogo

Depois de alterada as informações acima, vamos abrir o arquivo “GameMain.java” (que é o arquivo no qual vamos escrever o código do nosso jogo), que está situado no pacote “game.android.project” (dentro da pasta “src”).

Para visualizarmos uma imagem no jogo, precisamos fazer uso da classe **Image**, responsável por essa finalidade. Para isso, dentro da classe **GameMain** vamos declarar um atributo do tipo **Image**.

Na classe **GameMain**, na seção de declaração de atributos, vamos declarar um atributo chamado *imagem*, conforme podemos conferir em seguida:

```
Image imagem;
```



Agora dentro do construtor da classe (o método **GameMain**), vamos carregar o nosso atributo (objeto), com os seguintes parâmetros conforme você confere no código abaixo:

```
public GameMain(Context context) {  
    //Método construtor, disparado toda vez que o jogo é carregado na memória  
    super(context);  
    this.context = context;  
    surfaceHolder = getHolder();  
  
    //Insira o código aqui  
    imagem = new Image(context, R.drawable.game_android_icon, 0, 0, 48, 48);  
  
    setFocusable(true);  
}
```

Vamos analisar os parâmetros do comando inserido acima. O construtor da classe **Image** possui os seguintes parâmetros :

```
imagem = new Image(context, R.drawable.game_android_icon, 0, 0, 48, 48);
```

O primeiro parâmetro do construtor é padrão, cujo argumento sempre será o “context”. O segundo parâmetro corresponde a imagem que vamos visualizar no jogo (que é a imagem “game\_android\_icon.png”).

Antes de continuarmos com a explicação dos parâmetros, vamos entender essa “notação” que foi utilizada para expressar a imagem que será carregada. Observe que o argumento utilizado para especificar a imagem a ser carregada foi “*R.drawable.game\_android\_icon*”.

Conforme já foi explicado, todas as imagens que vamos utilizar em uma aplicação Android ficam no diretório “drawable-mdpi” (que fica dentro da pasta “res”). Na teoria, como a imagem (com diretório e tudo), é expressa dessa forma :

```
“res/drawable-mdpi/game_android_icon.png”
```

No Android, devemos utilizar a notação:

```
“R.drawable.game_android_icon”
```



A letra “R” da expressão acima, representa o diretório “res”, e a palavra “drawable” representa o diretório “drawable-mdpi”. Se observarmos acima, o nome da imagem não leva a “extensão”, pois o Android já identifica automaticamente a imagem e sua extensão.

Vejamos alguns exemplos da análise feita na tabela abaixo:

Caminho completo da imagem	Como expressar no Android
“res/drawable-mdpi/aviao.png”	R.drawable.aviao
“res/drawable-mdpi/cenario.jpg”	R.drawable.cenario
“res/drawable-mdpi/personagem.png”	R.drawable.personagem

Agora vamos continuar com o comentário dos parâmetros.

O terceiro e quarto parâmetro do construtor correspondem , respectivamente, as coordenadas X e Y do objeto na tela do jogo (cujos valores são “0” e “0”). Já o quinto e o sexto parâmetro correspondem, respectivamente, a largura e a altura do objeto (cujo valores são “48” e “48”).

Bom, o que fizemos até agora foi carregar o objeto (imagem) na memória. Agora precisamos desenhar a imagem carregada na tela. Para isso devemos utilizar o método **onDraw**, responsável por esse finalidade. Dentro do método **onDraw** vamos escrever o código abaixo:

```
protected void onDraw(Canvas canvas) {  
    super.onDraw(canvas);  
  
    Update();  
  
    canvas.drawColor(Color.BLACK);  
  
    imagem.Draw(canvas);  
}
```

Se observarmos o método acima, inserimos a seguinte linha de comando :

```
imagem.Draw(canvas);
```



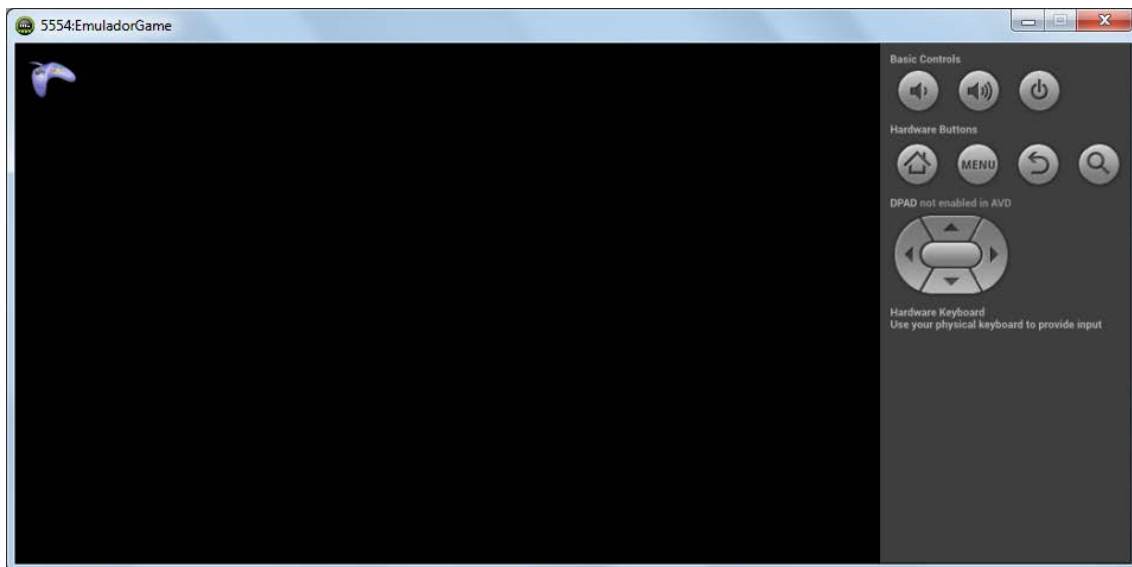
# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

Que é responsável por desenhar a imagem na tela, através do método **Draw**. Nesse método passamos como parâmetro o “canvas”, que representa a “tela” onde a imagem será desenhada.

Vamos conferir na figura abaixo como ficará o resultado na tela do emulador.



**Imagem em exibição na tela**

E ai, aprendeu como visualiza uma imagem na tela do jogo ? com certeza que sim!

Vamos aproveitar para colocar mais imagens na tela do jogo. Para esse exemplo irei colocar mais uma imagem, chamada “logotipo\_android.png” (que já acompanha este material).

Conforme já havia falado, todas as imagens que vamos utilizar em um jogo fica situado dentro do diretório “drawable-mdpi”, logo, a imagem que vamos usar para ser visualizada deve estar dentro desse diretório. Irei mostrar aqui duas formas de se colocar uma imagem dentro desse diretório, sendo que começarei pela forma mais “tradicional”, e depois pela forma mais fácil.

### **Forma tradicional**

Para importar um arquivo, clique com o botão direito do mouse sobre a pasta “res/drawable-mdpi” e selecione “Import”, depois selecione “File System” (Que se encontra dentro da pasta “General”, conforme mostra a figura abaixo) e em seguida clique em “Next”.

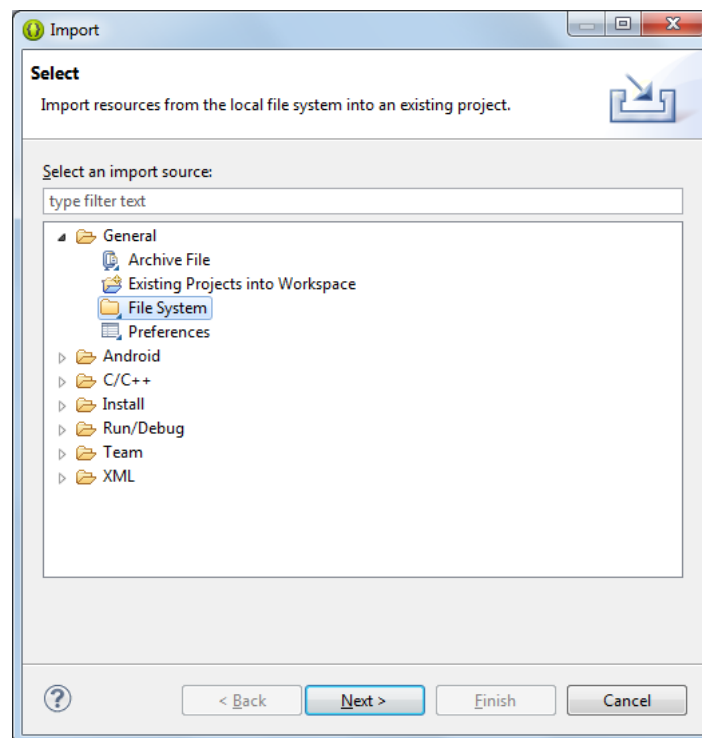




# Apostila de Android

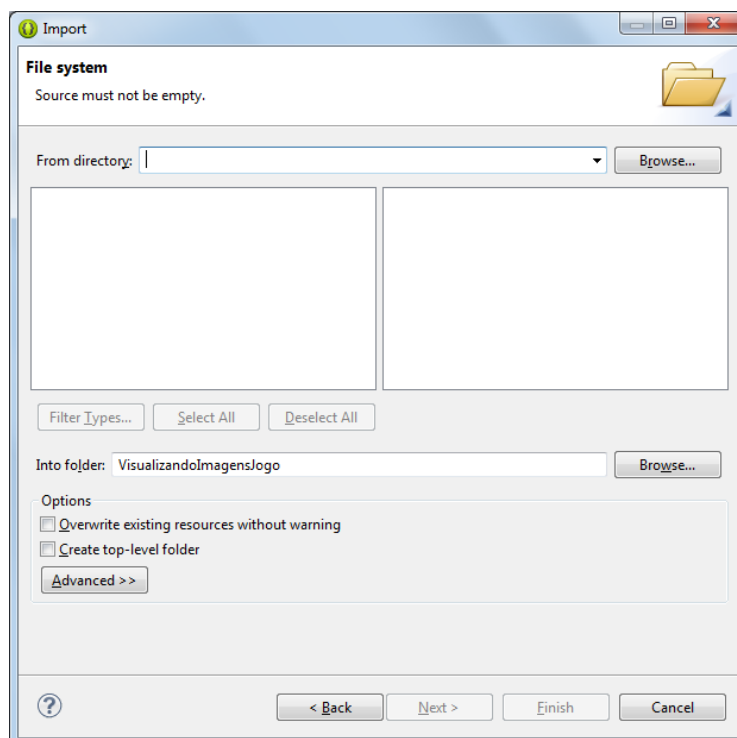
## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



**Selecionando a opção “File System”**

Após clicar em “Next” será exibida a caixa de diálogo como demonstra a figura abaixo:



**Caixa de diálogo – “Import”**

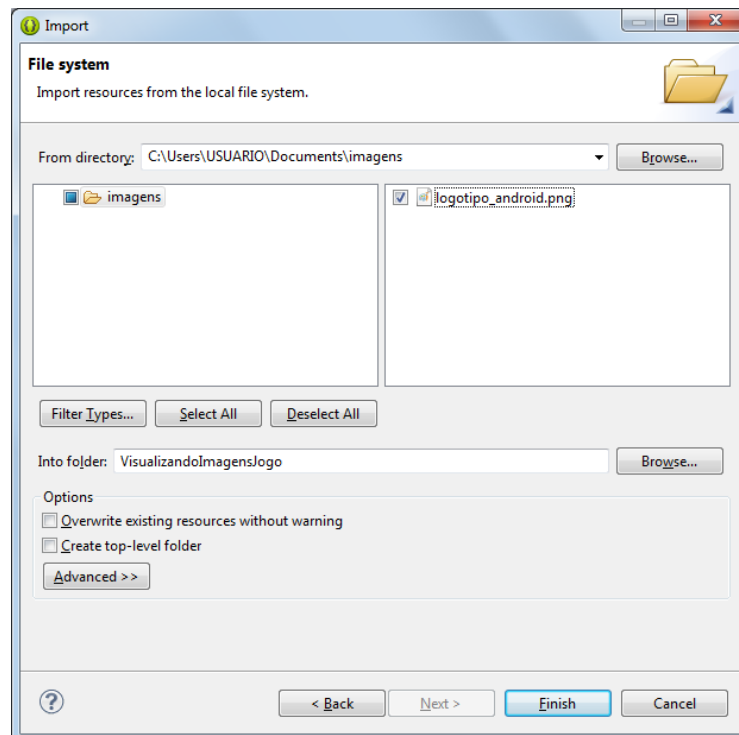


# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

Clique no botão “Browse...” para selecionar o diretório onde se encontram a imagem. Feito isso, marque o arquivo (imagem) para que ele seja importado para a pasta “res/drawable-mdpi” . Veja a figura abaixo:



**Imagem selecionada**

Depois disso, é só clicar em “Finish” para importar as imagens para o projeto.

### **Forma mais fácil**

A segunda forma, que diria que é a mais fácil de todas, é você ir até o diretório onde se encontra a imagem , para que você em seguida possa selecioná-la, e logo após copiá-la (o famoso “Ctrl+C”). Feito isso vá até o projeto que criamos para selecionarmos o diretório “drawable-mdpi” para colarmos as imagens dentro da pasta, simplesmente dando CTRL+V (simples não ?).

Com a imagem já inserida no projeto, vamos alterar as seguintes linhas de comandos nos métodos que citarei abaixo. Primeiramente vamos declarar mais um atributo chamado *logotipo*, na seção de declaração de atributos (onde declaramos o atributo *imagem*), conforme você confere em seguida:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
:  
Image imagem;  
Image logotipo;
```

Agora dentro do método construtor **GameMain** vamos efetuar as mudanças, conforme é mostrado no código abaixo:

```
public GameMain(Context context) {  
  
    super(context);  
    this.context = context;  
    surfaceHolder = getHolder();  
  
    imagem = new Image(context, R.drawable.game_android_icon, 450, 230, 48,  
48);  
    logotipo = new Image(context, R.drawable.logotipo_android, 370, 200, 55,  
72);  
  
    setFocusable(true);  
}
```

Se observarmos o código acima, adicionamos mais uma imagem para ser carregada (o atributo *logotipo*), com os seus respectivos parâmetros. Na imagem anteriormente adicionada, alteramos o valor dos argumentos relacionados a coordenadas X e Y, mudando a posição de sua exibição na tela.

Agora dentro do método **onDraw** vamos digitar o seguinte código, conforme é mostrado em seguida:

```
protected void onDraw(Canvas canvas) {  
  
    super.onDraw(canvas);  
  
    Update();  
  
    canvas.drawColor(Color.BLACK);  
  
    imagem.Draw(canvas);  
    logotipo.Draw(canvas);  
  
}
```

Se observarmos o código do método **onDraw** acima, a única coisa que adicionamos foi a seguinte linha de código:

```
logotipo.Draw(canvas);
```

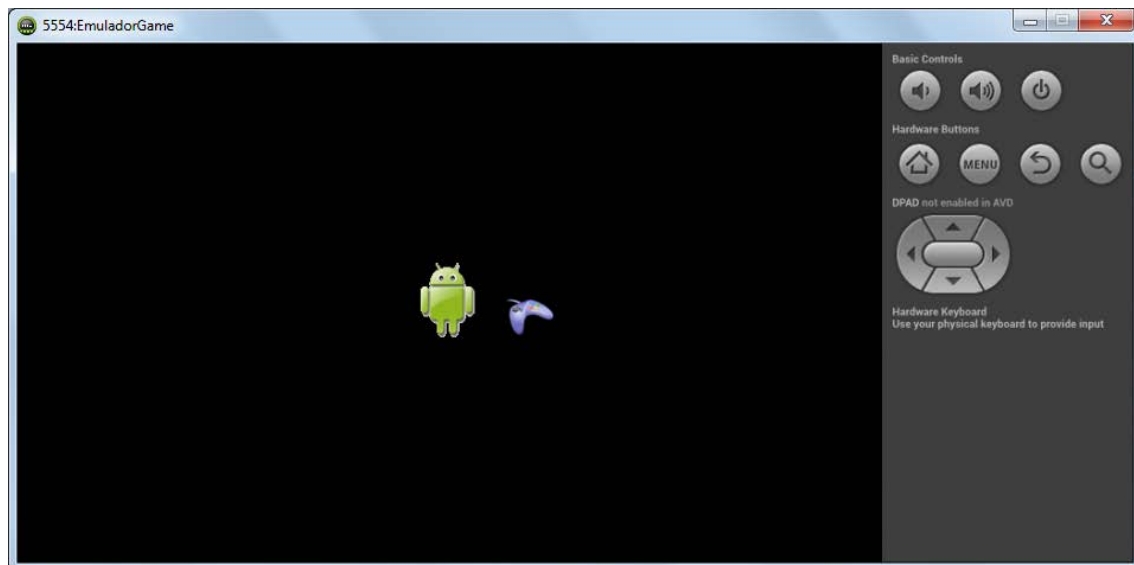
Que exibe na tela o “logo” do Android. Vejamos na imagem seguinte o resultado:



# Apostila de Android

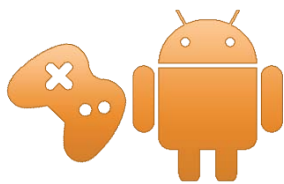
## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



**Imagem em exibição na tela**

Até aqui aprendemos como colocar uma imagem para ser exibida na tela de um jogo. No próximo capítulo iremos aprender como movimentar essa imagem (e também outros tipos de elementos) pela tela do jogo.



## Capítulo 6 Movendo elementos do jogo pela tela

No capítulo anterior aprendemos como visualizar elementos (como imagens) na tela, porém, as imagens vistas ficam num ponto “fixo” da tela, definido pelo código. Agora vamos aprender a movimentar esses elementos pela tela do jogo, usando os recursos oferecidos pelo Android.

Neste tópico vamos aprender a movimentar os elementos do jogo usando o teclado do dispositivo (que no emulador vai ser “emulado” pelo teclado do computador) e o “touch screen”. Iremos começar aprendendo a movimentar os elementos através do teclado e no próximo exemplo iremos aprender a movimentá-los via “touch screen”.

Antes de começarmos, vamos importar o modelo de projeto para o ADT, pois será a partir dele que vamos construir nossos jogos, como fizemos anteriormente.

Depois de importar o projeto, vamos alterar algumas propriedades do projeto, conforme podemos conferir abaixo:

Nome do projeto : MovendoObjetoViaTeclado

Constante “app\_name” : Movendo Objetos via Teclado

Constante “title\_game\_activity” : Movendo Objetos via Teclado

Depois de criado o projeto, vamos adicionar as seguintes imagens (que se encontram presentes neste material) , que se chamam “aviao.png” , “nuvem.png” e “ceu\_azul.png”.

Depois de adicionar as imagens no projeto (dentro do diretório “drawable-mdpi”), vamos abrir o arquivo “GameMain.java” para escrevermos o código do nosso jogo.



Para começar, vamos declarar os seguintes atributos na seção de declaração de atributos da classe **GameMain** :

```
Image aviao, nuvem, ceu_azul;
```

Agora dentro do método construtor **GameMain**, vamos adicionar os seguintes comandos, conforme você confere em seguida:

```
public GameMain(Context context) {  
  
    super(context);  
    this.context = context;  
    surfaceHolder = getHolder();  
  
    aviao = new Image(context, R.drawable.aviao, 0, 0, 90, 70);  
    nuvem = new Image(context, R.drawable.nuvem, 250, 100, 90, 70);  
  
    setFocusable(true);  
}
```

Nas linhas de comando que adicionamos acima (no método **GameMain**), carregamos as imagens do avião e da nuvem.

Agora vamos no método **onSizeChanged**, para adicionarmos o seguinte código:

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {  
  
    super.onSizeChanged(w, h, oldw, oldh);  
  
    ceu_azul = new Image(context, R.drawable.ceu_azul, 0, 0, w, h);  
}
```

Na linha que adicionarmos no método acima, carregamos a imagem do céu azul. Agora a pergunta que eu faço é : Por quê que carregamos a imagem do céu azul nesse método ? A resposta é simples, simplesmente porque precisamos que a imagem do céu azul se ajuste ao tamanho da tela, logo, a largura e a altura da tela deverá ser as mesmas do tamanho da tela.

Neste método temos os seguintes argumentos, *w* que armazena a largura da tela e *h* que armazena a altura da tela. Esses valores foram passados como argumentos no construtor da imagem do céu azul, como podemos conferir em seguida:

```
ceu_azul = new Image(context, R.drawable.ceu_azul, 0, 0, w, h);
```



Agora vamos no método **onDraw** para adicionarmos as seguintes linhas de comando:

```
protected void onDraw(Canvas canvas) {  
  
    super.onDraw(canvas);  
    Update();  
    canvas.drawColor(Color.BLACK);  
  
    ceu_azul.Draw(canvas);  
    nuvem.Draw(canvas);  
    aviao.Draw(canvas);  
  
}
```

Nas linhas de comando que adicionarmos acima, todas elas, desenharam as imagens na tela do jogo. Agora, as linhas de comando acima foram adicionadas em uma sequência, que precisamos entender e prestar atenção. Irei explicar aqui.

A primeira linha de comando que adicionarmos, desenha a imagem do céu azul na tela, até aí tudo bem. Na linha seguinte desenhamos a nuvem na tela, essa nuvem será desenhada “na frente” da imagem do céu azul. Por último, desenhamos a imagem do avião, e esse avião será desenhado na frente da imagem da nuvem e do céu azul.

O que podemos entender da explicação acima? Que a última imagem que será desenhada na tela sempre será desenhada na frente das anteriormente adicionadas.

Como havia falado, iremos movimentar os objetos via teclado (neste caso aqui, o avião). Vamos voltar na seção de declaração de atributos, e lá vamos declarar as seguintes instruções, como segue:

```
enum Sentido { PRA_FRENTE, PRA_TRAS, PARADO }  
Sentido sentidoAviao = Sentido.PARADO;
```

Para trabalharmos com o teclado do dispositivo (que será “emulado” pelo teclado do PC), vamos fazer uso dos métodos **onKeyDown** e **onKeyUp**. Primeiramente vamos no método **onKeyDown** para digitarmos o seguinte código:





```
public boolean onKeyDown(int keyCode, KeyEvent event) {  
  
    if(keyCode == KeyEvent.KEYCODE_DPAD_RIGHT)  
        sentidoAviao = Sentido.PRA_FRENTE;  
    else if(keyCode == KeyEvent.KEYCODE_DPAD_LEFT)  
        sentidoAviao = Sentido.PRA_TRAS;  
  
    return super.onKeyDown(keyCode, event);  
}
```

Irei comentar agora instruções que foram adicionadas no método acima. A expressão:

```
if(keyCode == KeyEvent.KEYCODE_DPAD_RIGHT)
```

Verifica se o usuário pressionou a tecla seta para direita, caso verdadeiro, atualiza o valor do atributo *sentidoAviao* com um valor que indica que o objeto irá se mover para frente (no caso, a constante “Sentido.PRA\_FRENTE”), como você confere no código em seguida:

```
sentidoAviao = Sentido.PRA_FRENTE;
```

A explicação é similar para a próxima avaliação condicional (que verifica se foi pressionada a seta para esquerda).

Agora vamos no método **onKeyUp** para adicionar o seguinte código:

```
public boolean onKeyUp(int keyCode, KeyEvent event) {  
  
    if(keyCode == KeyEvent.KEYCODE_DPAD_RIGHT)  
        sentidoAviao = Sentido.PARADO;  
    else if(keyCode == KeyEvent.KEYCODE_DPAD_LEFT)  
        sentidoAviao = Sentido.PARADO;  
  
    return super.onKeyUp(keyCode, event);  
}
```

Na avaliação condicional colocada no método acima, se a tecla seta para direita ou esquerda for solta (liberada), o atributo *sentidoAviao* irá assumir um valor que indica que o objeto irá ficar parado (no caso, a constante “Sentido.PARADO”)

Nos eventos (métodos) de teclado utilizados acima, o que ocorre é somente a atualização do valor do atributo *sentidoAviao*. A atualização do atributo não fará (ainda) que o objeto se movimento para frente ou para trás. Precisamos processar esse valor em um método que irá realmente “movimentar” o objeto



na tela. O movimento do objeto será processado no método **Update**. Vamos no método **Update** para digitarmos o seguinte código:

```
public void Update() {  
    if(sentidoAviao == Sentido.PRA_FRENTE)  
        aviao.MoveByX(10);  
    else if(sentidoAviao == Sentido.PRA_TRAS)  
        aviao.MoveByX(-10);  
}
```

Irei comentar as instruções do método acima. A expressão :

```
if(sentidoAviao == Sentido.PRA_FRENTE)
```

Verifica se o atributo *sentidoAviao* assumi o valor “Sentido.PRA\_FRENTE”, que indica que o objeto será movimentado para frente. Se a condição acima for verdadeira, o avião será movimento para frente, através do método **MoveByX**, conforme você pode conferir abaixo:

```
aviao.MoveByX(10);
```

O comando acima vai deslocar o avião para frente (na verdade, para direita), de 10 em 10 pixels, conforme você pode conferir no valor do argumento do método acima, que indica o tamanho do deslocamento em pixels.

Na condição seguinte, é avaliado se o atributo *sentidoAviao* assumi o valor “Sentido.PRA\_TRAS”, que indica que o objeto será movimentado para trás. Se a condição acima for verdadeira, o avião será movimento para trás, através do método **MoveByX**, conforme você pode conferir abaixo:

```
aviao.MoveByX(-10);
```

O comando acima vai deslocar o avião para trás (na verdade, para esquerda), de 10 em 10 pixels (observe que , conforme você pode conferir no argumento, o valor está “negativo”, indicando que ele fará o movimento no sentido contrário).

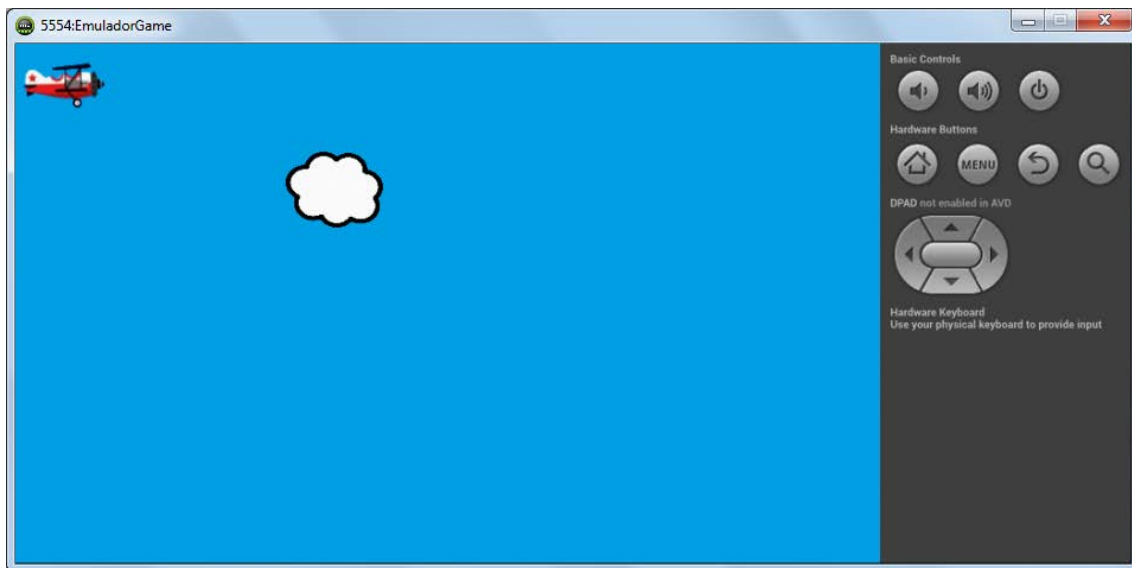
Vamos executar a nossa aplicação. O resultado você confere na imagem seguinte:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



### Movendo o avião pela tela

Se observarmos o jogo, nós conseguimos mover o avião para frente e para trás, pressionando as teclas atribuídas a ele, porém, a nuvem que adicionamos no jogo está imóvel. O ideal seria que essa nuvem se movimentasse, dando a sensação de que o avião está voando (em movimento). Para colocarmos a nuvem em movimento, precisaremos adicionar mais algumas instruções, para começar vamos declarar um atributo, conforme você pode conferir em seguida:

```
int largura_tela;
```

Agora dentro do método **onSizeChanged**, logo após o comando que adicionamos, vamos adicionar a seguinte instrução:

```
largura_tela = w;
```

Que retorna para a variável *largura\_tela* o valor da largura da tela do dispositivo (armazenado no argumento *w* do método).

Agora dentro do método **Update** vamos adicionar o seguinte comando, como segue:

```
nuvem.MoveByX(-15);  
  
if(nuvem.GetX() < -nuvem.GetWidth())  
    nuvem.SetX(largura_tela);
```



Irei explicar cada linha de comando das instruções adicionadas acima. A linha:

```
nuvem.MoveByX(-15);
```

Desloca a nuvem para a esquerda, de 15 em 15 pixels (lembre-se : valor positivo significa deslocamento pra direita, e valor negativo deslocamento para esquerda). A próxima instrução :

```
if(nuvem.GetX() < -nuvem.GetWidth())
```

Verifica se a nuvem saiu for a da tela. Como interpretamos a avaliação feita pela condição acima ? Irei explicar para você. O método **GetX** (do objeto *nuvem*, do tipo **Image**) retorna a posição da coordenada X do objeto na tela, e o método **GetWidth** retorna a largura do objeto. Quando a posição X de qualquer objeto for menor que o valor negativo de sua largura, significa que o objeto não está visível na tela, ou seja, está fora dela (fora pelo lado esquerdo da tela). Quando isso acontece, é executada a seguinte instrução:

```
nuvem.SetX(largura_tela);
```

Que diz que o objeto será posicionado, na coordenada X, no final da tela do dispositivo pela direita. O método **SetX** define ,de forma absoluta a posição do objeto, na coordenada X. Como valor da coordenada X foi definido o mesmo valor da largura da tela (armazenado na variável *largura\_tela*).

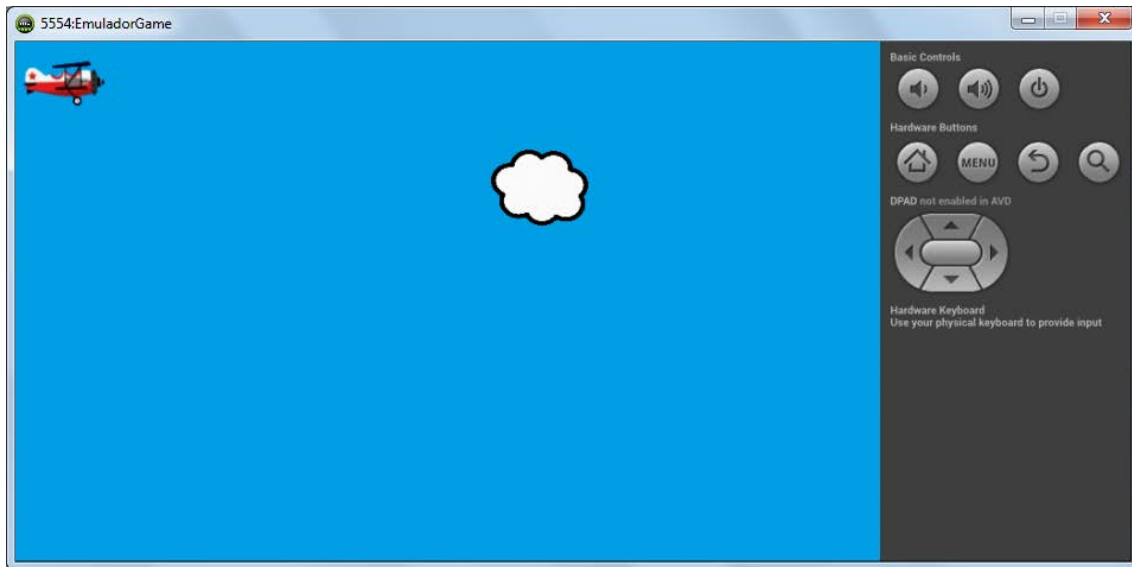
Vamos executar o nosso jogo para conferirmos o resultado, conforme podemos conferir na imagem seguinte:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



**Nuvem em movimento na tela**

Até aqui aprendemos a movimentar os objetos utilizando o teclado certo (e até também de forma automática) ? A forma que mostrei acima não será muito utilizada, visto que , a maioria dos aparelhos (como os mais modernos Smartphones e Tablets Android) suportam somente “touch screen” (é raro ver algum aparelho que ainda tenha teclado).

Agora iremos desenvolver o mesmo exemplo, só que ao invés do teclado vamos usar o famoso “touch screen” (agora a coisa vai ficar divertida!).

Vamos importar o nosso modelo de projeto para o ADT e em seguida vamos alterar as seguintes propriedades, como segue:

Nome do projeto : MovendoObjetoViaTouchScreen

Constante “app\_name” : Movendo Objetos via Touch Screen

Constante “title\_game\_activity” : Movendo Objetos via Touch Screen

Depois de criado o projeto, vamos adicionar as seguintes imagens (que se encontram presentes neste material) , que se chamam “aviao.png” , “nuvem.png” , “ceu\_azul.png”, “seta\_direita.png” e “seta\_esquerda.png”.



Depois de adicionar as imagens no projeto (dentro do diretório “drawable-mdpi”), vamos abrir o arquivo “GameMain.java” para escrevermos o código do nosso jogo.

Para começar, vamos declarar os seguintes atributos na seção de declaração de atributos da classe **GameMain** :

```
Image aviao, nuvem, ceu_azul, seta_direita, seta_esquerda;

enum Sentido { PRA_FRENTE, PRA_TRAS, PARADO }
Sentido sentidoAviao = Sentido.PARADO;

int largura_tela;
```

Agora dentro do método construtor **GameMain**, vamos adicionar os seguintes comandos, conforme você confere em seguida:

```
public GameMain(Context context) {

    super(context);
    this.context = context;
    surfaceHolder = getHolder();

    aviao = new Image(context, R.drawable.aviao, 0, 0, 90, 70);
    nuvem = new Image(context, R.drawable.nuvem, 250, 100, 90, 70);

    setFocusable(true);
}
```

Agora no método **onSizeChanged**, vamos adicionar os seguintes comandos como segue:

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {

    super.onSizeChanged(w, h, oldw, oldh);

    ceu_azul = new Image(context, R.drawable.ceu_azul, 0, 0, w, h);
    largura_tela = w;

    seta_esquerda = new Image(context, R.drawable.seta_esquerda, 0, h - 96,
    96, 96);
    seta_direita = new Image(context, R.drawable.seta_direita, w - 96, h - 96,
    96, 96);
}
```



No método acima , como podemos ver, carregamos as imagens da seta para direita e da seta para esquerda. As duas imagens são ajustadas de acordo com a resolução da tela para que elas se encontrem, respectivamente, no canto inferior esquerdo (para seta esquerda) e no canto inferior direito (para seta direita).

Agora no método **Update** vamos adicionar o seguinte código :

```
public void Update() {  
  
    if(sentidoAviao == Sentido.PRA_FRENTE)  
        aviao.MoveByX(10);  
    else if(sentidoAviao == Sentido.PRA_TRAS)  
        aviao.MoveByX(-10);  
  
    nuvem.MoveByX(-15);  
  
    if(nuvem.GetX() < -nuvem.GetWidth())  
        nuvem.SetX(largura_tela);  
  
}
```

No método **onDraw** vamos adicionar o seguinte código:

```
protected void onDraw(Canvas canvas) {  
  
    super.onDraw(canvas);  
  
    Update();  
  
    canvas.drawColor(Color.BLACK);  
  
    ceu_azul.Draw(canvas);  
    nuvem.Draw(canvas);  
    aviao.Draw(canvas);  
    seta_esquerda.Draw(canvas);  
    seta_direita.Draw(canvas);  
  
}
```

Se você observar no código acima, existem duas imagens, que representa setas de direção (seta para direita e seta para esquerda). O movimento do avião será feito pressionando as setas de direção. Onde será feita a verificação





do pressionamento das setas ? No método **onTouchEvent**. Vamos agora no método **onTouchEvent** para adicionarmos o seguinte código abaixo:

```
public boolean onTouchEvent(MotionEvent event) {

    if(event.getAction() == MotionEvent.ACTION_DOWN) {

        if(seta_esquerda.IsTouch(event.getX(), event.getY()))
        {
            sentidoAviao = Sentido.PRA_TRAS;
        }
        else if(seta_direita.IsTouch(event.getX(), event.getY()))
        {
            sentidoAviao = Sentido.PRA_FRENTE;
        }
    }
    else if(event.getAction() == MotionEvent.ACTION_UP)
    {
        sentidoAviao = Sentido.PARADO;
    }

    return true;

}
```

Agora vamos a explicação do código do método. A primeira linha de comando:

```
if(event.getAction() == MotionEvent.ACTION_DOWN)
```

Verifica se ocorreu um pressionamento (toque) na tela do dispositivo. Se a situação acima for verdadeira, é efetuada a seguinte avaliação:

```
if(seta_esquerda.IsTouch(event.getX(), event.getY()))
```

Verifica se a imagem foi “tocada” na tela, através do método **IsTouch** . Nesse método são passados dois argumentos, que correspondem, respectivamente, as coordenadas X e Y de onde ocorreu o toque na tela. Se a condição acima for verdadeira, a variável de controle *sentidoAviao* recebe o status “Sentido.PRA\_TRAS”, que indica o avião vai andar para trás. A avaliação condicional seguinte é similar ao que foi explicado agora.

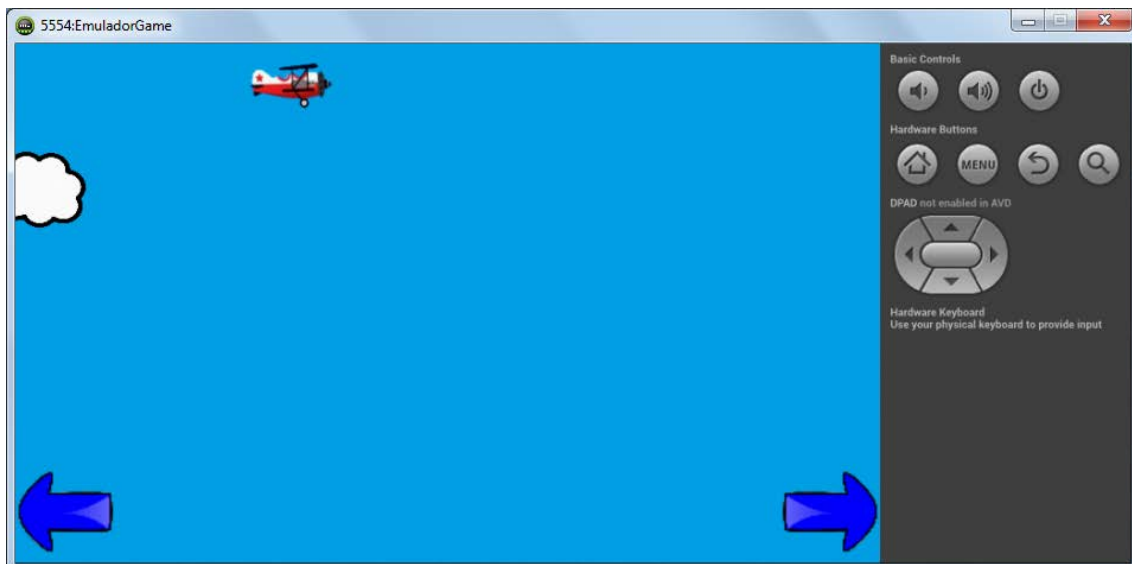


Uma outra avaliação que acontece nesse código é essa:

```
if(event.getAction() == MotionEvent.ACTION_UP)
```

Que verifica se ocorreu uma liberação de toque na tela (ou seja, que verifica se o dedo foi retirado da tela). Caso a avaliação acima seja verdadeira, a variável *sentidoAviao* recebe o status "Sentido.PARADO", que indica o avião vai ficar parado.

Vamos executar a nossa aplicação ? O resultado você confere na figura seguinte:



**Movendo o avião via touch screen**

Quando movimentamos o avião para frente e para trás, existe a possibilidade do mesmo sair totalmente da tela (indo em qualquer uma das direções). Vamos desenvolver agora um código que faça com que o avião não saia da tela. Para isso, vamos no método **Update** para digitarmos o seguinte código, após o último código inserido:

```
if(aviao.GetX() < 0)
{
    nuvem.SetX(0);
    sentidoAviao = Sentido.PARADO;
}
else if(aviao.GetX() > (largura_tela - aviao.GetWidth()))
{
    nuvem.SetX(largura_tela - aviao.GetWidth());
    sentidoAviao = Sentido.PARADO;
}
```



Irei explicar agora o código inserido acima. A expressão:

```
if(aviao.GetX() < 0)
```

Verifica se o avião está saindo da tela pela esquerda (se a posição X for menor que 0, indica essa situação). Se a condição acima for verdadeira, a posição X do avião é definida para 0 (através do método **SetX**) e o avião para de se movimentar. Na próxima avaliação:

```
if(aviao.GetX() > (largura_tela - aviao.GetWidth()))
```

É verificado se o avião saiu da tela pela direita (caso a posição X dele seja maior do que a (largura da tela - largura do avião)). Se a condição acima for verdadeira, é feito o bloqueio do avião, impedindo que o mesmo saia da tela, usando as mesmas técnicas apresentadas na explicação anterior.

No exemplo que fizemos até agora, o nosso avião se movimenta somente para frente e para trás, mas, não seria ideal se nosso avião também se movimentasse para cima e para baixo? Vamos adicionar esse recurso em nosso jogo. Em nosso projeto vamos adicionar mais duas imagens chamadas “seta\_cima.png” e “seta\_baixo.png”.

Vamos adicionar mais dois atributos (do tipo **Image**) chamado *seta\_cima* e *seta\_baixo*, na seção de declaração de atributos, conforme podemos conferir abaixo:

```
Image aviao, nuvem, ceu_azul, seta_direita, seta_esquerda, seta_cima,  
seta_baixo;
```

Agora no código, na estrutura de enumeração abaixo:

```
enum Sentido { PRA_FRENTE, PRA_TRAS, PARADO }
```



Vamos adicionar mais duas constantes : PRA\_CIMA e PRA\_BAIXO. Confira como ficou :

```
enum Sentido { PRA_FRENTE, PRA_TRAS, PRA_CIMA, PRA_BAIXO, PARADO }
```

Também na seção de declaração de variáveis, vamos declarar um atributo chamado *altura\_tela*, conforme você confere em seguida:

```
int altura_tela;
```

No método **onSizeChanged** vamos adicionar as seguintes linhas de comando:

```
seta_cima = new Image(context, R.drawable.seta_cima, 96, h - 96, 96, 96);  
seta_baixo = new Image(context, R.drawable.seta_baixo, w - (96 * 2), h - 96,  
96, 96);  
altura_tela = h;
```

Agora no método **onDraw** vamos adicionar o seguinte código:

```
seta_cima.Draw(canvas);  
seta_baixo.Draw(canvas);
```

No método **onTouchEvent** vamos adicionar o seguinte código:

:

```
if(event.getAction() == MotionEvent.ACTION_DOWN) {  
    if(seta_esquerda.IsTouch(event.getX(), event.getY()))  
    {  
        sentidoAviao = Sentido.PRA_TRAS;  
    }  
    else if(seta_direita.IsTouch(event.getX(), event.getY()))  
    {  
        sentidoAviao = Sentido.PRA_FRENTE;  
    }  
    else if(seta_cima.IsTouch(event.getX(), event.getY()))  
    {  
        sentidoAviao = Sentido.PRA_CIMA;  
    }  
}
```



```
        else if(seta_baixo.IsTouch(event.getX(), event.getY()))
        {
            sentidoAviao = Sentido.PRA_BAIXO;
        }
    }
    :
```

Agora , para finalizar, no método **Update** vamos adicionar o seguinte código, que será responsável por movimentar o avião também para cima e para baixo:

```
    :
    if(sentidoAviao == Sentido.PRA_FRENTE)
        aviao.MoveByX(10);
    else if(sentidoAviao == Sentido.PRA_TRAS)
        aviao.MoveByX(-10);
    else if(sentidoAviao == Sentido.PRA_CIMA)
        aviao.MoveByY(-10);
    else if(sentidoAviao == Sentido.PRA_BAIXO)
        aviao.MoveByY(10);

    nuvem.MoveByX(-15);
    if(nuvem.GetX() < -nuvem.GetWidth())
        nuvem.SetX(largura_tela);

    if(aviao.GetX() < 0)
    {
        nuvem.SetX(0);
        sentidoAviao = Sentido.PARADO;
    }
    else if(aviao.GetX() > (largura_tela - aviao.GetWidth()))
    {
        nuvem.SetX(largura_tela - aviao.GetWidth());
        sentidoAviao = Sentido.PARADO;
    }
    if(aviao.GetY() < 0)
    {
        nuvem.SetY(0);
        sentidoAviao = Sentido.PARADO;
    }
    else if(aviao.GetY() > (altura_tela - aviao.GetHeight()))
    {
        nuvem.SetX(largura_tela - aviao.GetHeight());
        sentidoAviao = Sentido.PARADO;
    }
    :
    :
```

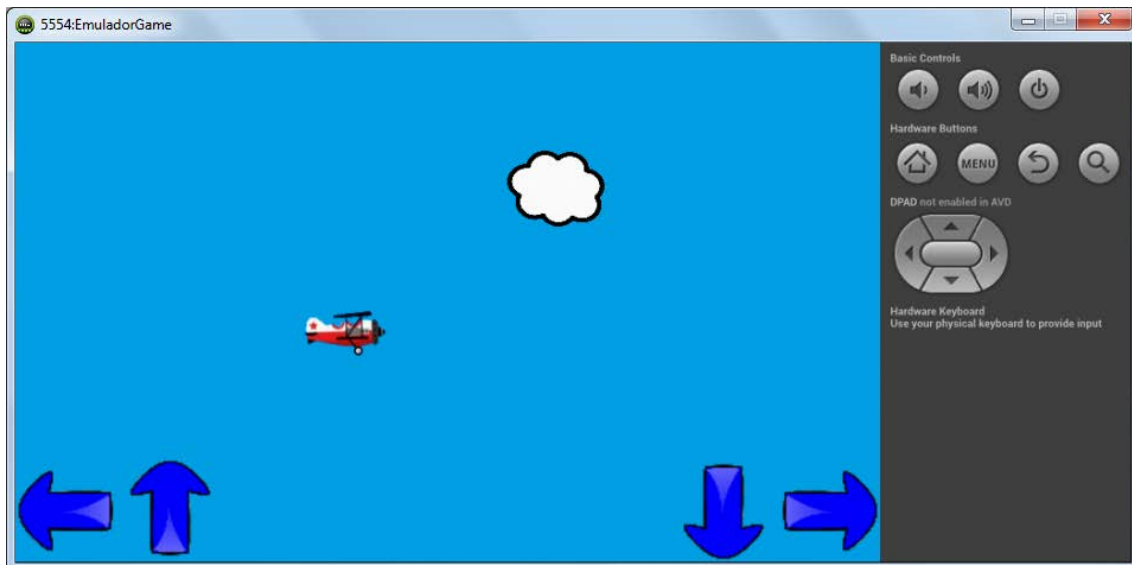
Experimente executar agora o jogo para conferir os resultados, conforme você pode ver na figura seguinte:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



**Avião se movimentando em todas as direções**

Lembre-se: Para movimentar o avião acima, você deve movê-lo para uma determinada direção, por vez.



## **Capítulo 7** Trabalhando com animação de sprites (A classe AnimationSprites)

**N**os capítulos anteriores, aprendemos a visualizar imagens “estáticas” no jogo (eu pode ser qualquer elemento do jogo como uma árvore, um poste, um bloco e etc). Agora, se eu quisesse que algum elemento do jogo apresenta-se alguma animação ? Neste capítulo vamos aprender como realizar essas animações usando sprites no jogo.

Na maioria dos jogos em 2D (talvez podemos dizer todos), alguns os elementos apresentam algum tipo de animação (como o movimento de um personagem ou a ação de algum objeto). Agora a pergunta que faço é : como são realizadas essas animações no jogo ? Essas animações são realizadas utilizando o que nós chamamos de sprites. O que vem a ser uma sprite (não é refrigerante rs) ? Sprites nada mais são do que um conjunto de imagens, onde cada imagem representa o movimento ou ação de um determinado elemento do jogo (como um personagem ou um objeto do jogo qualquer).

Vamos ver abaixo um conjunto de imagens (sprites) de um personagem, onde cada imagem representa um movimento:



**Sprites de um personagem de um jogo (dando soco)**

Como funciona a animação de um personagem (ou objeto) durante o jogo ? Se observarmos as imagens acima, cada uma delas representa um movimento, e cada uma delas é exibida numa fatia de “tempo”, dando a sensação de movimento ou ação.





Para trabalharmos com animação de sprites aqui no Android, nós vamos fazer uso da classe chamada **AnimationSprites**, destinada somente para esse tipo de tarefa.

Para começarmos, vamos importar nosso modelo de projeto e em seguida, vamos realizar algumas modificações nele, conforme é mostrado em seguida:

Nome do projeto : AnimacaoSprites

Constante “app\_name” : Animacao de Sprites

Constante “title\_game\_activity” : Animacao de Sprites

Depois de criado o projeto, vamos adicionar as seguintes imagens (sprites) no projeto : “naruto\_parado\_1.png”, “naruto\_parado\_2.png”, “naruto\_parado\_3.png” e “naruto\_parado\_4.png”.

Depois de colocar as imagens dentro do projeto, vamos abrir o arquivo “GameMain.java”, para digitarmos o código do nosso jogo. Na declaração de atributos, vamos realizar a seguinte declaração, conforme é mostrado em seguida:

```
AnimationSprites naruto;
```

Agora dentro do método construtor **GameMain** , vamos escrever o seguinte código, conforme é mostrado abaixo:

```
public GameMain(Context context) {  
  
    super(context);  
    this.context = context;  
    surfaceHolder = getHolder();  
  
    naruto = new AnimationSprites(context, 0, 0, 124, 164);  
    naruto.Add(R.drawable.naruto_parado_1);  
    naruto.Add(R.drawable.naruto_parado_2);  
    naruto.Add(R.drawable.naruto_parado_3);  
    naruto.Add(R.drawable.naruto_parado_4);  
    naruto.Start(3, true);  
  
    setFocusable(true);  
}
```



Irei explicar o código inserido no método acima. A linha de comando:

```
naruto = new AnimationSprites(context, 0, 0, 124, 164);
```

Carrega a instância do objeto *naruto* do tipo **AnimationSprites**. No primeiro argumento da classe, sempre passamos o valor “context”. No segundo e terceiro parâmetro, passamos respectivamente a coordenada X e Y do objeto na tela. Já os dois últimos parâmetros representam , respectivamente , a largura e a altura do objeto (124 para largura e 164 para altura). Nas linhas de comando seguinte:

```
:  
  
naruto.Add(R.drawable.naruto_parado_1);  
naruto.Add(R.drawable.naruto_parado_2);  
naruto.Add(R.drawable.naruto_parado_3);  
naruto.Add(R.drawable.naruto_parado_4);  
  
:
```

Adicionamos as imagens (sprites) dentro do objeto, para que possamos ter uma animação do personagem, através do método **Add**. Em seguida, vem a seguinte instrução:

```
naruto.Start(3, true);
```

Que tem a finalidade de “iniciar” a animação, através do método **Start**. Esse método possui dois parâmetros : O primeiro parâmetro é responsável por definir, em frames, o intervalo da troca de imagens, e o segundo definimos se a animação vai ficar “em loop” (caso **true**) ou não (caso **false**).

Agora no método **onDraw** vamos adicionar a seguinte instrução:

```
protected void onDraw(Canvas canvas) {  
  
    super.onDraw(canvas);  
  
    Update();  
  
    canvas.drawColor(Color.BLACK);  
    naruto.Draw(canvas);  
  
}
```

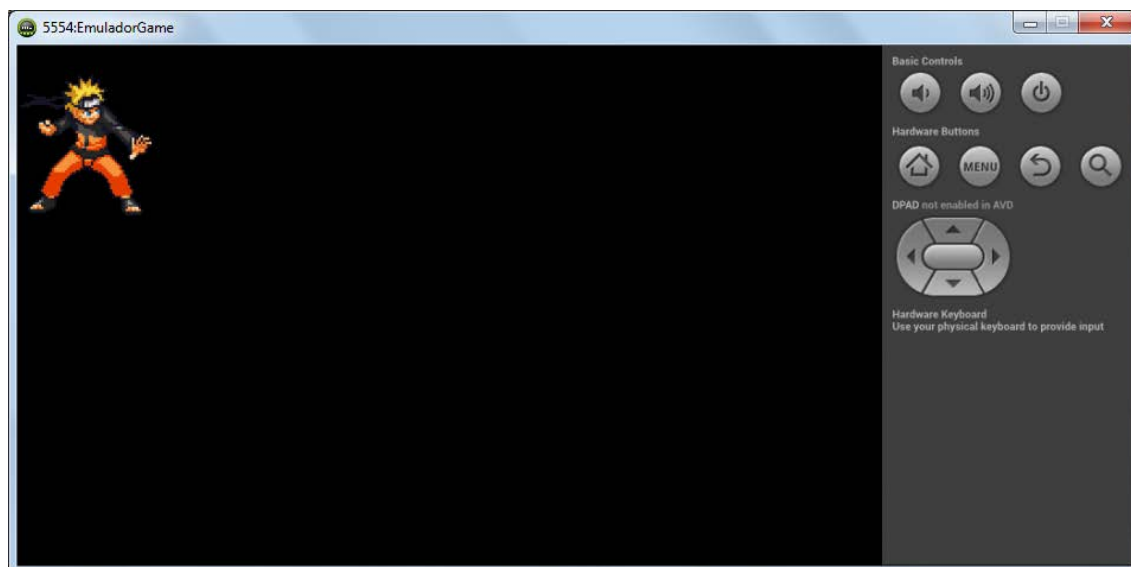


# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

Vamos executar agora a nossa demonstração ? Veja como ficou na figura seguinte:



**Animação do personagem em execução**

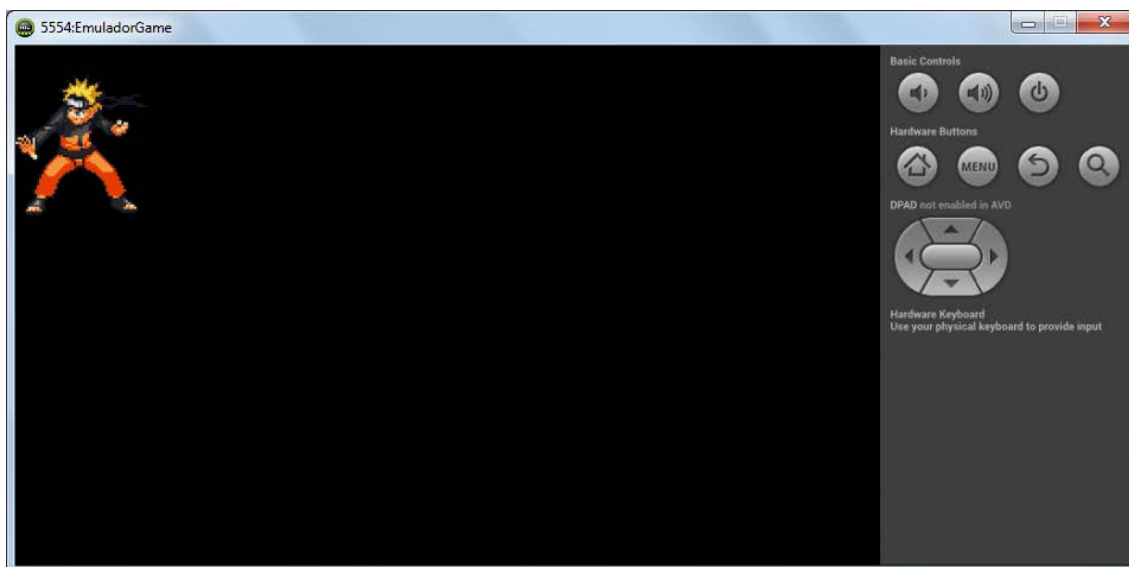
Em uma animação de sprites (e até mesmo em uma imagem estática), podemos aplicar o efeito e “flip” (inversão) de uma imagem, na horizontal. Para invertermos uma imagem na horizontal, basta adicionar um parâmetro no método **Draw** do objeto. Voltemos para o método **onDraw** para substituímos a linha abaixo:

```
naruto.Draw(canvas);
```

Por essa:

```
naruto.Draw(canvas,FlipEffect.HORIZONTAL);
```

Que inverte as imagens da animação (como havia falado, esse procedimento acima funciona também para imagens estáticas). Veja o resultado na figura seguinte:



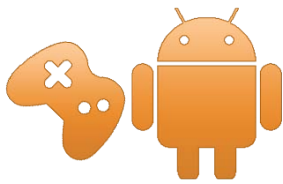
**Animação do personagem invertida na horizontal**

### **Regras para a criação de sprites**

Quando criamos sprites (animações) de um determinado elemento de um jogo, normalmente, cada imagem que irá representar uma ação desse elemento, deve ter o mesmo tamanho (padronizado para cada elemento do jogo).

Todos os movimentos do personagem acima (naruto) , estão cada um deles em uma imagem, e todas essas imagens possui a mesma largura e altura.

Até agora aprendemos a visualizar uma imagem estática no jogo, dar movimento aos elementos do jogo e aprendemos agora neste capítulo, como visualizar e realizar uma animação de sprites, por meio da classe **AnimationSprites**.



## **Capítulo 8 Detectando colisões entre objetos no jogo**

Uma das técnicas mais importantes que não se pode faltar em um jogo, é a detecção de colisões entre elementos. Neste capítulo, iremos aprender a detectar se dois objetos (ou mais) se colidem um com o outro no jogo, através dos recursos oferecidos neste material.

Para a detecção de colisão entre elementos, iremos fazer uso da classe **Collision**, que já se encontra presente no modelo de projeto que usamos.

Para começar, vamos importar nosso modelo de projeto para o ADT, e em seguida vamos fazer as seguintes modificações:

Nome do projeto : ColisaoEntreObjetos

Constante “app\_name” : Colisao entre Objetos

Constante “title\_game\_activity” : Colisao entre Objetos

Depois de criado o projeto , vamos adicionar as seguintes imagens (dentro do diretório “drawable-mdpi”) : “barra\_horizontal.png”, “barra\_vertical.png”, “personagem\_1.png” e “personagem\_2.png”.

Agora vamos abrir o arquivo “GameMain.java” para escrevermos o código do nosso jogo. Dentro da seção de declaração de atributos, vamos realizar as declarações abaixo, como segue:

```
AnimationSprites personagem;  
Image barra_esquerda, barra_direita, barra_cima, barra_baixo;  
Image seta_direita, seta_esquerda, seta_cima, seta_baixo;  
  
enum Sentido { PRA_ESQUERDA, PRA_DIREITA, PRA_CIMA, PRA_BAIXO, PARADO }  
Sentido sentidoPersonagem = Sentido.PARADO;
```

Agora dentro do método **onSizeChanged** (isso mesmo, não é no método **GameMain** não), vamos adicionar as seguintes instruções:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {

    super.onSizeChanged(w, h, oldw, oldh);

    personagem = new AnimationSprites(context, (w / 2) - 36, 200, 72, 70);
    personagem.Add(R.drawable.personagem_1);
    personagem.Add(R.drawable.personagem_2);
    personagem.Start(3, true);

    barra_cima = new Image(context, R.drawable.barra_horizontal, (w / 2) - 200
    , 10, 400, 60);
    barra_baixo = new Image(context, R.drawable.barra_horizontal, (w / 2) - 200
    , 300, 400, 60);

    barra_esquerda = new Image(context, R.drawable.barra_vertical,
    barra_cima.GetX() - 60 , barra_cima.GetY() - 1, 60, 354);

    barra_direita = new Image(context, R.drawable.barra_vertical,
    barra_cima.GetX() + barra_cima.GetWidth() - 1 , barra_cima.GetY(), 60, 354);

    seta_esquerda = new Image(context, R.drawable.seta_esquerda, 0, h - 96, 96,
    96);
    seta_direita = new Image(context, R.drawable.seta_direita, w - 96, h - 96,
    96, 96);

    seta_cima = new Image(context, R.drawable.seta_cima, 96, h - 96, 96, 96);
    seta_baixo = new Image(context, R.drawable.seta_baixo, w - (96 * 2), h -
    96, 96, 96);

}
```

Agora dentro do método **onDraw** vamos adicionar o seguinte código:

```
protected void onDraw(Canvas canvas) {

    super.onDraw(canvas);

    Update();

    canvas.drawColor(Color.BLACK);
    barra_baixo.Draw(canvas);
    barra_cima.Draw(canvas);
    barra_esquerda.Draw(canvas);
    barra_direita.Draw(canvas);
    personagem.Draw(canvas);

    seta_esquerda.Draw(canvas);
    seta_direita.Draw(canvas);
    seta_cima.Draw(canvas);
    seta_baixo.Draw(canvas);

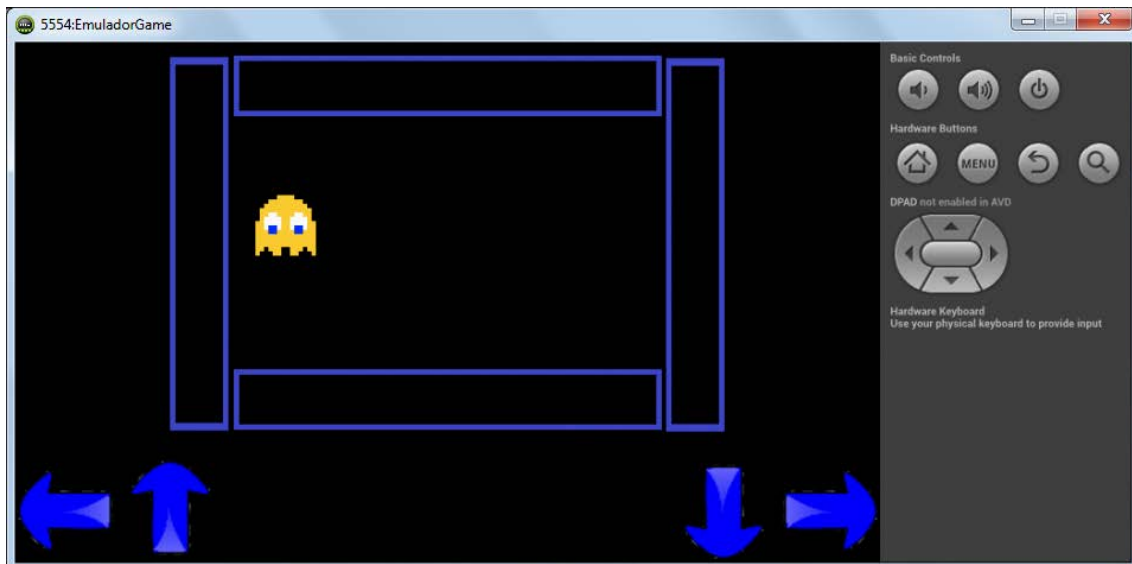
}
```



E para finalizar, no método **Update**, vamos adicionar o seguinte código:

```
public void Update() {  
  
    if(sentidoPersonagem == Sentido.PRA_DIREITA)  
        personagem.MoveByX(10);  
    else if(sentidoPersonagem == Sentido.PRA_ESQUERDA)  
        personagem.MoveByX(-10);  
    else if(sentidoPersonagem == Sentido.PRA_CIMA)  
        personagem.MoveByY(-10);  
    else if(sentidoPersonagem == Sentido.PRA_BAIXO)  
        personagem.MoveByY(10);  
  
}
```

Agora vamos ver como vai ficar a execução do nosso jogo, conforme podemos conferir na imagem seguinte:



**Jogo em execução**

A idéia desse jogo é fazer com que o nosso personagem fique “preso” dentro dessa arena. Se movimentarmos o nosso personagem, podemos observar que ele se movimenta “livremente” pela tela, sem obstáculos. Nós vamos fazer com que o nosso personagem se “colida” com as barras horizontais e verticais, fazendo com que o mesmo não saia da arena na qual ele se encontra.

Para adicionarmos o código de colisão, vamos no método **Update** para adicionarmos as seguinte instruções, conforme você confere em seguida:





# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
public void Update() {  
  
    if(sentidoPersonagem == Sentido.PRA_DIREITA)  
    {  
        personagem.MoveByX(10);  
  
        if(Collision.Check(personagem, barra_direita)){  
            personagem.MoveByX(-10);  
            sentidoPersonagem = Sentido.PARADO;  
        }  
    }  
    else if(sentidoPersonagem == Sentido.PRA_ESQUERDA)  
    {  
        personagem.MoveByX(-10);  
  
        if(Collision.Check(personagem, barra_esquerda)){  
            personagem.MoveByX(10);  
            sentidoPersonagem = Sentido.PARADO;  
        }  
    }  
    else if(sentidoPersonagem == Sentido.PRA_CIMA)  
    {  
        personagem.MoveByY(-10);  
  
        if(Collision.Check(personagem, barra_cima)){  
            personagem.MoveByY(10);  
            sentidoPersonagem = Sentido.PARADO;  
        }  
    }  
    else if(sentidoPersonagem == Sentido.PRA_BAIXO)  
    {  
        personagem.MoveByY(10);  
  
        if(Collision.Check(personagem, barra_baixo)){  
            personagem.MoveByY(-10);  
            sentidoPersonagem = Sentido.PARADO;  
        }  
    }  
}
```

Irei explicar agora o código que avalia a colisão do personagem com as barras horizontais e verticais. A expressão abaixo:

```
if(sentidoPersonagem == Sentido.PRA_DIREITA)
```



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

Verifica se a variável *sentidoPersonagem* está assumindo o status “Sentido.PRA\_DIREITA” (indicando que o personagem está andando para direita). Se a condição acima for verdadeira, o personagem irá se deslocar para direita, 10 pixels, através da instrução abaixo:

```
personagem.MoveByX(10);
```

Em algum momento do deslocamento, poderá haver um determinado momento em que o personagem poderá se colidir com a barra vertical direita, agora, como avaliar se houve a colisão ? Para verificar se houve um colisão entre dois objetos usamos o método **Check** da classe **Collision**, como podemos ver na instrução abaixo:

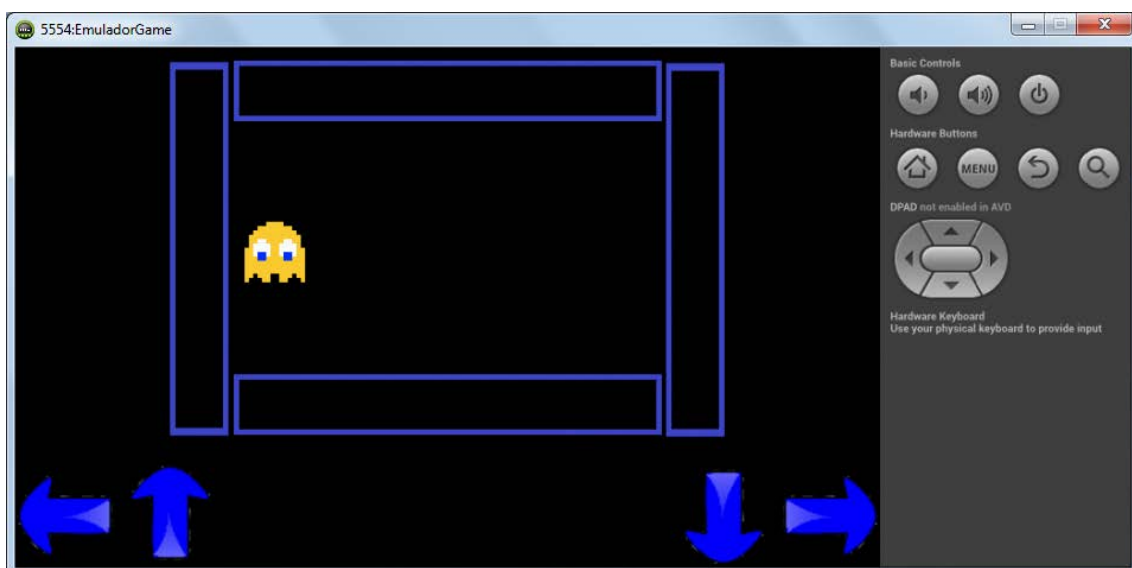
```
if(Collision.Check(personagem, barra_direita))
```

Se ocorrer uma colisão, nosso personagem irá voltar um passo anterior (se ele andou 10 pixels para direita e nesse movimento ele colidiu com a barra vertical, ele voltará um passo anterior), conforme você confere em seguida:

```
personagem.MoveByX(-10);
```

A instrução seguinte atualiza o status da variável *sentidoPersonagem* para “Sentido.PARADO”, indicando que o personagem vai estar parado.

A explicação das condições seguintes é “similar” ao da explicação feita agora. Vamos agora executar o nosso jogo, o resultado você confere em seguida:



**Personagem em colisão com a barra vertical durante movimento**



Como pudemos observar acima, a colisão é uma das técnicas de desenvolvimento de jogos mais importantes, com essa técnica conseguimos muitas coisas (como verificar se o personagem está pisando em algum bloco, se um tiro atingiu o inimigo e etc.).

Vamos criar aqui mais um exemplo de colisão entre objetos. Vamos importar aqui mais um modelo de projeto, e em seguida vamos realizar as seguintes modificações:

Nome do projeto : ColisaoEntreObjetos2

Constante “app\_name” : Colisao entre Objetos 2

Constante “title\_game\_activity” : Colisao entre Objetos 2

A idéia desse exemplo consiste em um personagem em que ele dispara um tiro para atingir um morcego. Quando o morcego é atingido , o mesmo voa diretamente para cima, saindo da tela.

Agora dentro do nosso projeto vamos adicionar as seguintes imagens : “cenario.png”, “heroi.png”, “morcego1\_voando\_1.png”, “morcego1\_voando\_2.png” e “tiro.png”.

Agora dentro do arquivo “GameMain.java” vamos escrever os seguintes códigos, que serão apresentados a seguir.

Dentro da classe **GameMain** vamos realizar as seguintes declarações de atributos, conforme é mostrado a seguir:

```
Image heroi, cenario;  
AnimationSprites morcego;  
Image tiro;  
  
boolean acertou;
```

Agora dentro do método **onSizeChanged** , vamos adicionar o seguinte código abaixo:

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {  
    super.onSizeChanged(w, h, oldw, oldh);  
  
    cenario = new Image(context, R.drawable.cenario, 0, 0, w, h);  
    heroi = new Image(context, R.drawable.heroi, 150, 305, 100, 130);  
    morcego = new AnimationSprites(context, 500, 350, 91, 60);  
    morcego.Add(R.drawable.morcego1_voando_1);  
    morcego.Add(R.drawable.morcego1_voando_2);  
}
```



# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

```
    morcego.Start(3, true);

    tiro = new Image(context, R.drawable.tiro, heroi.GetX() +
        heroi.GetWidth(), heroi.GetY() + 40, 16, 8);

    acertou = false;
}
```

Dentro do método **onDraw** vamos escrever o seguinte código abaixo:

```
protected void onDraw(Canvas canvas) {

    super.onDraw(canvas);

    Update();

    canvas.drawColor(Color.BLACK);

    cenário.Draw(canvas);
    heroi.Draw(canvas);
    morcego.Draw(canvas);
    tiro.Draw(canvas);

}
```

Agora dentro do método **Update** vamos adicionar o seguinte código abaixo:

```
public void Update() {

    tiro.MoveByX(5);

    if(Collision.Check(tiro, morcego))
    {
        if(!acertou)
        {
            acertou = true;
        }
    }

    if(acertou)
        morcego.MoveByY(-10);

}
```



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

Vamos avaliar o código da colisão acima. A instrução :

```
tiro.MoveByX(5);
```

Move o tiro para a direita, de 5 em 5 pixels. Na linha:

```
if(Collision.Check(tiro, morcego))
```

Verifica se houve a colisão do tiro com o morcego. Se a condição acima for verdadeira, a variável *acertou* assume o valor **true** (verdadeiro).

Na próxima condição:

```
if(acertou)
```

Avalia se a variável *acertou* assumi o valor **true**. Se a condição acima for verdadeiro, o morcego é deslocado para acima, através da instrução abaixo:

```
morcego.MoveByY(-10);
```

Vamos executar agora o nosso jogo ? O resultado você poderá conferir na figura seguinte:



**Jogo em execução**



# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

A técnica de colisão, como havia falado, é uma das técnicas mais importantes utilizadas no desenvolvimento de jogos, com ela podemos detectar se o personagem está pisando em alguma coisa, se o tiro atingiu algum elemento do jogo (como acabamos de realizar acima), se um personagem colidiu com um muro e etc.

No próximo capítulo, iremos aprender a criar elementos dinâmicos, durante a execução do jogo.





## Capítulo 9 Criando elementos dinâmicos durante a execução do jogo

Uma das atividades que normalmente ocorrem durante a execução de um jogo é a criação dinâmica de vários elementos (como por exemplo, vários inimigos vindo em sua direção). Neste capítulo iremos aprender a criar elementos dinâmicos durante a execução do jogo.

Antes de começarmos vamos importar o modelo de projeto para o ADT, e em seguida vamos fazer as seguintes modificações conforme é mostrado em seguida:

Nome do projeto : CriandoElementosDinamicos

Constante “app\_name” : Criando Elementos Dinamicos

Constante “title\_game\_activity” : Criando Elementos Dinamicos

O exemplo que vamos desenvolver, demonstrando a técnica de criação dinâmica de elementos, vai consistir num pequeno jogo onde vamos controlar uma nave, que de tempo em tempo, vão surgindo as naves inimigas na tela. Nossa nave terá que disparar contra essas naves inimigas, que vem em sua direção.

Vamos colocar agora em nosso projeto as seguintes imagens : “seta\_direita.png”, “seta\_esquerda.png”, “seta\_cima.png”, “seta\_baixo.png”, “botão\_tiro.png”, “cena1.jpg”, “cena2.jpg”, “tiro.png”, “nave.png”, “inimigo.png”.

Agora vamos abrir o arquivo “GameMain.java” para escrevermos o código do nosso jogo. Se observarmos o código do arquivo, existe uma seção na qual se encontram declaradas todas as bibliotecas que vamos utilizar em nosso jogo, conforme podemos conferir em seguida:





# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
GameMain.java X
package game.android.project;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import game.util.*;

public class GameMain extends SurfaceView implements Runnable {

    GameState gameState;

    Thread thread = null;
    SurfaceHolder surfaceHolder;
    volatile boolean running = false;
    Context context;
```

#### Seção de declaração de bibliotecas (seção “import”)

Na seção acima, vamos precisar “importar” uma biblioteca que precisaremos utilizar para a construção do nosso jogo, e essa biblioteca (ou pacote, como normalmente chamamos) que vamos usar é a “java.util”. Na seção de bibliotecas, logo após a linha :

```
import game.util.*;
```

Vamos declarar a seguinte biblioteca:

```
import java.util.*;
```

Agora dentro da seção de declaração de atributos da classe **GameMain** vamos adicionar as seguintes declarações, como segue:

```
ArrayList<Image> aInimigos;

ArrayList<Image> aTiros;

Image cena1, cena2, nave;

Image seta_direita, seta_esquerda,
seta_cima, seta_baixo, botao_tiro;

enum Sentido { PRA_FRENTE, PRA_TRAS, PRA_CIMA, PRA_BAIXO, PARADO }
Sentido sentidoNave = Sentido.PARADO;

int contaFrames = 0;
boolean deuTiro = false;
```



Irei explicar agora as duas primeiras declarações acima. As instruções:

```
ArrayList<Image> aInimigos;  
  
ArrayList<Image> aTiros;
```

Declara, cada uma, um “array” (ou vetor) de imagens. O tipo **ArrayList** é uma estrutura em que nela podemos adicionar (ou remover) vários elementos durante a execução do jogo de forma dinâmica. Pelo nome que atribuímos a cada atributo, podemos identificar que o primeiro só vai armazenar inimigos, e o outro só vai armazenar tiros (que serão efetuados pela nave que iremos controlar no jogo).

Vamos agora dentro do método construtor **GameMain** para adicionarmos as seguintes instruções:

```
public GameMain(Context context) {  
  
    super(context);  
    this.context = context;  
    surfaceHolder = getHolder();  
  
    nave = new Image(context, R.drawable.nave, 0, 100, 80, 51);  
    aTiros = new ArrayList<Image>();  
    aInimigos = new ArrayList<Image>();  
  
    setFocusable(true);  
}
```

Agora dentro do método **onSizeChanged** vamos adicionar o seguinte código abaixo:

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {  
  
    super.onSizeChanged(w, h, oldw, oldh);  
  
    cena1 = new Image(context, R.drawable.cena_1, 0, 0, w, h);  
    cena2 = new Image(context, R.drawable.cena_2, w, 0, w, h);  
  
    seta_esquerda = new Image(context, R.drawable.seta_esquerda, 0, h - 96,  
6, 96);  
    seta_direita = new Image(context, R.drawable.seta_direita, w - 96, h - 96,  
96, 96);  
    seta_cima = new Image(context, R.drawable.seta_cima, 96, h - 96, 96, 96);  
    seta_baixo = new Image(context, R.drawable.seta_baixo, w - (96 * 2), h -  
96, 96, 96);  
  
    botao_tiro = new Image(context, R.drawable.botao_tiro, 0, h - (96 * 2),  
96, 96);  
  
}
```



Irei explicar aqui um trecho de código , do método acima, para mostrar uma técnica aqui que vou utilizar neste exemplo que consiste no deslocamento de cenário “em loop”. Nesse jogo vou utilizar duas imagens, onde cada uma corresponde a um cenário (chamadas respectivamente de *cena1* e *cena2*), conforme você pode conferir nas instruções seguintes:

```
cena1 = new Image(context, R.drawable.cena_1, 0, 0, w, h);  
cena2 = new Image(context, R.drawable.cena_2, w, 0, w, h);
```

Observe que ambas as imagens possuem a mesma largura e altura da tela do dispositivo (se olharmos os parâmetros acima constatamos isso). A primeira cena começa na coordenada X com “0”, e a segunda cena começa na coordenada “X” com o mesmo valor da largura da primeira cena (e consequentemente, da mesma largura da tela do dispositivo). Essa técnica faz com que as duas imagens fiquem ao lado uma da outra. No código do método **Update** (que veremos logo mais), vamos visualizar o código que realiza o esquema de deslocamento “em loop”, que dá a sensação de “cenário infinito”.

Agora dentro do método **onDraw** vamos escrever as seguintes instruções:

```
protected void onDraw(Canvas canvas) {  
  
    super.onDraw(canvas);  
  
    Update();  
  
    canvas.drawColor(Color.BLACK);  
  
    cena1.Draw(canvas);  
    cena2.Draw(canvas);  
    nave.Draw(canvas);  
    seta_esquerda.Draw(canvas);  
    seta_baixo.Draw(canvas);  
    seta_cima.Draw(canvas);  
    seta_direita.Draw(canvas);  
    botao_tiro.Draw(canvas);  
  
}
```

E agora dentro do método **Update** vamos adicionar as seguintes instruções abaixo:

```
public void Update() {  
  
    if(sentidoNave == Sentido.PRA_FRENTE)  
        nave.MoveByX(10);  
  
}
```



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
else if(sentidoNave == Sentido.PRA_TRAS)
    nave.MoveByX(-10);
else if(sentidoNave == Sentido.PRA_CIMA)
    nave.MoveByY(-10);
else if(sentidoNave == Sentido.PRA_BAIXO)
    nave.MoveByY(10);

cena1.MoveByX(-10);
cena2.MoveByX(-10);

if(cena1.GetX() < -cena1.GetWidth())
    cena1.SetX(cena2.GetX() + cena2.GetWidth());

if(cena2.GetX() < -cena2.GetWidth())
    cena2.SetX(cena1.GetX() + cena1.GetWidth());

}
```

Agora dentro do método **onTouchEvent** vamos escrever o seguinte código abaixo:

```
public boolean onTouchEvent(MotionEvent event) {

    if(event.getAction() == MotionEvent.ACTION_DOWN) {

        if(seta_esquerda.IsTouch(event.getX(), event.getY()))
        {
            sentidoNave = Sentido.PRA_TRAS;
        }
        else if(seta_direita.IsTouch(event.getX(), event.getY()))
        {
            sentidoNave = Sentido.PRA_FRENTE;
        }
        else if(seta_cima.IsTouch(event.getX(), event.getY()))
        {
            sentidoNave = Sentido.PRA_CIMA;
        }
        else if(seta_baixo.IsTouch(event.getX(), event.getY()))
        {
            sentidoNave = Sentido.PRA_BAIXO;
        }

    }
    else if(event.getAction() == MotionEvent.ACTION_UP)
    {
        sentidoNave = Sentido.PARADO;
    }

    return true;
}
```

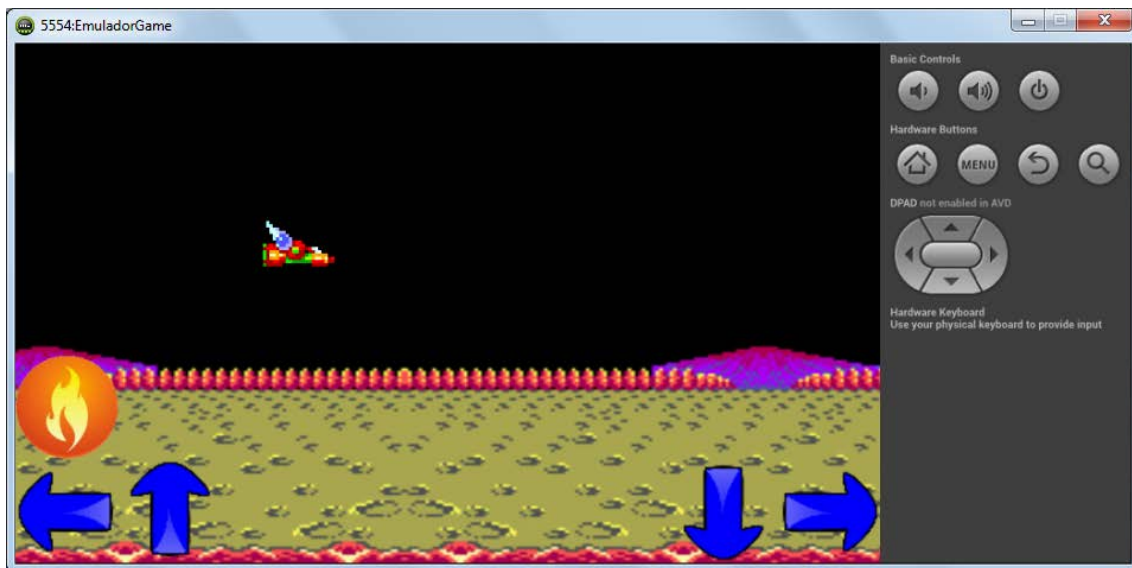
Vamos colocar o nosso jogo para executar ? O resultado você confere na figura seguinte:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



Jogo em execução

Até o presente momento no jogo, o que conseguimos fazer somente é mover o avião. Agora iremos implementar o código que vai efetuar os disparos da nave e a criação dos inimigos em tempo de execução.

Vamos agora no método **Update** para digitarmos o seguinte código (destacado em **negrito**) :

```
public void Update() {  
  
    if(sentidoNave == Sentido.PRA_FRENTE)  
        nave.MoveByX(10);  
    else if(sentidoNave == Sentido.PRA_TRAS)  
        nave.MoveByX(-10);  
    else if(sentidoNave == Sentido.PRA_CIMA)  
        nave.MoveByY(-10);  
    else if(sentidoNave == Sentido.PRA_BAIXO)  
        nave.MoveByY(10);  
  
    cena1.MoveByX(-10);  
    cena2.MoveByX(-10);  
  
    if(cena1.GetX() < -cena1.GetWidth())  
        cena1.SetX(cena2.GetX() + cena2.GetWidth());  
  
    if(cena2.GetX() < -cena2.GetWidth())  
        cena2.SetX(cena1.GetX() + cena1.GetWidth());  
  
    if(deuTiro)  
    {  
        aTiros.add(new Image(context, R.drawable.tiro, nave.GetX() +  
            nave.GetWidth(), nave.GetY() + 25, 24, 30));  
        deuTiro = false;  
    }  
}
```



```
contaFrames++;
if(contaFrames == 25)
{
    contaFrames = 0;

    int pos_y = (int) Math.round(Math.random() * 400);

    aInimigos.add(new Image(context, R.drawable.inimigo, 800, pos_y, 63,
        48));
}

for(int x = 0 ; x < aInimigos.size() ; x++)
{
    aInimigos.get(x).MoveByX(-10);
    if(aInimigos.get(x).GetX() < - aInimigos.get(x).GetWidth())
    {
        aInimigos.remove(x);
        x--;
    }
}

for(int x = 0 ; x < aTiros.size() ; x++)
{
    aTiros.get(x).MoveByX(10);
    for(int y = 0 ; y < aInimigos.size() ; y++)
    {
        if(Collision.Check(aTiros.get(x), aInimigos.get(y)))
        {
            aInimigos.remove(y);
            aTiros.remove(x);
            x--;
            break;
        }
    }
}
}
```

Irei comentar aqui os códigos em azul do método acima, que são responsáveis pelo movimento do tiro e dos inimigos, como também pela colisão entre os mesmos. O código:

```
if(deuTiro)
{
    aTiros.add(new Image(context, R.drawable.tiro, nave.GetX() +
        nave.GetWidth(), nave.GetY() + 25, 24, 30));
    deuTiro = false;
}
```



Verifica se a variável *deuTiro* assume o valor **true**. Se a condição acima for verdadeira, será adicionado o “array” *aTiros* uma imagem do tiro, que vai representar o tiro disparado pela nossa nave. O código seguinte:

```
contaFrames++;  
if(contaFrames == 25)  
{  
    contaFrames = 0;  
  
    int pos_y = (int) Math.round(Math.random() * 400);  
  
    aInimigos.add(new Image(context, R.drawable.inimigo, 800, pos_y, 63,  
48));  
}
```

Cria a cada 25 frames uma nave inimiga, posicionando essa nave sempre para surgir da direita. Irei explicar a seguinte linha de comando :

```
int pos_y = (int) Math.round(Math.random() * 400);
```

Cria um valor aleatório entre 0 e 400, que será o valor da coordenada Y onde a nave será criada (que vai estar armazenada na variável *pos\_y*).

O seguinte loop abaixo:

```
for(int x = 0 ; x < aInimigos.size() ; x++)  
{  
    aInimigos.get(x).MoveByX(-10);  
    if(aInimigos.get(x).GetX() < - aInimigos.get(x).GetWidth())  
    {  
        aInimigos.remove(x);  
        x--;  
    }  
}
```

Movimenta todos os inimigos pela tela. Se a nave inimiga sair pelo lado esquerdo da tela, ela é removida do “array”.

O código abaixo :

```
for(int x = 0 ; x < aTiros.size() ; x++)  
{  
    aTiros.get(x).MoveByX(10);  
    for(int y = 0 ; y < aInimigos.size() ; y++)  
    {  
        if(Collision.Check(aTiros.get(x), aInimigos.get(y)))  
        {  
            aInimigos.remove(y);  
            aTiros.remove(x);  
        }  
    }  
}
```





```
                x--;  
                break;  
            }  
        }  
    }
```

Movimenta os tiros disparados pela nossa nave, e avalia se o tiro colidiu com alguma nave inimiga. Se ocorrer alguma colisão do tiro com alguma nave inimiga, a nave e o tiro são removidos da tela.

E agora, dentro do método **onTouchEvent** vamos colocar o o seguinte código destacado **em negrito**:

```
public boolean onTouchEvent(MotionEvent event) {  
  
    if(event.getAction() == MotionEvent.ACTION_DOWN) {  
        if(seta_esquerda.IsTouch(event.getX(), event.getY()))  
        {  
            sentidoNave = Sentido.PRA_TRAS;  
        }  
        else if(seta_direita.IsTouch(event.getX(), event.getY()))  
        {  
            sentidoNave = Sentido.PRA_FRENTE;  
        }  
        else if(seta_cima.IsTouch(event.getX(), event.getY()))  
        {  
            sentidoNave = Sentido.PRA_CIMA;  
        }  
        else if(seta_baixo.IsTouch(event.getX(), event.getY()))  
        {  
            sentidoNave = Sentido.PRA_BAIXO;  
        }  
        else if(botao_tiro.IsTouch(event.getX(), event.getY()))  
        {  
            deuTiro = true;  
        }  
    }  
    else if(event.getAction() == MotionEvent.ACTION_UP)  
    {  
        sentidoNave = Sentido.PARADO;  
    }  
  
    return true;  
}
```

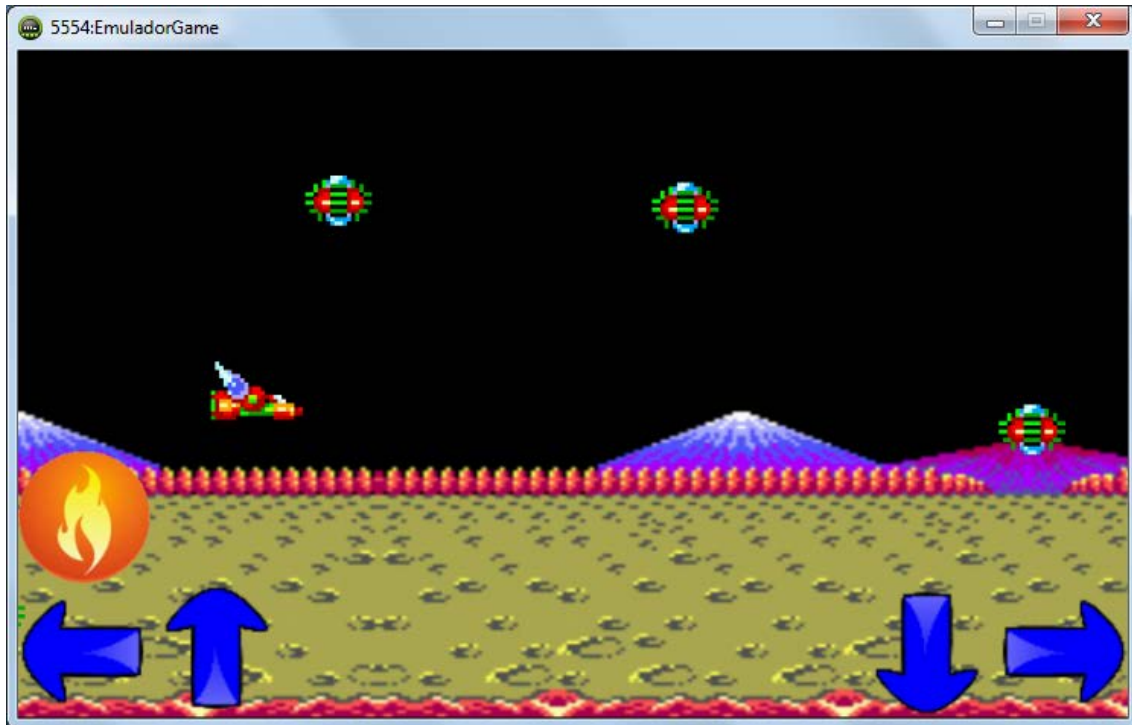


# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

Agora vamos colocar o nosso jogo para executar. O resultado você confere na figura seguinte:



**Jogo em execução**

Neste capítulo aprendemos as técnicas de detecção de colisão entre objetos, que são muito importantes em um jogo, sem ela, 85% do mesmo não acontece. No próximo capítulo iremos aprender a trabalhar com duas classes, uma responsável pelo gerenciamento de elementos no jogo (a classe **Scene**) e uma outra responsável pela criação de um personagem de um jogo (a classe **Character**).



## **Capítulo 10** Trabalhando com as classes **Scene** e **Character**

Neste capítulo iremos aprender a trabalhar com duas classes importantes, presentes no meu modelo de projeto. Uma é responsável pelo gerenciamento de elementos na tela do jogo (a classe **Scene**) e outra é muito útil para a criação de personagens que estarão presentes no jogo (a classe **Character**). Ambas as classes trabalhadas juntas, ajudam muito na construção de jogos, principalmente no estilo plataforma. Vamos conhecer elas.

### **10.1) A classe Scene**

Para começar este capítulo, irei demonstrar como utilizar a classe **Scene**, responsável pelo gerenciamento dos elementos que serão visualizados na tela do jogo. Ela possui, praticamente, quase todos os métodos que as classes anteriormente vistas e citadas (como as classes **Image** e **AnimationSprites**), porém, voltada para o gerenciamento e manipulação dos elementos presentes dentro dela.

Para demonstrar o funcionamento desta classe, vamos primeiramente importar o nosso projeto para dentro do ADT e em seguida, mudar as seguintes propriedades:

Nome do projeto : TrabalhandoComCenario

Constante “app\_name” : Trabalhando Com Cenario

Constante “title\_game\_activity” : Trabalhando Com Cenario



Depois de criado o projeto vamos adicionar dentro do diretório “drawable-mdpi” as imagens que iremos utilizar para este exemplo: “cena.png”, “bloco.png”, “chao.png”.

Depois de colocadas as imagens dentro do diretório, vamos abrir o arquivo “GameMain.java” para digitarmos o código no jogo. Para começar, vamos construir um cenário da forma tradicional como aprendemos até agora. Vamos dentro da classe **GameMain** para declararmos os seguintes atributos abaixo:

```
Image cenario;  
Image bloco, bloco2;  
Image chao;
```

Agora dentro do método **onSizeChanged** vamos digitar o seguinte código abaixo:

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {  
  
    super.onSizeChanged(w, h, oldw, oldh);  
  
    cena = new Image(context, R.drawable.cena, 0, 0, w, h);  
    chao = new Image(context, R.drawable.chao, 0, h - 80, w, 80);  
    bloco = new Image(context, R.drawable.bloco, 250, h - 200, 96, 48);  
    bloco2 = new Image(context, R.drawable.bloco, 550, h - 200, 96, 48);  
  
}
```

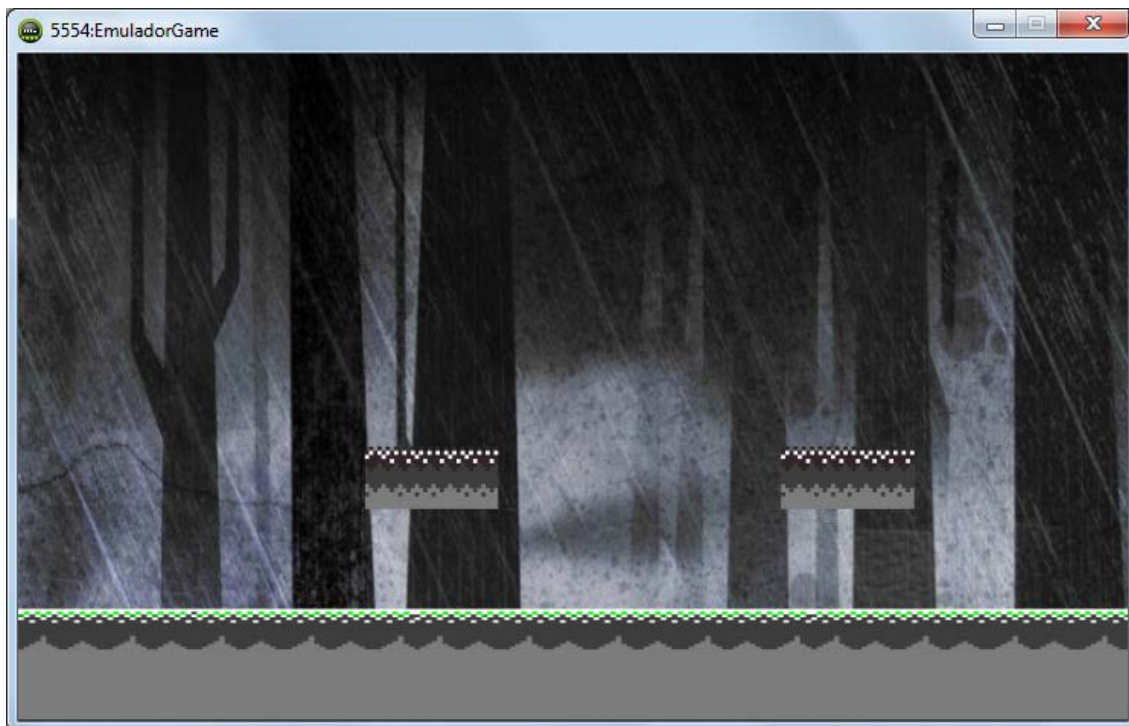
Agora dentro do método **onDraw** vamos adicionar as seguintes instruções abaixo:

```
protected void onDraw(Canvas canvas) {  
  
    super.onDraw(canvas);  
  
    Update();  
  
    canvas.drawColor(Color.BLACK);  
  
    cena.Draw(canvas);  
    chao.Draw(canvas);  
    bloco.Draw(canvas);  
    bloco2.Draw(canvas);  
  
}
```



```
}
```

Depois de escrever o código vamos executar o nosso jogo. O resultado você confere na figura seguinte:



**Jogo em execução (Visualizando o cenário)**

O cenário que construímos para o jogo acima foram feitos utilizando as técnicas apresentadas até agora. Agora vamos utilizar, conforme havia falado, a classe **Scene**, responsável pelo gerenciamento dos elementos dentro do jogo.

A classe **Scene** se encontra dentro do pacote “game.util.scene”, logo, precisaremos “importar” esse pacote para podermos utilizar esta classe. Vamos na seção de importação de pacotes para digitarmos a seguinte instrução:

```
import game.util.scene.*;
```

Dentro da classe **GameMain** vamos declarar o seguinte atributo abaixo, que será um atributo do tipo **Scene**:

```
Scene scene;
```



Agora dentro do método construtor **GameMain** vamos adicionar a seguinte linha de código:

```
scene = new Scene();
```

Dentro do método **onSizeChanged**, vamos adicionar todos os elementos que criamos dentro dele para o nosso objeto *scene*, conforme podemos conferir nas instruções abaixo:

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {  
  
    super.onSizeChanged(w, h, oldw, oldh);  
  
    cena = new Image(context, R.drawable.cena, 0, 0, w, h);  
    chao = new Image(context, R.drawable.chao, 0, h - 80, w, 80);  
    bloco = new Image(context, R.drawable.bloco, 250, h - 200, 96, 48);  
    bloco2 = new Image(context, R.drawable.bloco, 550, h - 200, 96, 48);  
  
    scene.Add(cena);  
    scene.Add(chao);  
    scene.Add(bloco);  
    scene.Add(bloco2);  
  
}
```

Irei comentar aqui as instruções que foram adicionadas. A instrução:

```
scene.Add(cena);
```

Adiciona o elemento *cena* dentro do objeto *scene*, para que ele possa ser gerenciado. O comentário é o mesmo para as outras instruções adicionadas.

Agora dentro do método **onDraw** vamos substituir as instruções:

```
cena.Draw(canvas);  
chao.Draw(canvas);  
bloco.Draw(canvas);  
bloco2.Draw(canvas);
```

Por essa única instrução:

```
scene.Draw(canvas);
```

A instrução acima desenha todos os elementos que estão dentro do objeto “scene” (que foram adicionados através do método **Add**). Os elementos dentro do objeto “scene” são desenhados na tela na sequência em que eles foram





adicionados, ou seja, se o primeiro elemento adicionado dentro do objeto foi o cenário, logo, o cenário será desenhado primeiro e assim por diante.

Se executarmos o nosso jogo iremos ter o mesmo resultado, que já foi mostrado anteriormente.

Se você observou, para cada elemento do jogo foi criado uma variável (como “*cena*”, “*chao*” e etc.). Poderíamos também construir o mesmo cenário sem utilizar nenhuma variável. Como ? Utilizando o objeto *scene* veja como isso é possível conferindo o código abaixo dentro do método **onSizeChanged**.

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {  
  
    super.onSizeChanged(w, h, oldw, oldh);  
  
    scene.Add(new Image(context, R.drawable.cena, 0, 0, w, h));  
    scene.Add(new Image(context, R.drawable.chao, 0, h - 80, w, 80));  
    scene.Add(new Image(context, R.drawable.bloco, 250, h - 200, 96, 48));  
    scene.Add(new Image(context, R.drawable.bloco, 550, h - 200, 96, 48));  
  
}
```

Como você pode observar, aos invés de criar um atributo e adicioná-lo dentro do objeto *scene* (através do método **Add**), criei a instância de cada imagem “diretamente” dentro do método **Add**, dispensando o uso dos atributos.

Quando essa técnica que mostrei será útil ? Essa técnica será de grande utilidade se os elementos que serão visíveis no jogo, não forem de alguma forma, ser utilizados em outras partes do código (como por exemplo, para avaliar a colisão de um desses elementos com algum outro objeto e etc.), caso contrário, faz-se necessário a utilização de atributos para representar o elemento do jogo.

Se executarmos o nosso jogo vamos ter o mesmo resultado, já mostrado anteriormente.

## 10.2) A classe Character

A classe **Scene**, além de ser muito útil para a construção de um cenário de um jogo (gerenciando os elementos presentes dentro dela), pode ser utilizada em conjunto com a classe **Character**, que iremos aprender a trabalhar agora.





A classe **Character** é utilizada para aqueles que desejam construir um personagem de um jogo, de forma fácil e rápida. Ela oferece uma série de recursos que facilitam a construção de um personagem.

Antes de prosseguirmos, vamos importar o nosso modelo de projeto para o ADT, e em seguida vamos mudar as seguintes propriedades:

Nome do projeto : TrabalhandoComPersonagem

Constante "app\_name" : Trabalhando Com Personagem

Constante "title\_game\_activity" : Trabalhando Com Personagem

Depois de criado o projeto vamos adicionar dentro do diretório "drawable-mdpi" as imagens que iremos utilizar para este exemplo: "cena.png", "bloco.png", "chao.png", "personagem\_parado.png", "personagem\_pulando.png", "personagem\_andando\_1.png", "personagem\_andando\_2.png", "personagem\_andando\_3.png", "personagem\_andando\_4.png", "personagem\_andando\_5.png", "personagem\_andando\_6.png", "personagem\_andando\_7.png", "seta\_esquerda.png", "seta\_direita.png", "botao\_pulo.png".

Agora dentro do código da nossa classe **GameMain** vamos digitar as seguintes instruções que serão mencionadas. Como neste jogo vamos trabalhar com as classes **Character** e **Scene**, precisamos importar os pacotes que pertencem as suas respectivas classes, logo, na seção de importação de pacotes vamos digitar as seguintes instruções abaixo:

```
import game.util.scene.*;  
import game.util.character.Character;
```

Se nós observarmos acima, fizemos duas importações, sendo uma de nível "explícito" que corresponde ao uso da classe **Character** (a instrução "game.util.character.Character). Porquê que isso foi necessário ? Pelo fato de a classe **Character** também já existir dentro do pacote padrão Java (o pacote "java.lang") que já é naturalmente (e implicitamente) "importado" para dentro do projeto em desenvolvimento.



Agora dentro da seção de declaração de atributos vamos digitar as seguintes instruções abaixo:

```
Image cena;  
Image bloco, bloco2;  
Image chao;  
  
Scene scene;  
  
Character personagem;
```

As declarações acima são iguais ao que já fizemos no exemplo anterior sobre criação de cenário. A única diferença aqui é que estamos declarando um atributo chamado *personagem*, que será o objeto que vai representar um “personagem” de um jogo (objeto do tipo **Character**).

Agora dentro do método **GameMain** vamos escrever as seguintes instruções:

```
public GameMain(Context context) {  
  
    super(context);  
    this.context = context;  
    surfaceHolder = getHolder();  
  
    scene = new Scene();  
  
    personagem = new Character(context, 100, 10, 70, 112);  
  
    personagem.AddNewSpriteIdle(R.drawable.personagem_parado);  
  
    personagem.AddNewSpriteWalking(R.drawable.personagem_andando_1);  
    personagem.AddNewSpriteWalking(R.drawable.personagem_andando_2);  
    personagem.AddNewSpriteWalking(R.drawable.personagem_andando_3);  
    personagem.AddNewSpriteWalking(R.drawable.personagem_andando_4);  
    personagem.AddNewSpriteWalking(R.drawable.personagem_andando_5);  
    personagem.AddNewSpriteWalking(R.drawable.personagem_andando_6);  
    personagem.AddNewSpriteWalking(R.drawable.personagem_andando_7);  
  
    personagem.AddNewSpriteJumping(R.drawable.personagem_pulando);  
  
    personagem.Idle(3, false);  
  
    setFocusable(true);  
}
```

Vamos analisar algumas instruções inseridas no método acima. A instrução:

```
personagem = new Character(context, 100, 10, 70, 112);
```



Carrega a instância da classe **Character** no objeto *personagem*. O primeiro argumento do construtor sempre é *context*, e o restante referem-se, respectivamente, as suas coordenadas (X e Y) e ao seu tamanho (largura e altura) na tela.

A próxima instrução:

```
personagem.AddNewSpriteIdle(R.drawable.personagem_parado);
```

Carrega no objeto uma “sprite” (imagem) que representa o nosso personagem “parado” no jogo, através do método **AddNewSpriteIdle**. Através desse método adicionamos “sprites” (que fazem parte da animação) que representam o personagem em estado de “repouso” (parado). Podemos adicionar uma imagem ou um conjunto de imagens que vai representar essa situação.

A instruções seguintes:

```
personagem.AddNewSpriteWalking(R.drawable.personagem_andando_1);  
personagem.AddNewSpriteWalking(R.drawable.personagem_andando_2);  
personagem.AddNewSpriteWalking(R.drawable.personagem_andando_3);  
personagem.AddNewSpriteWalking(R.drawable.personagem_andando_4);  
personagem.AddNewSpriteWalking(R.drawable.personagem_andando_5);  
personagem.AddNewSpriteWalking(R.drawable.personagem_andando_6);  
personagem.AddNewSpriteWalking(R.drawable.personagem_andando_7);
```

Adiciona no objeto um conjunto de sprites (animações) que representa o nosso personagem andando no jogo, através do método **AddNewSpriteWalking**. Nesse método adicionamos imagens que representam o personagem “andando” no jogo. Observe que são 7 imagens que representa a animação dele “andando”, logo, cada imagem deve ser adicionada linha a linha através do método, conforme você confere acima.

A instrução seguinte:

```
personagem.AddNewSpriteJumping(R.drawable.personagem_pulando);
```

Adiciona uma imagem que vai representar o nosso personagem “pulando” no jogo, através do método **AddNewSpriteJumping**. Nesse método adicionamos imagens que representam o personagem “pulando” no jogo, podendo essa animação ser formada por uma ou mais imagens.

A instrução seguinte:

```
personagem.Idle(3, false);
```



Dispara a animação dele em estado de repouso (parado). Neste método temos dois argumentos, sendo o primeiro equivalente ao número de frames que corresponde a troca de imagens (que na instrução acima está com o valor 3, ou seja, a cada 3 frames ele troca a imagem, caso nessa animação houver mais de uma imagem) e o segundo que indica se a animação vai estar em “loop”.

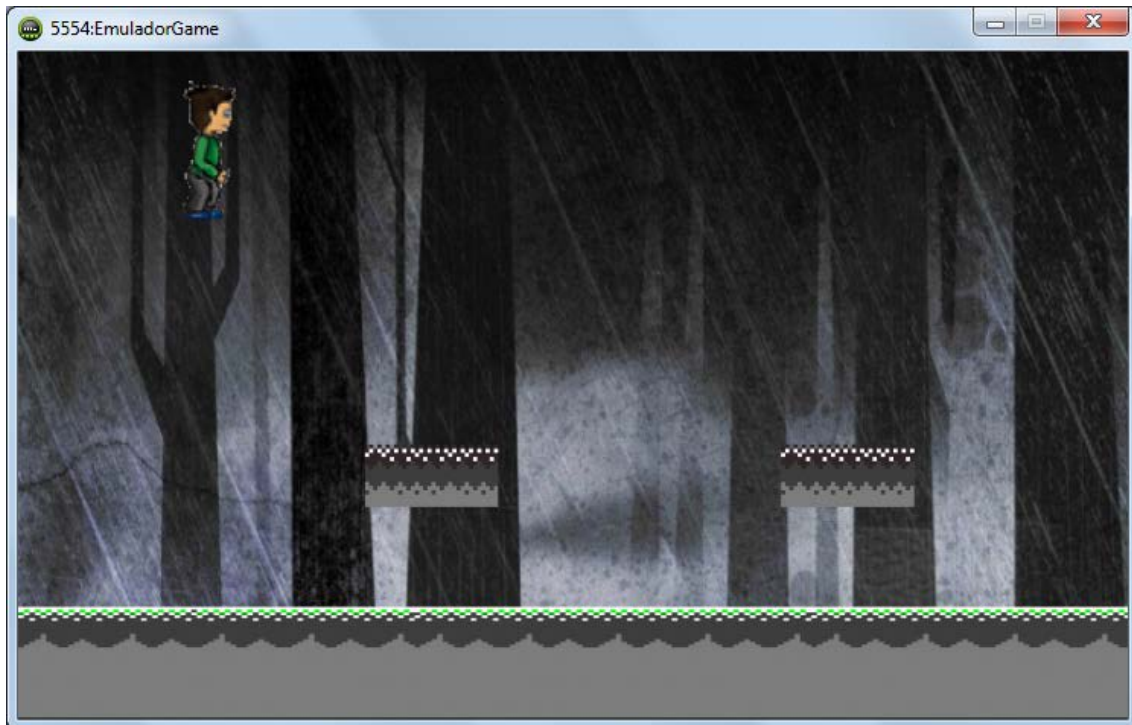
Agora dentro do método **onSizeChanged** vamos escrever o seguinte código:

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {  
  
    super.onSizeChanged(w, h, oldw, oldh);  
  
    cena = new Image(context, R.drawable.cena, 0, 0, w, h);  
    chao = new Image(context, R.drawable.chao, 0, h - 80, w, 80);  
    bloco = new Image(context, R.drawable.bloco, 250, h - 200, 96, 48);  
    bloco2 = new Image(context, R.drawable.bloco, 550, h - 200, 96, 48);  
  
    scene.Add(cena);  
    scene.Add(chao);  
    scene.Add(bloco);  
    scene.Add(bloco2);  
  
}
```

Agora dentro do método **onDraw** vamos digitar o seguinte código abaixo:

```
protected void onDraw(Canvas canvas) {  
  
    super.onDraw(canvas);  
  
    Update();  
  
    canvas.drawColor(Color.BLACK);  
  
    scene.Draw(canvas);  
    personagem.Draw(canvas);  
  
}
```

Depois de escrever todo o código solicitado acima, vamos executar o nosso jogo. Veja o resultado na figura seguinte:



**Jogo em execução (Testando o personagem)**

Se nós observarmos o jogo, o nosso personagem está lá em cima da tela. O ideal seria se ele caísse até o chão certo? A classe **Character** já possui dentro dela todo um conjunto de recursos de colisão e física dentro dele para essa situação. Vamos fazer passo a passo essa situação acontecer.

Dentro do método **Update** vamos adicionar a seguinte instrução:

```
public void Update() {  
    personagem.Update(scene);  
}
```

O método **Update** da classe **Character** é responsável por processar o pulo, a queda e a colisão do personagem com os elementos presentes dentro da cena do jogo (dentro de um objeto do tipo **Scene**). Vamos executar agora o nosso jogo para conferirmos o resultado, como mostra a figura seguinte:





**Jogo em execução (Testando o personagem)**

Olha só o que aconteceu agora : Executamos o jogo com a modificação feita acima o personagem simplesmente caiu e não parou mais. Porquê que isso aconteceu ? Simples, porque não definimos quais são os pontos (ou elementos) o personagem vai colidir, executando somente a queda dele.

Precisamos definir agora quais elementos o nosso personagem vai colidir. Vamos começar definindo como “ponto de colisão” o elemento “chão”. Para isso, vamos no método construtor **GameMain** para adicionarmos a seguinte instrução destacada **em negrito**:

:

```
personagem.Idle(3, false);  
personagem.AddCollisionElementOfFallByTag("chao");
```

Na instrução acima definimos o elemento de colisão no qual o personagem vai se colidir durante a queda do mesmo (através do método **AddCollisionElementOfFallByTag**). A definição dos elementos de colisão aqui são feitos por elementos “rotulados” (por “tags”). Observe que definimos no método um elemento rotulado como “chao”, ou seja, toda vez que durante a queda do personagem ele colidir com um elemento cujo rotulo seja chão, ele vai parar e ficar naquele ponto.



Agora vamos executar o jogo novamente para visualizarmos o resultado:



**Jogo em execução (Testando o personagem)**

Peraí ? De novo ele caiu e atravessou o chão. Por quê que isso aconteceu se definimos o “rotulo” ? Porque ainda não definimos um elemento com esse rótulo denominado “chao”. Mas a gente não tem um objeto (elemento) chamado *chao* neste jogo, que representa o chão em que o personagem vai pisar ? Sim, mas , se prestarmos atenção, o nome da nossa variável (atributo) é que se chama *chao*, porém, ela não possui um rótulo (ou “tag”) chamado “chao”. Precisamos definir para esse elemento *chao* um rótulo chamado “chao”. Para isso vamos fazer uso do método **SetTag** para definirmos um rótulo para o elemento.

Vamos no método **onSizeChanged** para definirmos a *ta* para o nosso elemento, conforme podemos conferir na instrução **em negrito**:

:

```
cena = new Image(context, R.drawable.cena, 0, 0, w, h);  
chao = new Image(context, R.drawable.chao, 0, h - 80, w, 80);
```

```
chao.SetTag("chao");
```

```
bloco = new Image(context, R.drawable.bloco, 250, h - 200, 96, 48);  
bloco2 = new Image(context, R.drawable.bloco, 550, h - 200, 96, 48)  
:
```



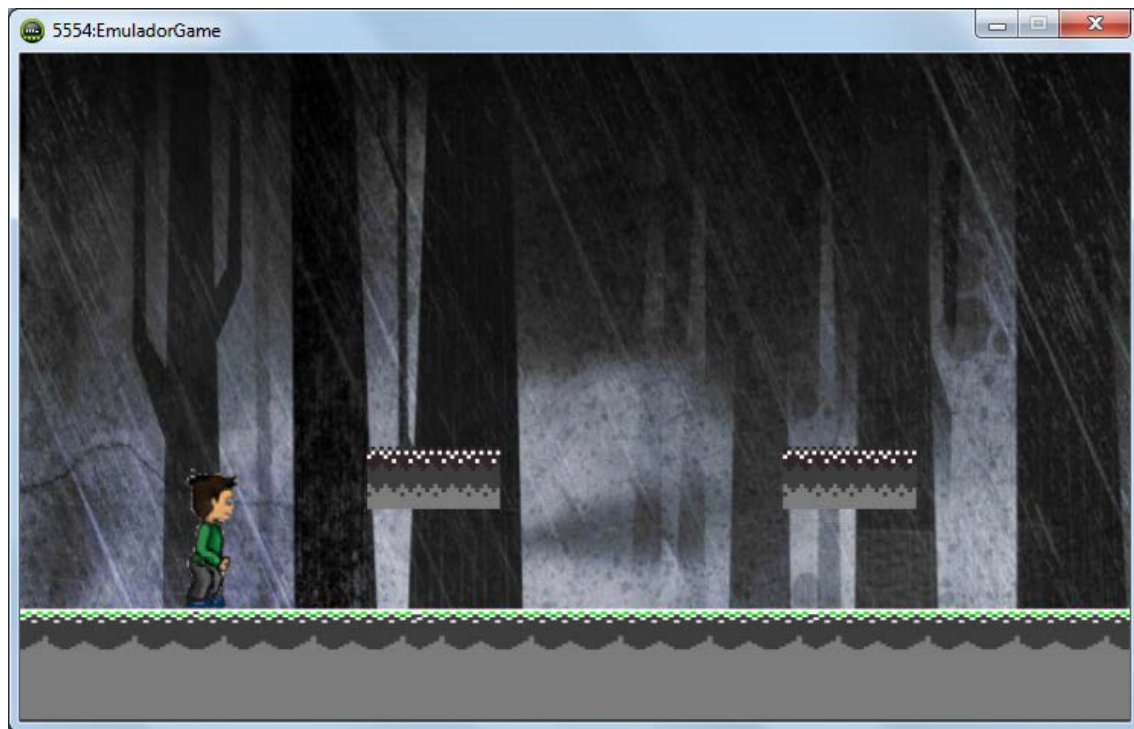


# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

Agora sim nós temos um elemento rotulado como “chao” no nosso jogo. Vamos agora executar o nosso jogo para conferirmos o resultado, como é mostrado na figura seguinte:



**Jogo em execução (Testando o personagem)**

E ai, esta entendendo como funciona esta classe ? Viu como é fácil construir um personagem com ela ? Com certeza que sim!!!

Agora vamos aprofundar um pouco mais e vamos definir os controles e o movimento para o nosso personagem.

Dentro da seção de declaração de atributos vamos realizar a seguinte declaração:

```
Image seta_direita;  
Image seta_esquerda;  
Image botao_pulo;
```

Os atributos que declaramos acima correspondem aos controles do personagem, que consiste no movimento dele para esquerda e direita, incluindo também o seu pulo.

Dentro da seção de declaração de variáveis (atributos), vamos realizar agora a seguinte declaração:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
enum Sentido { DIREITA, ESQUERDA, PARADO }
```

```
Sentido sentidoPersonagem = Sentido.PARADO;
```

O enumerador **Sentido** guardará dentro de sua estrutura os valores (rótulos) que vão representar os movimentos do personagem. O atributo *sentidoPersonagem* (do tipo **Sentido**) será a nossa variável de controle, na qual vamos definir a direção e o movimento do personagem. Por padrão, esse atributo assume o valor “Sentido.PARADO”, que indica que o personagem vai estar “parado”.

Agora dentro do método **onSizeChanged** vamos adicionar o seguinte código **em negrito**:

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
    super.onSizeChanged(w, h, oldw, oldh);

    cena = new Image(context, R.drawable.cena, 0, 0, w, h);
    chao = new Image(context, R.drawable.chao, 0, h - 80, w, 80);
    chao.SetTag("chao");
    bloco = new Image(context, R.drawable.bloco, 250, h - 200, 96, 48);
    bloco2 = new Image(context, R.drawable.bloco, 550, h - 200, 96, 48);

    seta_esquerda = new Image(context, R.drawable.seta_esquerda, 0, h -
    96, 96, 96);

    seta_direita = new Image(context, R.drawable.seta_direita, w - 96, h -
    96, 96, 96);

    botao_pulo = new Image(context, R.drawable.botao_pulo, w - (96 * 2), h
    - 96, 96, 96);

    :
```

Agora dentro do método **onDraw** vamos adicionar as linhas de código destacadas **em negrito**:

```
protected void onDraw(Canvas canvas) {

    super.onDraw(canvas);

    Update();

    canvas.drawColor(Color.BLACK);

    scene.Draw(canvas);
    personagem.Draw(canvas);
    seta_direita.Draw(canvas);
    seta_esquerda.Draw(canvas);
    botao_pulo.Draw(canvas);

}
```



Agora dentro do método **onTouchEvent** vamos adicionar o seguinte código:

```
public boolean onTouchEvent(MotionEvent event) {  
    if(event.getAction() == MotionEvent.ACTION_DOWN)  
    {  
        if(seta_direita.IsTouch(event.getX(), event.getY()))  
        {  
            if((sentidoPersonagem == Sentido.PARADO) || (sentidoPersonagem ==  
                Sentido.ESQUERDA))  
            {  
                personagem.WalkingToRight(2, true);  
                sentidoPersonagem = Sentido.DIREITA;  
            }  
            else {  
                personagem.Idle(3, false);  
                sentidoPersonagem = Sentido.PARADO;  
            }  
        }  
        else if(seta_esquerda.IsTouch(event.getX(), event.getY()))  
        {  
            if((sentidoPersonagem == Sentido.PARADO) || (sentidoPersonagem ==  
                Sentido.DIREITA))  
            {  
                personagem.WalkingToLeft(2, true);  
                sentidoPersonagem = Sentido.ESQUERDA;  
            }  
            else {  
                personagem.Idle(3, false);  
                sentidoPersonagem = Sentido.PARADO;  
            }  
        }  
        else if(botao_pulo.IsTouch(event.getX(), event.getY()))  
        {  
            personagem.Jump(3, true);  
        }  
        return true;  
    }  
}
```

Dentro de cada condição acima, relativa ao seu movimento (direita , esquerda e parado), quando verdadeira, dispara a animação dele (“andando”, pelos métodos **WalkingToRight** e **WalkingToLeft** ou “parado” pelo método **Idle**) antes da atualização do status do sentido da direção.

Agora dentro do método **Update** vamos adicionar a seguinte linha de código **em negrito**:

```
public void Update() {  
    if(sentidoPersonagem == Sentido.DIREITA)  
        personagem.MoveByX(10);  
    else if(sentidoPersonagem == Sentido.ESQUERDA)  
        personagem.MoveByX(-10);  
    personagem.Update(scene);  
}
```

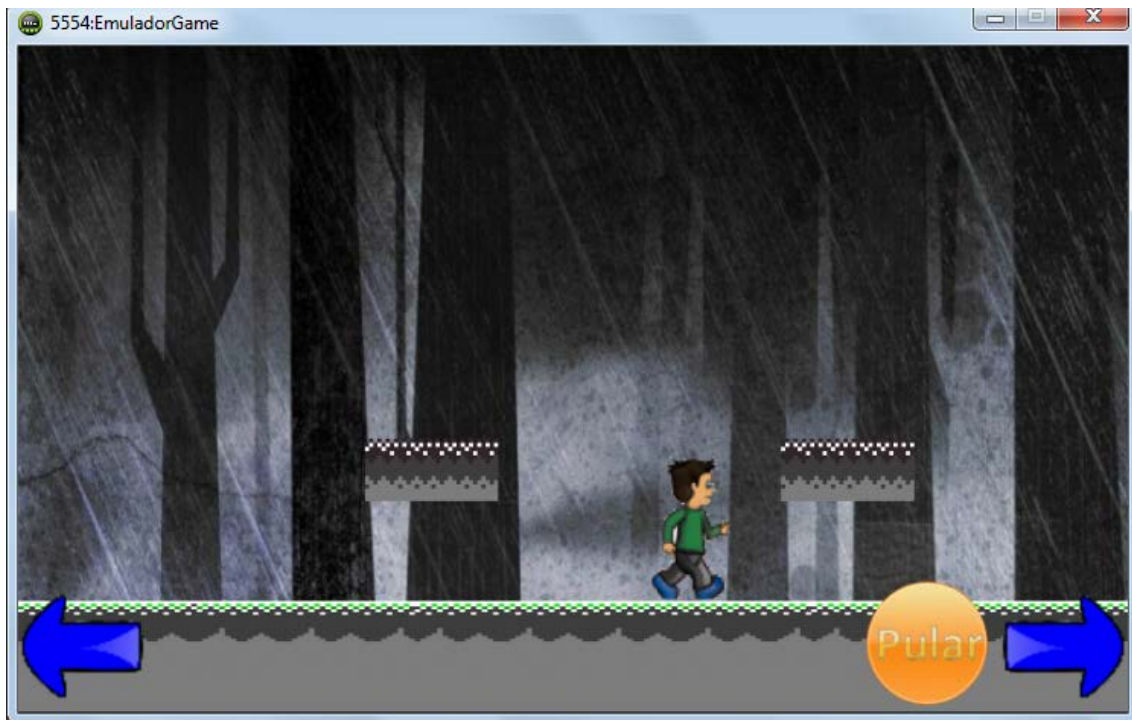


# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

O código adicionado no evento **onTouchEvent** atualiza os status do movimento e exibe a animação da ação tomada, e já aqui no **Update**, o código adicionado realiza o deslocamento do personagem de acordo com seu status.



**Jogo em execução (Testando o personagem)**

Se testarmos o jogo acima, podemos verificar que se deixarmos o personagem andando “constantemente” para uma das direções ele some e não volta mais (mesmo a gente tentando voltar a mostrar ele na tela). Por quê isso acontece ? Isso acontece porque não desenvolvemos nenhum código que impedisse o personagem a ficar somente na tela. Vamos adicionar esse código ? Para começar, vamos primeiramente declarar um atributo chamado *largura\_tela*, conforme você pode conferir abaixo:

```
int largura_tela;
```

Esse variável irá armazenar a largura da tela do dispositivo (que é conseguida dentro do método **onSizeChanged**). Dentro do método **onSizeChanged** vamos adicionar a seguinte instrução abaixo:

```
largura_tela = w;
```



Agora dentro do método **Update** vamos atualizar o código existente pelo seguinte código **em negrito**:

```
public void Update() {  
    if(sentidoPersonagem == Sentido.DIREITA)  
    {  
        personagem.MoveByX(10);  
  
        if(personagem.GetX() > (largura_tela - personagem.GetWidth()))  
        {  
            personagem.SetX(largura_tela - personagem.GetWidth());  
            personagem.Idle(3, false);  
            sentidoPersonagem = Sentido.PARADO;  
        }  
    }  
    else if(sentidoPersonagem == Sentido.ESQUERDA)  
    {  
        personagem.MoveByX(-10);  
  
        if(personagem.GetX() < 0)  
        {  
            personagem.SetX(0);  
            personagem.Idle(3, false);  
            sentidoPersonagem = Sentido.PARADO;  
        }  
    }  
  
    personagem.Update(scene);  
}
```

Li comentar agora as instruções dentro de cada condição de movimento do personagem. Quando o personagem se movimenta para a direita, existe a seguinte condição:

```
if(personagem.GetX() > (largura_tela - personagem.GetWidth()))
```

Que verifica se o personagem ultrapassou o limite da tela pela direita. Caso a condição seja verdadeira, o personagem “para”, fazendo com que o mesmo não passe da tela, conforme é realizado através do código abaixo (dentro da condição):

```
personagem.SetX(largura_tela - personagem.GetWidth());  
personagem.Idle(3, false);  
sentidoPersonagem = Sentido.PARADO;
```





# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

Quando o personagem se movimenta para a esquerda, existe a seguinte condição:

```
if(personagem.GetX() < 0)
```

Que verifica se o personagem ultrapassou o limite da tela pela esquerda. Caso a condição seja verdadeira, o personagem “pára”, fazendo com que o mesmo não passe da tela, conforme é realizado através do código abaixo (dentro da condição):

```
personagem.SetX(0);  
personagem.Idle(3, false);  
sentidoPersonagem = Sentido.PARADO;
```

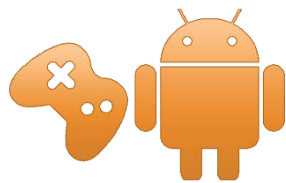
Vamos agora executar o nosso jogo para testarmos os resultados:



**Jogo em execução (Testando o personagem)**

Neste capítulo aprendemos a trabalhar com as classes **Scene** e **Character**, que podem ser muito utilizadas para a construção de jogos no estilo plataforma.

Nos capítulos seguintes iremos , de acordo com os exemplos apresentados, “aprofundar” um pouco na utilização dessas classes, explorando as suas funcionalidades e recursos.



## Capítulo 11 Trabalhando com HUD

**E**m qualquer jogo de qualquer gênero, normalmente existe no canto superior ou inferior da tela aquelas informações que são exibidas à respeito do personagem ou do jogo em si (como total de vidas, score, itens adquiridos e etc.). Essas informações são conhecidas como HUD (ou Head-UP Display) e são muito úteis para o jogador, para que o mesmo se oriente no jogo. Neste capítulo iremos aprender a construir uma HUD para que possamos implementar em nossos jogos.

### 11.1) Criando uma HUD simples

Antes de começarmos vamos importar o modelo de projeto para o ADT e em seguida vamos modificar as seguintes propriedades:

Nome do projeto : TrabalhandoComHUD

Constante “app\_name” : Trabalhando Com HUD

Constante “title\_game\_activity” : Trabalhando Com HUD

Depois de criado o projeto vamos adicionar dentro do diretório “drawable-mdpi” as imagens que iremos utilizar para este exemplo: “player.png”, “item.png”, “seta\_esquerda.png” e “seta\_direita.png”.

Agora vamos abrir o arquivo “GameMain.java” para escrevermos o código deste jogo. Primeiramente, vamos importar um pacote que será necessário para utilizarmos uma classe que será responsável por formatar o estilo, tamanho e tipo de fonte que utilizaremos para exibir as informações na tela, a classe **Paint**. Ela se encontra dentro do pacote “android.graphics”, logo, iremos importar esse pacote para o nosso programa, conforme você confere abaixo:

```
import android.graphics.*;
```





Também para o nosso programa, faremos uso da classe **Scene**, que se encontra dentro do pacote “game.util.scene”, logo, vamos importar esse pacote.

Agora dentro da seção de declaração de atributos da classe **GameMain** , vamos digitar as seguintes instruções abaixo:

```
Image player;
Image seta_direita, seta_esquerda;
Scene scene;
Paint p;
int total_itens = 0;

enum Sentido { DIREITA, ESQUERDA, PARADO }
Sentido sentidoPersonagem = Sentido.PARADO;
```

Agora dentro do método construtor **GameMain** vamos digitar o seguinte código abaixo:

```
public GameMain(Context context) {

    super(context);
    this.context = context;
    surfaceHolder = getHolder();

    scene = new Scene();

    p = new Paint();
    p.setStyle(Style.FILL_AND_STROKE);
    p.setTypeface(Typeface.create(Typeface.SANS_SERIF, Typeface.NORMAL));
    p.setTextSize(30);
    p.setColor(Color.BLACK);

    setFocusable(true);
}
```

No método acima, pelo que podemos ver, carregamos a instância do objeto *p* do tipo **Paint**. Após o carregamento de sua instância, definimos o estilo de preenchimento através do método **setStyle** (com o argumento “Style.FILL\_AND\_STROKE”). Em seguida definimos o tipo de fonte, através do método **setTypeface**. Nesse comando definimos uma fonte “SANS SERIF” (Uma das fontes padrões do Android) com estilo normal (sem negrito e itálico), através do método **create** da classe **Typeface** , passado no argumento do método **setTypeface**, como podemos conferir abaixo:

```
p.setTypeface(Typeface.create(Typeface.SANS_SERIF, Typeface.NORMAL));
```



Em seguida definimos o tamanho da fonte, através do método **setTextSize**, cujo valor será 30. E para finalizar, definimos a cor do texto que será preto, através do método **setColor**, com o argumento "Color.BLACK".

Agora dentro do método **onSizeChanged** vamos escrever o seguinte código:

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {  
  
    super.onSizeChanged(w, h, oldw, oldh);  
  
    player = new Image(context, R.drawable.player, 80, 220, 48, 48);  
  
    scene.Add(new Image(context, R.drawable.item, 200, 220, 48, 48,"item"));  
    scene.Add(new Image(context, R.drawable.item, 300, 220, 48, 48,"item"));  
    scene.Add(new Image(context, R.drawable.item, 400, 220, 48, 48,"item"));  
    scene.Add(new Image(context, R.drawable.item, 500, 220, 48, 48,"item"));  
    scene.Add(new Image(context, R.drawable.item, 600, 220, 48, 48,"item"));  
  
    seta_esquerda = new Image(context, R.drawable.seta_esquerda, 0, h - 96,  
    96, 96);  
    seta_direita = new Image(context, R.drawable.seta_direita, w - 96, h - 96,  
    96, 96);  
  
}
```

Se observarmos o código acima, utilizamos o método **Add** do objeto *scene* para adicionarmos os itens que serão visualizados na tela, conforme podemos ver na instrução seguinte:

```
scene.Add(new Image(context, R.drawable.item, 200, 220, 48, 48,"item"));
```

Todos os itens adicionados na tela possui uma tag chamada "item" (ultimo parâmetro do método construtor da classe **Image**), que será utilizado como "referência" durante o processamento da colisão do personagem com o item, para saber se o mesmo foi capturado.

Agora no método **onDraw** vamos escrever o seguinte código abaixo:

```
protected void onDraw(Canvas canvas) {  
  
    super.onDraw(canvas);  
  
    Update();  
  
    canvas.drawColor(Color.WHITE);  
    player.Draw(canvas);  
    scene.Draw(canvas);  
    seta_esquerda.Draw(canvas);  
    seta_direita.Draw(canvas);  
    canvas.drawText("Itens = " + total_itens, 10, 30, p);  
  
}
```



```
}
```

Irei comentar agora uma instrução acima, responsável por desenha as informações na tela. A instrução:

```
canvas.drawText("Itens = " + total_itens, 10, 30, p);
```

É responsável por mostrar as informações (texto) a serem exibidas na tela, através do método **drawText** do objeto *canvas*. O primeiro argumento do método é a frase ou conteúdo a ser exibido, que no caso será o total de itens que foram obtidos no jogo. No segundo e terceiro parâmetro definimos, respectivamente, as coordenadas X e Y de onde a informação será exibida na tela. O último parâmetro definimos um argumento do tipo **Paint**, que no caso será aquele objeto que chamamos de *p*, que armazena todas as configurações do texto que será exibido na tela (tamanho, cor, estilo e etc.).

Agora dentro da classe **GameMain** vamos criar um novo método. Esse método será responsável por “processar” a colisão dele com os outros elementos (itens) na tela. Vamos chamar esse método de **ProcessarColisao**, e seu código se encontra abaixo:

```
public void ProcessaColisao()
{
    for(GameElement e : scene.Elements())
    {
        if(e.GetTag() == "item")
        {
            if(Collision.Check(player, e))
            {
                scene.Remove(e);
                total_itens++;
                break;
            }
        }
    }
}
```

Agora dentro do método **Update** vamos digitar o seguinte código:

```
public void Update() {
    if(sentidoPersonagem == Sentido.DIREITA)
    {
        player.MoveByX(10);

        //Processa a colisão dele com os elementos
        ProcessaColisao();
    }
    else if(sentidoPersonagem == Sentido.ESQUERDA)
```



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
{
    player.MoveByX(-10);

    //Processa a colisão dele com os elementos
    ProcessaColisao();
}
}
```

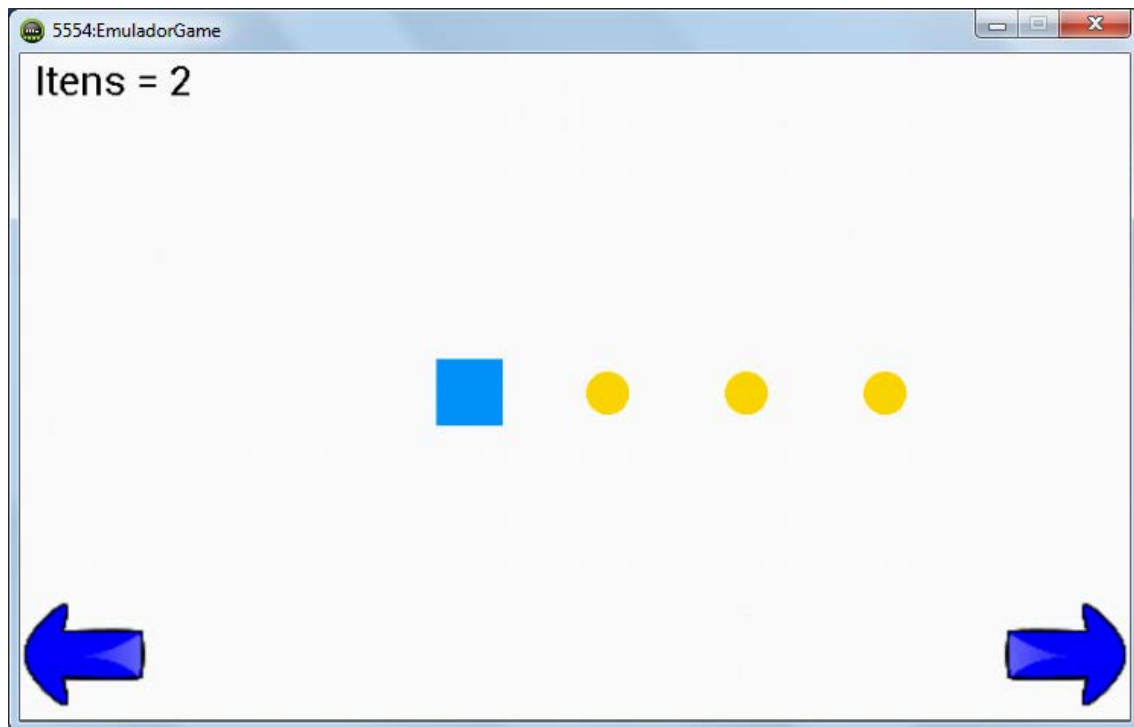
Agora dentro do método **onTouchEvent** vamos escrever o seguinte código abaixo:

```
public boolean onTouchEvent(MotionEvent event) {

    if(event.getAction() == MotionEvent.ACTION_DOWN)
    {
        if(seta_esquerda.IsTouch(event.getX(), event.getY()))
            sentidoPersonagem = Sentido.ESQUERDA;
        else if(seta_direita.IsTouch(event.getX(), event.getY()))
            sentidoPersonagem = Sentido.DIREITA;
    }
    else if(event.getAction() == MotionEvent.ACTION_UP)
    {
        sentidoPersonagem = Sentido.PARADO;
    }

    return true;
}
```

Depois de escrever todos os códigos acima, vamos colocar o nosso jogo acima para executar. O resultado você confere na figura seguinte:



**Jogo em execução (Mostrando itens capturados na tela)**

### 11.2) Criando uma HUD personalizada

Aqui agora neste tópico iremos construir uma HUD personalizada (onde além de trabalharmos com textos, iremos também exibir imagens). Para essa HUD Personalizada iremos construir uma classe que irá desenhar essa HUD na tela.

Para começar vamos importar o modelo de projeto para o ADT e em seguida , alterar as seguintes propriedades dele:

Nome do projeto : HUDPersonalizada

Constante "app\_name" : HUD Personalizada

Constante "title\_game\_activity" : HUD Personalizada

Agora vamos adicionar as seguintes imagens no projeto : "cenario1.png", "cenario2.png", "mickey\_parado\_1.png", "mickey\_parado\_2.png", "mickey\_andando\_1.png", "mickey\_andando\_2.png", "mickey\_andando\_3.png", "mickey\_andando\_4.png", "mickey\_andando\_5.png", "mickey\_andando\_6.png", "mickey\_pulando.png", "item\_maca.png", "chao.png", "seta\_direita.png", "seta\_esquerda.png", "botao\_pulo.png".



Depois de adicionada as imagens no projeto vamos carregar o arquivo “GameMain.java” para escrevermos o código do nosso jogo. Primeiramente, vamos realizar a seguinte “importação” de pacotes , que iremos utilizar em nosso projeto:

```
import game.util.scene.*;
import game.util.character.Character;
```

Dentro da seção de declaração de atributos, vamos escrever o seguinte código abaixo:

```
Image cenario1, cenario2;

Character mickey;

Image seta_direita, seta_esquerda, botao_pulo;

Scene scene;

enum Sentido { DIREITA, ESQUERDA, PARADO }
Sentido sentidoMickey = Sentido.PARADO;
```

Dentro do método construtor **GameMain** vamos escrever o seguinte código:

```
public GameMain(Context context) {

    super(context);
    this.context = context;
    surfaceHolder = getHolder();

    scene = new Scene();

    setFocusable(true);
}
```

Dentro do método **onSizeChanged** vamos adicionar o seguinte código:

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
    super.onSizeChanged(w, h, oldw, oldh);

    mickey = new Character(context, (w / 2) - (93 / 2), 20, 93, 103);

    mickey.AddNewSpriteIdle(R.drawable.mickey_parado_1);
    mickey.AddNewSpriteIdle(R.drawable.mickey_parado_1);
    mickey.AddNewSpriteIdle(R.drawable.mickey_parado_1);
    mickey.AddNewSpriteIdle(R.drawable.mickey_parado_1);
    mickey.AddNewSpriteIdle(R.drawable.mickey_parado_2);

    mickey.AddNewSpriteWalking(R.drawable.mickey_andando_1);
    mickey.AddNewSpriteWalking(R.drawable.mickey_andando_2);
    mickey.AddNewSpriteWalking(R.drawable.mickey_andando_3);
    mickey.AddNewSpriteWalking(R.drawable.mickey_andando_4);
    mickey.AddNewSpriteWalking(R.drawable.mickey_andando_5);
}
```



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
mickey.AddNewSpriteWalking(R.drawable.mickey_andando_6);
mickey.AddNewSpriteJumping(R.drawable.mickey_pulando);

mickey.Idle(3, true);
mickey.AddCollisionElementOfFallByTag("chao");

cenario1 = new Image(context, R.drawable.cenario1, 0, 0, w, h);
cenario2 = new Image(context, R.drawable.cenario2, w, 0, w, h);

scene.Add(cenario1);
scene.Add(cenario2);

scene.Add(new Image(context, R.drawable.chao, 0, h - 40, w, 40, "chao"));

seta_esquerda = new Image(context, R.drawable.seta_esquerda, 0, h - 96, 96, 96);
seta_direita = new Image(context, R.drawable.seta_direita, w - 96, h - 96, 96, 96);

botao_pulo = new Image(context, R.drawable.botao_pulo, w - (96 * 2), h - 96, 96, 96);

//Adiciona os itens
scene.Add(new Image(context, R.drawable.item_maca, 500, h - 40 - 40, 40, 40, "item"));
scene.Add(new Image(context, R.drawable.item_maca, 600, h - 40 - 40, 40, 40, "item"));
scene.Add(new Image(context, R.drawable.item_maca, 700, h - 40 - 40, 40, 40, "item"));

scene.Add(new Image(context, R.drawable.item_maca, 1200, h - 40 - 40, 40, 40, "item"));
scene.Add(new Image(context, R.drawable.item_maca, 1300, h - 40 - 40, 40, 40, "item"));
scene.Add(new Image(context, R.drawable.item_maca, 1400, h - 40 - 40, 40, 40, "item"));

scene.Add(new Image(context, R.drawable.item_maca, 1700, h - 40 - 40, 40, 40, "item"));
scene.Add(new Image(context, R.drawable.item_maca, 1800, h - 40 - 40, 40, 40, "item"));
scene.Add(new Image(context, R.drawable.item_maca, 1900, h - 40 - 40, 40, 40, "item"));

}
```

Vamos observar o seguinte trecho de código inserido no método acima. Observe que quando carregamos as imagens que vão representar a animação dele em estado de repouso (parado), fizemos uso das seguintes instruções:

```
mickey.AddNewSpriteIdle(R.drawable.mickey_parado_1);
mickey.AddNewSpriteIdle(R.drawable.mickey_parado_1);
mickey.AddNewSpriteIdle(R.drawable.mickey_parado_1);
mickey.AddNewSpriteIdle(R.drawable.mickey_parado_1);
mickey.AddNewSpriteIdle(R.drawable.mickey_parado_2);
```





Observe que “repetimos” a instrução que adiciona a imagem “mickey\_parado\_1” quatro vezes, mas por quê fiz isso ? Justamente para dar um “intervalo” maior, pois, nessa animação dele em repouso ele pisca os olhos, logo, é preciso que o personagem fique de olhos abertos por um bom tempo antes de fechar (a imagem dele de olhos fechados está em “mickey\_parado\_2”).

Agora dentro do método **onDraw** vamos adicionar as seguintes instruções:

```
protected void onDraw(Canvas canvas) {  
  
    super.onDraw(canvas);  
  
    Update();  
    canvas.drawColor(Color.BLACK);  
    scene.Draw(canvas);  
    mickey.Draw(canvas);  
    seta_esquerda.Draw(canvas);  
    seta_direita.Draw(canvas);  
    botao_pulo.Draw(canvas);  
}
```

Agora dentro desta classe **GameMain** vamos criar um método chamado **ProcessaColisao**, que vai conter as seguintes instruções:

```
public void ProcessaColisao()  
{  
  
    for(GameElement e : scene.Elements())  
    {  
        if(e.GetTag() == "item")  
        {  
            if(Collision.Check(mickey, e))  
            {  
                scene.Remove(e);  
                break;  
            }  
        }  
    }  
}
```

Agora dentro do método **Update** vamos adicionar as seguintes instruções:

```
public void Update() {  
  
    if(sentidoMickey == Sentido.DIREITA)  
    {  
        cenario1.MoveByX(-12);  
    }  
}
```



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
cenario2.MoveByX(-12);

//Move todos os itens (maças)
for(GameElement e : scene.Elements())
{
    if(e.GetTag()=="item")
        e.MoveByX(-12);
}

if(cenario1.GetX() < -cenario1.GetWidth())
    cenario1.SetX(cenario2.GetX() + cenario2.GetWidth());

if(cenario2.GetX() < -cenario2.GetWidth())
    cenario2.SetX(cenario1.GetX() + cenario1.GetWidth());

ProcessaColisao();
}
else if(sentidoMickey == Sentido.ESQUERDA)
{
    cenario1.MoveByX(12);
    cenario2.MoveByX(12);

    //Move todos os itens (maças)
    for(GameElement e : scene.Elements())
    {
        if(e.GetTag()=="item")
            e.MoveByX(12);
    }

    if(cenario1.GetX() > 800)
        cenario1.SetX(cenario2.GetX() - cenario2.GetWidth());

    if(cenario2.GetX() > 800)
        cenario2.SetX(cenario1.GetX() - cenario1.GetWidth());

    ProcessaColisao();
}

mickey.Update(scene);
}
```

E para finalizar, vamos adicionar dentro do método **onTouchEvent** o seguinte código:

```
public boolean onTouchEvent(MotionEvent event) {

    if(event.getAction() == MotionEvent.ACTION_DOWN)
    {
        if(seta_esquerda.IsTouch(event.getX(), event.getY()))
        {
            if((sentidoMickey == Sentido.PARADO) || (sentidoMickey ==
                Sentido.DIREITA))
            {

```



# Apostila de Android

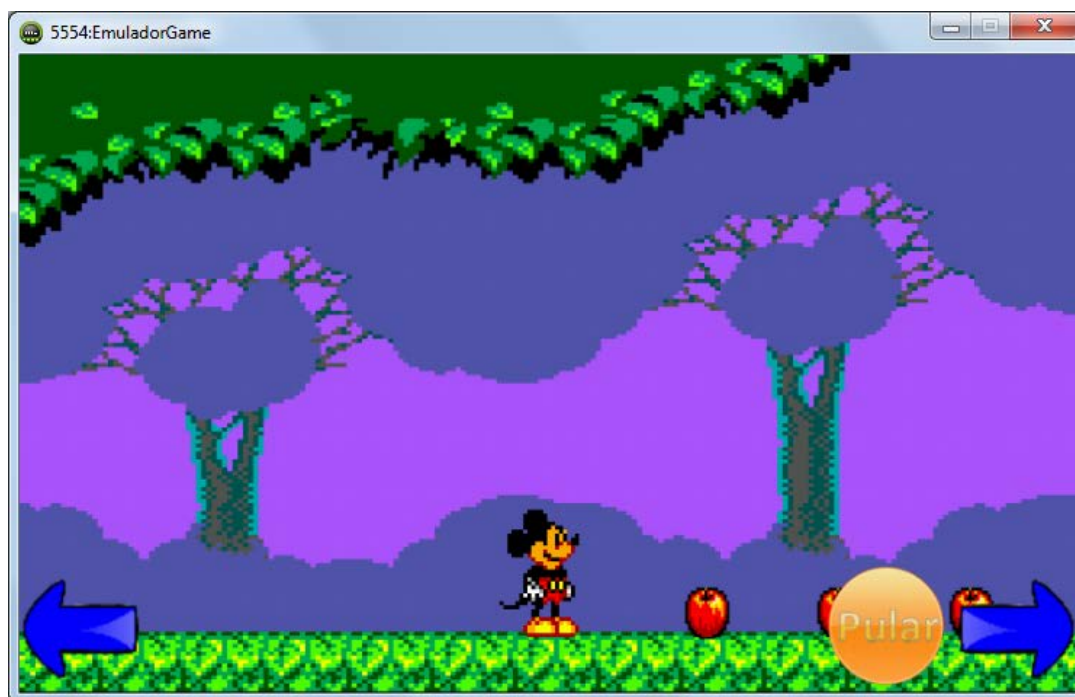
## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
        sentidoMickey = Sentido.ESQUERDA;
        mickey.WalkingToLeft(3, true);
    }
    else {
        sentidoMickey = Sentido.PARADO;
        mickey.Idle(3, true);
    }
}
else if(seta_direita.IsTouch(event.getX(), event.getY()))
{
    if((sentidoMickey == Sentido.PARADO) || (sentidoMickey ==
        Sentido.ESQUERDA))
    {
        sentidoMickey = Sentido.DIREITA;
        mickey.WalkingToRight(3, true);
    }
    else {
        sentidoMickey = Sentido.PARADO;
        mickey.Idle(3, true);
    }
}
else if(botao_pulo.IsTouch(event.getX(), event.getY()))
{
    mickey.Jump(3, false);
}
}

return true;
}
```

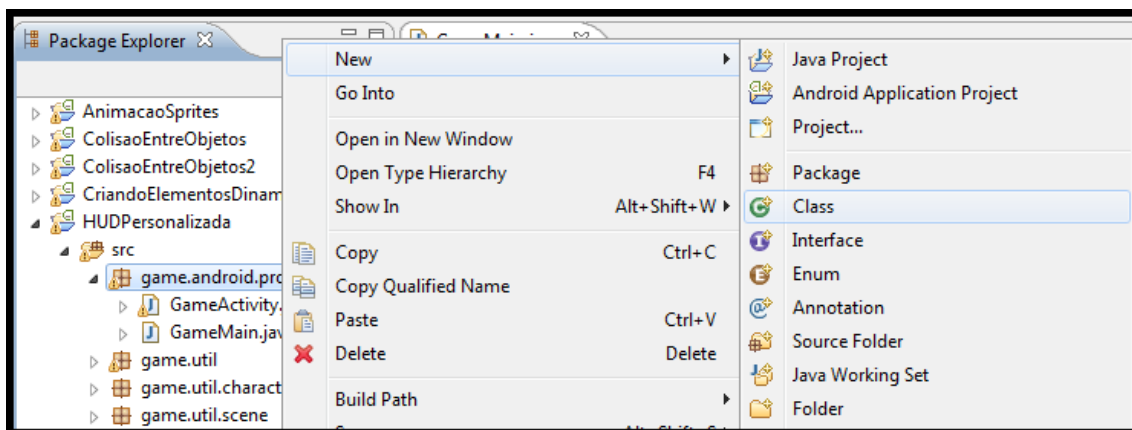
Vamos executar o nosso jogo ? O resultado você confere na figura seguinte:



Jogo em execução (Sem HUD na tela)



Se testarmos o jogo acima ele irá funcionar perfeitamente (o movimento/ação do personagem e a captura dos itens), porém, faltou adicionarmos a nossa “HUD personalizada”. Para construirmos essa HUD iremos fazer o uso de classes, ou seja, iremos criar uma classe que vai desenhar essa HUD. Para criarmos uma classe em nosso projeto Android, basta clicar com o botão direito sobre o pacote “game.android.project” (situado dentro do diretório “src” do nosso projeto) e em seguida selecionar “New” / “Class”, conforme demonstra a figura seguinte:



**Criando uma classe**

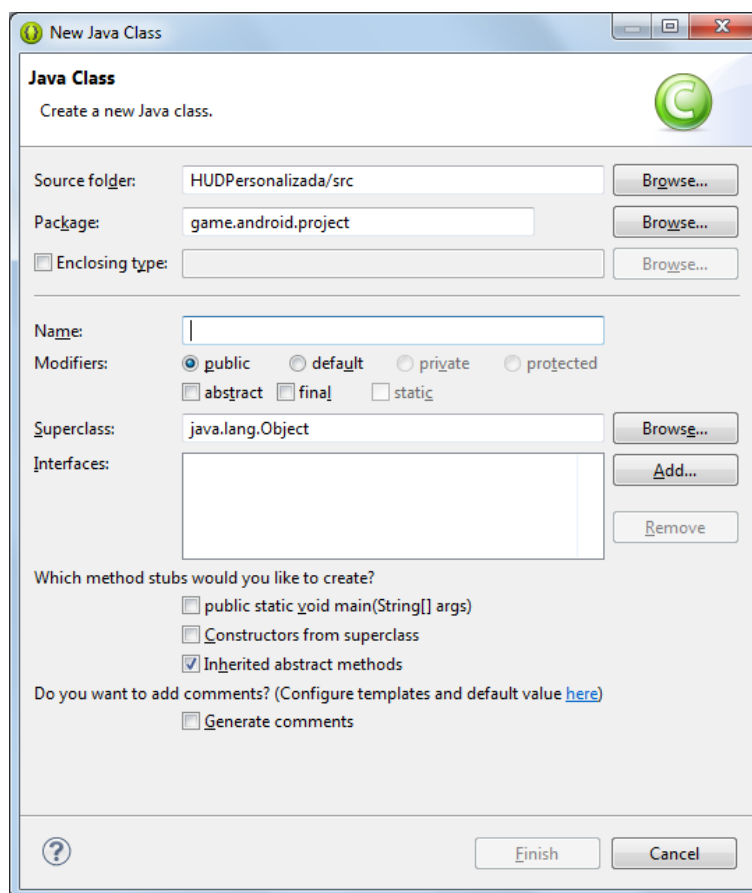
Feito isso será aberta a seguinte caixa de diálogo:



# Apostila de Android

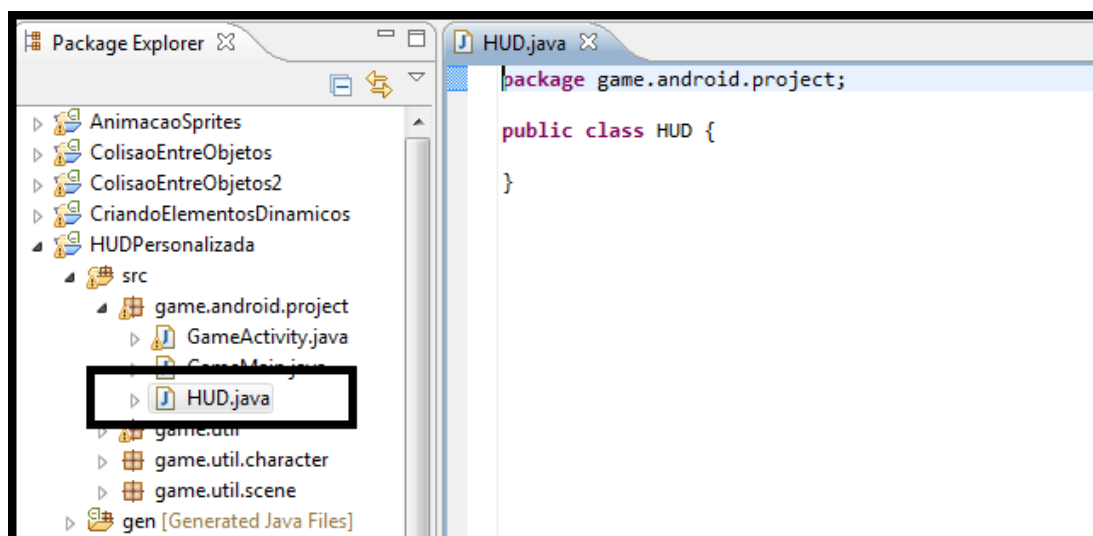
## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



**Caixa de diálogo – New Java Class**

No campo “Name” (onde informamos o nome da classe a ser criada), vamos digitar o nome da nossa classe que vai se chamar “HUD” (tudo em maiúsculo e sem aspas, é claro). Feito isso vamos clicar no botão “Finish” para que a classe possa ser gerada, conforme você pode conferir na figura seguinte:



**Classe HUD criada (Observe o destaque)**



Agora irei colocar aqui o código completo da nossa classe HUD, que vai te que ser digitado dentro do arquivo (classe) gerado:

```
package game.android.project;

import android.content.*;
import android.graphics.*;
import android.graphics.Paint.Style;
import game.util.*;

public class HUD {

    Paint p;
    Image icone;

    int total_itens;

    public HUD(Context context, int x, int y)
    {
        icone = new Image(context, R.drawable.item_maca, x, y, 50, 50);
        p = new Paint();
        p.setStyle(Style.FILL_AND_STROKE);
        p.setTypeface(Typeface.create(Typeface.SANS_SERIF,
        Typeface.BOLD));
        p.setTextSize(50);
        p.setColor(Color.WHITE);
    }

    public void Draw(Canvas canvas)
    {
        icone.Draw(canvas);
        canvas.drawText(" x " + total_itens, icone.GetX() + 55,
        icone.GetY() + 45, p);
    }

    public void SetTotalItens(int total_itens)
    {
        this.total_itens = total_itens;
    }
}
```

Se observarmos o código da classe acima, existe um método chamado **SetTotalItens** responsável por definir o valor do total de itens que será exibido na tela através da HUD.

Agora vamos voltar para a classe **GameMain** e vamos declarar os seguintes atributos.

```
HUD hud;
int total_itens = 0;
```



Dentro do construtor da classe vamos digitar as seguintes instruções **em negrito**:

```
public GameMain(Context context) {  
    super(context);  
    this.context = context;  
    surfaceHolder = getHolder();  
  
    scene = new Scene();  
    hud = new HUD(context, 30, 30);  
    hud.SetTotalItens(total_itens);  
  
    setFocusable(true);  
}
```

Dento do método **onDraw** vamos colocar agora o seguinte código destacado **em negrito**:

```
protected void onDraw(Canvas canvas) {  
  
    super.onDraw(canvas);  
  
    Update();  
  
    canvas.drawColor(Color.BLACK);  
    scene.Draw(canvas);  
    mickey.Draw(canvas);  
    seta_esquerda.Draw(canvas);  
    seta_direita.Draw(canvas);  
    botao_pulo.Draw(canvas);  
  
    hud.Draw(canvas);  
  
}
```

Agora dentro do método **ProcessaColisao** vamos adicionar o seguinte método destacado **em negrito**:

```
public void ProcessaColisao()  
{  
    for(GameElement e : scene.Elements())  
    {  
        if(e.GetTag() == "item") {  
            if(Collision.Check(mickey, e)) {  
                scene.Remove(e);  
                total_itens++;  
                hud.SetTotalItens(total_itens);  
  
                break;  
            }  
        }  
    }  
}
```





# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

Agora dentro do método **Update** iremos colocar as seguintes instruções destacadas **em negrito**:

```
public void Update() {  
    if(sentidoMickey == Sentido.DIREITA)  
    {  
        :  
        :  
        if(cenario2.GetX() < -cenario2.GetWidth())  
            cenario2.SetX(cenario1.GetX() + cenario1.GetWidth());  
  
        ProcessaColisao();  
    }  
    else if(sentidoMickey == Sentido.ESQUERDA)  
    {  
        :  
        :  
        if(cenario2.GetX() > 800)  
            cenario2.SetX(cenario1.GetX() - cenario1.GetWidth());  
  
        ProcessaColisao();  
    }  
  
    mickey.Update(scene);  
}
```

Vamos executar o nosso jogo agora ? Acompanhe como ficou o resultado na figura seguinte:



Jogo em execução (Com HUD na tela)



## **Capítulo 12 Programação com Inteligência Artificial (I.A)**

**A** Inteligência Artificial é um dos ramos da computação que trata do comportamento inteligente. Na programação com jogos, essa técnica é fortemente empregada para programar movimentos e ações dos elementos dos jogos que precisam de “Inteligência” para atuar (como por exemplo, o movimento de um inimigo para “perseguir” um personagem). Neste capítulo iremos aprender como programar com I.A para utilizarmos em nossos jogos, para torná-lo mais divertidos e interessantes.

Até agora os jogos que desenvolvemos não trabalhou com nenhuma técnica de I.A. A partir de agora irei demonstrar algumas técnicas de I.A , de uma forma bem clara e básica. Uma das técnicas mais simples de programação com Inteligência Artificial é baseada em “números randômicos” (ou números aleatórios). Com essa técnica, por exemplo, podemos construir um elemento do jogo que possa se mover “aleatoriamente” pela tela, de acordo com o valor gerado.

### **12.1) Definindo movimentos aleatórios para um elemento de jogo**

Neste tópico, iremos desenvolver um jogo em que haverá um inimigo que irá se mover “aleatoriamente” pela tela, de acordo com o valor randômico gerado. Para começarmos, vamos importar o modelo de projeto para o ADT, e em seguida vamos mudar as seguintes propriedades abaixo:

Nome do projeto : IA\_MovimentosAleatorios

Constante “app\_name” : IA – Movimentos Aleatórios

Constante “title\_game\_activity” : IA – Movimentos Aleatórios

Agora dentro do projeto que acabamos de criar vamos adicionar as seguintes imagens: “inimigo\_1\_1.png”, “inimigo\_1\_2.png”, “inimigo\_2\_1.png”, “inimigo\_2\_2.png”, “inimigo\_3\_1.png”, “inimigo\_3\_2.png”, “inimigo\_4\_1.png”, “inimigo\_4\_2.png”, “player.png”, “seta\_esquerda.png”, “seta\_direita.png”, “seta\_cima.png”, “seta\_baixo.png”.



Esse jogo de demonstração que iremos desenvolver (ênfatisando o uso da I.A) será baseado no PAC-MAN. Nesse jogo haverá o nosso personagem principal e os inimigos, que se moverão pela tela de forma aleatória.

Dentro do nosso projeto iremos criar uma classe chamada **Inimigo**, que vai representar o inimigo que vai estar presente no jogo. Dentro da mesma, estará todo o código de movimento aleatório, baseado em IA usando números randômicos. Vamos agora criar a nossa classe **Inimigo**, que vai estar dentro do pacote "game.android.project" (situado dentro do diretório "src"). O processo de criação de classes já foi mostrado no tópico anterior, qualquer dúvida é só consultar.

Depois de criada a classe **Inimigo**, vamos digitar seu código conforme é mostrado a seguir:

```
package game.android.project;

import android.content.*;
import game.util.*;

public class Inimigo extends AnimationSprites {

    int contaFrame = 0;
    enum Sentido { DIREITA, ESQUERDA, CIMA, BAIXO }

    Sentido sentidoInimigo;

    public Inimigo(Context c, int x, int y, int width, int height) {
        super(c, x, y, width, height);
        Add(R.drawable.inimigo_1_1);
        Add(R.drawable.inimigo_1_2);
        Start(3, true);

        int valor = (int) Math.round(Math.random() * 3) + 1;

        switch (valor) {
            case 1: sentidoInimigo = Sentido.ESQUERDA; break;
            case 2: sentidoInimigo = Sentido.DIREITA; break;
            case 3: sentidoInimigo = Sentido.CIMA; break;
            default: sentidoInimigo = Sentido.BAIXO; break;
        }
    }

    public void Update()
    {
        contaFrame++;

        if(contaFrame == 10)
        {
            contaFrame = 0;
            int valor = (int) Math.round(Math.random() * 3) + 1;
```



```
        switch (valor) {
            case 1: sentidoInimigo = Sentido.ESQUERDA; break;
            case 2: sentidoInimigo = Sentido.DIREITA; break;
            case 3: sentidoInimigo = Sentido.CIMA; break;
            case 4: sentidoInimigo = Sentido.BAIXO; break;
        }
    }

    switch (sentidoInimigo) {
        case ESQUERDA: MoveByX(-10); break;
        case DIREITA: MoveByX(10); break;
        case CIMA: MoveByY(-10); break;
        case BAIXO: MoveByY(10); break;
    }
}

}
```

Irei explicar aqui agora o código da classe que acabamos de criar. Se observarmos a nossa classe **Inimigo**, ela é derivada da classe **AnimationSprites**, ou seja, além de visualizarmos a animação do inimigo em movimento, a classe terá instruções de I.A para movimentar o mesmo pela tela. Vamos observar os seguintes atributos declaramos abaixo:

```
int contaFrame = 0;

enum Sentido { DIREITA, ESQUERDA, CIMA, BAIXO }
Sentido sentidoInimigo;
```

A variável *contaFrame* que declaramos acima será utilizada para “contar” os frames de movimento do inimigo na tela, ou seja, quando essa variável atingir um determinado valor, será trocado “aleatoriamente” o movimento do mesmo, e em seguida, seu valor “resetado” novamente para zero “0”.

A estrutura de enumeração **Sentido** guarda os possíveis movimentos que o inimigo fará (como rótulos e em caixa alta) no jogo, e a variável *sentidoInimigo* será utilizada para definir o movimento do mesmo.

Agora vamos analisar o código dentro do construtor da classe **Inimigo**. Dentro do construtor existem as seguintes instruções:

```
Add(R.drawable.inimigo_1_1);
Add(R.drawable.inimigo_1_2);
```

Que adiciona as imagens no objeto para animação através do método **Add** (da classe **AnimationSprites**). A instrução :

```
int valor = (int) Math.round(Math.random() * 3) + 1;
```



Retorna para a variável *valor* um valor aleatório entre 1 e 4. A função **Math.random** é responsável pela geração do número aleatório, na faixa entre 0 e 1. Observe que foi necessário “multiplicar” o valor gerado por 3, pois, o valor gerado pela função é sempre um número de ponto flutuante. Como quero que o número gerado seja um número “inteiro”, foi preciso o uso da função **Math.round**, para que o valor gerado seja “arredondado” para um número inteiro. Como o número gerado pela função é um número entre 0 e 3, foi preciso somar o valor final gerado, com mais um.

De acordo com o valor gerado pela função, será definida a direção do nosso inimigo. Isso é conseguido através da instrução seguinte:

```
switch (valor) {  
    case 1: sentidoInimigo = Sentido.ESQUERDA; break;  
    case 2: sentidoInimigo = Sentido.DIREITA; break;  
    case 3: sentidoInimigo = Sentido.CIMA; break;  
    default: sentidoInimigo = Sentido.BAIXO; break;  
}
```

Agora vamos dentro do código do método **Update** para avaliarmos como funciona toda a I.A do movimento do inimigo. A primeira linha do método **Update** encontramos a seguinte instrução:

```
contaFrame++;
```

Que incrementa o valor da variável *contaFrame* a cada chamada do método. Essa variável controla o “intervalo” da troca de movimento do inimigo, que é avaliado na condição seguinte:

```
if(contaFrame == 10)
```

Toda vez que a condição acima for verdadeira, ou seja, a variável *contaFrame* atingir o valor 10, é efetuado a troca do sentido do inimigo e o valor da variável é “resetado” novamente para zero “0”. Após a condição encontramos as instruções seguintes:

```
switch (sentidoInimigo) {  
    case ESQUERDA: MoveByX(-10); break;  
    case DIREITA: MoveByX(10); break;  
    case CIMA: MoveByY(-10); break;  
    case BAIXO: MoveByY(10); break;  
}
```

Que realiza o movimento do inimigo de acordo com seu sentido.

Agora vamos carregar o arquivo “GameMain.java” para continuarmos a digitar o código do nosso jogo. Na seção de importação de pacotes, vamos digitar as seguintes instruções:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
import game.util.scene.*;
import game.util.character.Character;
```

Agora dentro da seção de declaração de atributos, vamos digitar as seguintes instruções:

```
Character packman;
Scene scene;
Image seta_direita, seta_esquerda, seta_cima, seta_baixo;

enum Movimento { DIREITA, ESQUERDA, CIMA, BAIXO, PARADO }
Movimento movimentoPersonagem = Movimento.PARADO;

enum Direcao { DIREITA, ESQUERDA }
Direcao direcaoPersonagem = Direcao.DIREITA;
```

Agora dentro do método **onSizeChanged** vamos digitar o seguinte código abaixo:

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {

    super.onSizeChanged(w, h, oldw, oldh);

    packman = new Character(context, 50, 150, 48, 48);
    packman.AddNewSpriteIdle(R.drawable.player_1);
    packman.AddNewSpriteWalking(R.drawable.player_1);
    packman.AddNewSpriteWalking(R.drawable.player_2);
    packman.Idle(3, false);

    scene = new Scene();

    scene.Add(new Inimigo(context, 400, 150, 48, 48));

    seta_esquerda = new Image(context, R.drawable.seta_esquerda, 0, h - 96,
    96, 96);

    seta_baixo = new Image(context, R.drawable.seta_baixo, 96, h - 96, 96,
    96);

    seta_cima = new Image(context, R.drawable.seta_cima, w - (96 * 2), h -
    96, 96, 96);

    seta_direita = new Image(context, R.drawable.seta_direita, w - 96, h -
    96, 96, 96);

}
```

Dentro do método **onDraw** vamos adicionar as seguintes instruções:

```
protected void onDraw(Canvas canvas) {
```





# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
super.onDraw(canvas);

Update();

canvas.drawColor(Color.BLACK);

scene.Draw(canvas);
packman.Draw(canvas);
seta_esquerda.Draw(canvas);
seta_baixo.Draw(canvas);
seta_cima.Draw(canvas);
seta_direita.Draw(canvas);

}
```

Agora dentro do método **Update** vamos adicionar as seguintes instruções:

```
public void Update() {

    switch(movimentoPersonagem)
    {
        case DIREITA : packman.MoveByX(10); break;
        case ESQUERDA : packman.MoveByX(-10); break;
        case CIMA : packman.MoveByY(-10); break;
        case BAIXO : packman.MoveByY(10); break;
    }

    for(GameElement e : scene.Elements())
    {
        ((Inimigo) e).Update();
    }
}
```

Irei comentar aqui o loop que foi inserido dentro do **Update**, e o que ele faz. O loop acima vai “percorrer” todos os elementos do objeto *scene*, e cada elemento processado dentro do loop é retornado pela variável de parâmetro *e*, como um objeto do tipo **GameElement**. Se voltarmos lá para o método **onSizeChanged**, adicionamos um elemento do tipo **Inimigo** (que é derivado da classe **AnimationSprites**, e que por sua vez é derivado da classe **GameElement**).

Como eu sei que o elemento dentro do objeto *scene* é um objeto do tipo **Inimigo**, e ele está armazenado na variável *e*, que é do tipo **GameElement**, precisamos “converter” (realizar um “cast”) o objeto do tipo **GameElement** para o tipo **Inimigo**, para executarmos o método **Update**, como você pode ver na instrução abaixo:

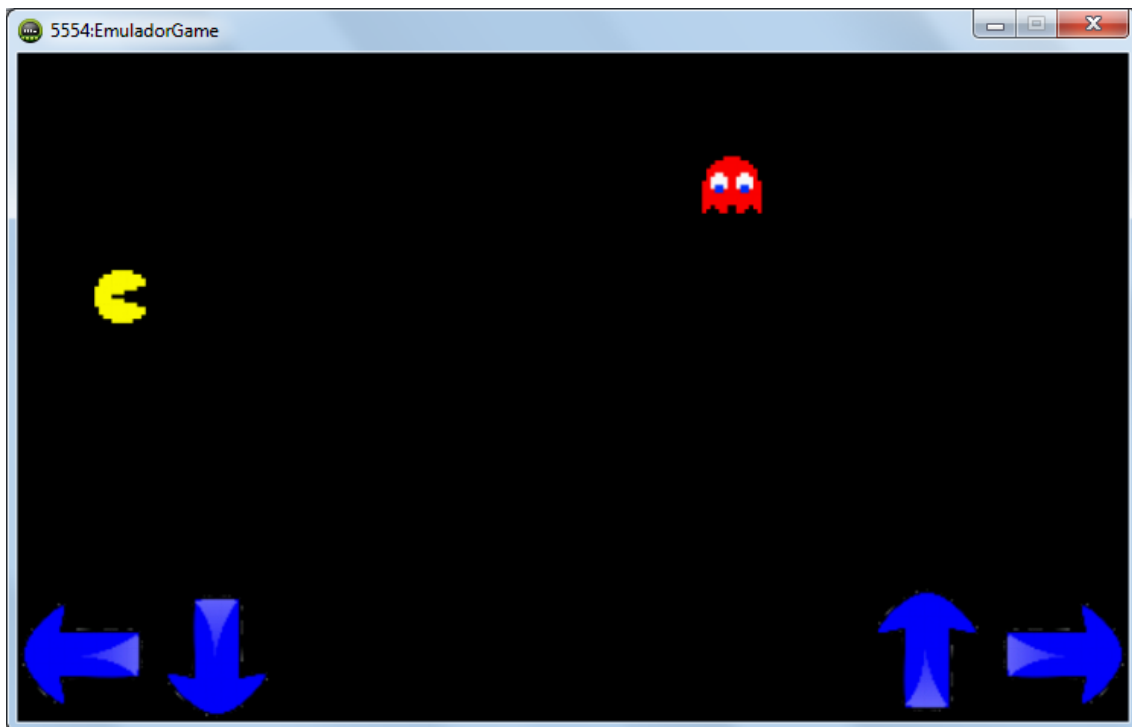
```
((Inimigo) e).Update();
```



Agora dentro do método **onTouchEvent** vamos adicionar as seguintes instruções:

```
public boolean onTouchEvent(MotionEvent event) {  
  
    if(event.getAction() == MotionEvent.ACTION_DOWN) {  
  
        if(seta_esquerda.IsTouch(event.getX(), event.getY()))  
        {  
            movimentoPersonagem = Movimento.ESQUERDA;  
            direcaoPersonagem = Direcao.ESQUERDA;  
            packman.WalkingToLeft(3, true);  
        }  
        else if(seta_direita.IsTouch(event.getX(), event.getY()))  
        {  
            movimentoPersonagem = Movimento.DIREITA;  
            direcaoPersonagem = Direcao.DIREITA;  
            packman.WalkingToRight(3, true);  
        }  
        else if(seta_cima.IsTouch(event.getX(), event.getY()))  
        {  
            movimentoPersonagem = Movimento.CIMA;  
            if(direcaoPersonagem == Direcao.DIREITA)  
                packman.WalkingToRight(3, true);  
            else  
                packman.WalkingToLeft(3, true);  
        }  
        else if(seta_baixo.IsTouch(event.getX(), event.getY()))  
        {  
            movimentoPersonagem = Movimento.BAIXO;  
            if(direcaoPersonagem == Direcao.DIREITA)  
                packman.WalkingToRight(3, true);  
            else  
                packman.WalkingToLeft(3, true);  
        }  
    }  
    else if(event.getAction() == MotionEvent.ACTION_UP)  
    {  
        movimentoPersonagem = Movimento.PARADO;  
        packman.Idle(3, false);  
    }  
  
    return true;  
}
```

Vamos executar o nosso jogo para conferir o resultado, como demonstra a figura seguinte:



**Jogo em Execução – Inimigo se movimentando aleatoriamente**

Se observarmos o jogo acima, o nosso inimigo quando ele se move aleatoriamente, as vezes e sai para “fora” da tela, e o mesmo possivelmente não retorna mais. A nossa I.A funciona, porém, ela precisa ser otimizada, de modo que o inimigo não saia da tela.

### 12.1.1) Melhorando o código : Não deixar ele sair da tela

Agora vamos melhorar o código da nossa classe **Inimigo** de forma que quando o nosso inimigo estiver se movimentando pela tela, o mesmo não saia do jogo. Para isso, vamos abrir o código da classe **Inimigo** e no método **Update** , vamos substituir o código mostrado abaixo:

```
:
switch (sentidoInimigo) {
    case ESQUERDA: MoveByX(-10); break;
    case DIREITA: MoveByX(10); break;
    case CIMA: MoveByY(-10); break;
    case BAIXO: MoveByY(10); break;
}
```

Por esse novo código:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
switch (sentidoInimigo) {
    case ESQUERDA: {
        MoveByX(-10);

        if(GetX() < 0)
        {
            sentidoInimigo = Sentido.DIREITA;
            contaFrame = 0;
        }

    }
    break;

    case DIREITA:{

        MoveByX(10);
        if(GetX() > (800 - GetWidth()))
        {
            sentidoInimigo = Sentido.ESQUERDA;
            contaFrame = 0;
        }
    }
    break;

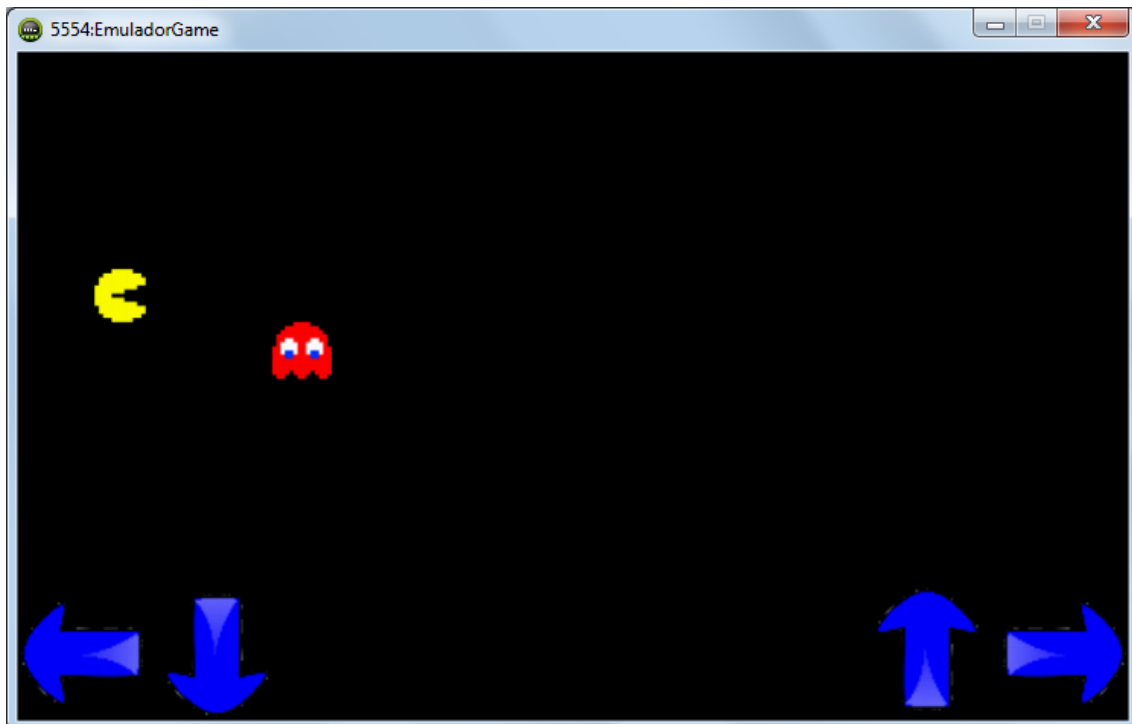
    case CIMA:
    {
        MoveByY(-10);
        if(GetY() < 0)
        {
            sentidoInimigo = Sentido.BAIXO;
            contaFrame = 0;
        }
    }
    break;

    case BAIXO: {

        MoveByY(10);
        if(GetY() > (480 - GetHeight()))
        {
            sentidoInimigo = Sentido.CIMA;
            contaFrame = 0;
        }
    }
    break;

}
```

Toda vez que o nosso inimigo, a partir de agora, estiver se movimentando pela tela, o mesmo irá avaliar se ele está saindo da tela, se essa situação acontecer, ele irá tomar o sentido contrário da direção em que ele estava saindo da tela.

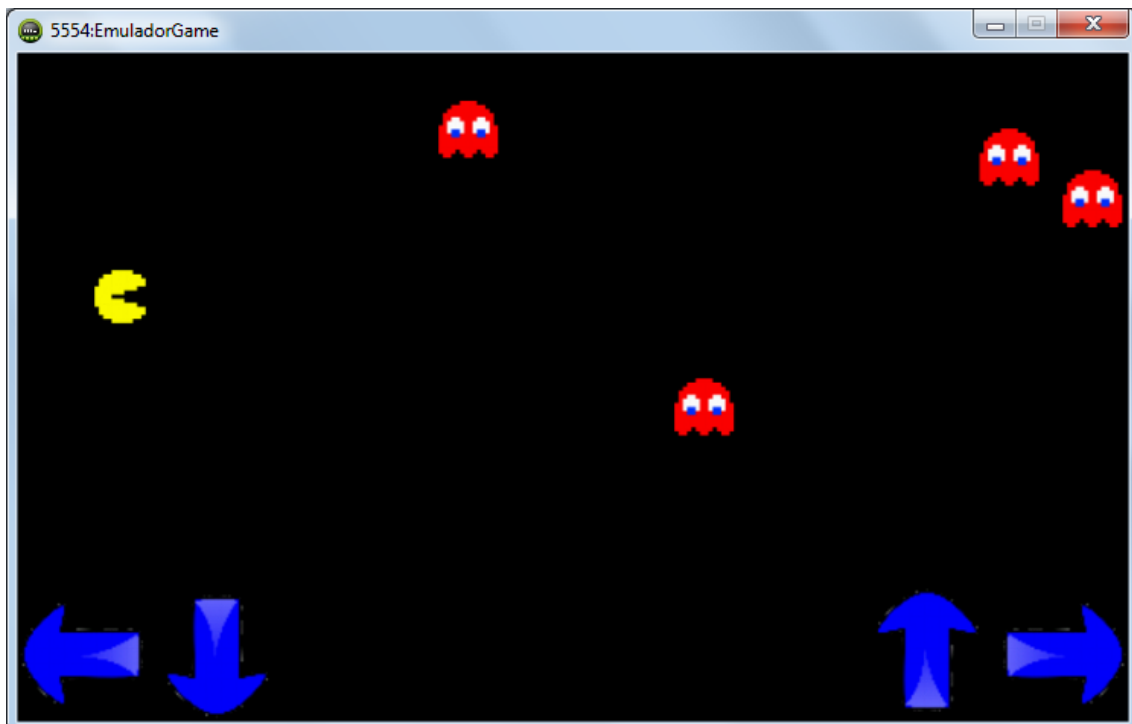


**Jogo em Execução – Inimigo se movimentando aleatoriamente sem sair da tela**

Vamos voltar para o código do método **onSizeChanged** , para adicionarmos mais alguns inimigos, como demonstra as instruções **em negrito**:

```
scene.Add(new Inimigo(context, 400, 150, 48, 48));  
  
scene.Add(new Inimigo(context, 500, 200, 48, 48));  
scene.Add(new Inimigo(context, 600, 250, 48, 48));  
scene.Add(new Inimigo(context, 300, 200, 48, 48));
```

Vamos executar o jogo agora para ver o resultado da alteração desse código:



Jogo em Execução - Múltiplos elementos (inimigos) na tela

### 12.1.2) Melhorando o código : Gerando inimigos de cores aleatórias

Se você observar o exemplo acima, todos os inimigos são da mesma cor (vermelho). Não seria mais interessante que cada um deles fosse de cor diferente ? Pois bem, na própria classe **Inimigo** podemos realizar essa tarefa. No construtor da classe **Inimigo** vamos substituir o seguinte código abaixo:

```
Add(R.drawable.inimigo_1_1);  
Add(R.drawable.inimigo_1_2);
```

Pelo seguinte novo código:

```
int cor = (int) Math.round(Math.random() * 3) + 1;  
if(cor == 1)  
{  
    Add(R.drawable.inimigo_1_1);  
    Add(R.drawable.inimigo_1_2);  
}  
else if(cor == 2)  
{  
    Add(R.drawable.inimigo_2_1);  
    Add(R.drawable.inimigo_2_2);  
}  
else if(cor == 3)  
{  
    Add(R.drawable.inimigo_3_1);  
}
```





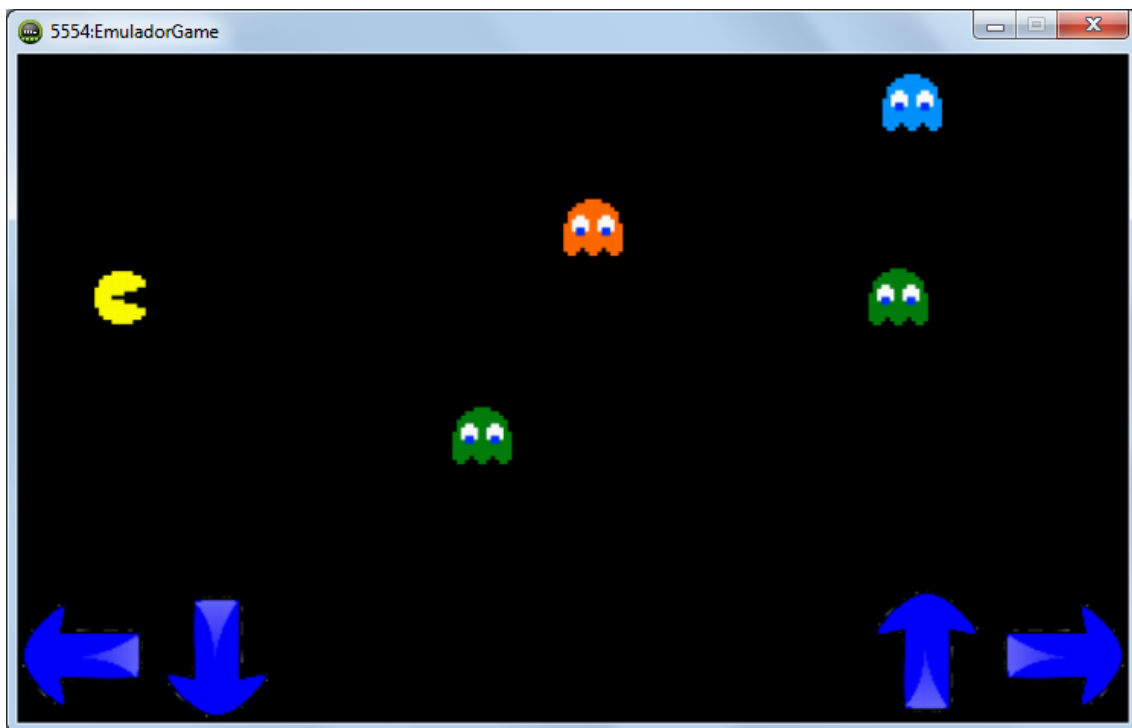
# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

```
        Add(R.drawable.inimigo_3_2);
    }
    else if(cor == 4)
    {
        Add(R.drawable.inimigo_4_1);
        Add(R.drawable.inimigo_4_2);
    }
}
```

Salve as alterações feitas na classe **Inimigo** e coloque novamente o jogo para executar. O resultado você confere agora na figura seguinte:



**Jogo em Execução - Inimigos de cores diferentes**

Agora vamos finalizar o nosso jogo. Toda vez em que nosso personagem estiver em movimento e ele tocar um dos seus inimigos, o mesmo será eliminado da tela. Para isso, vamos no código do método **Update** para digitarmos o seguinte código **em negrito**:

```
:
for(GameElement e : scene.Elements())
{
    ((Inimigo) e).Update();
}

if(movimentoPersonagem != Movimento.PARADO)
{
```



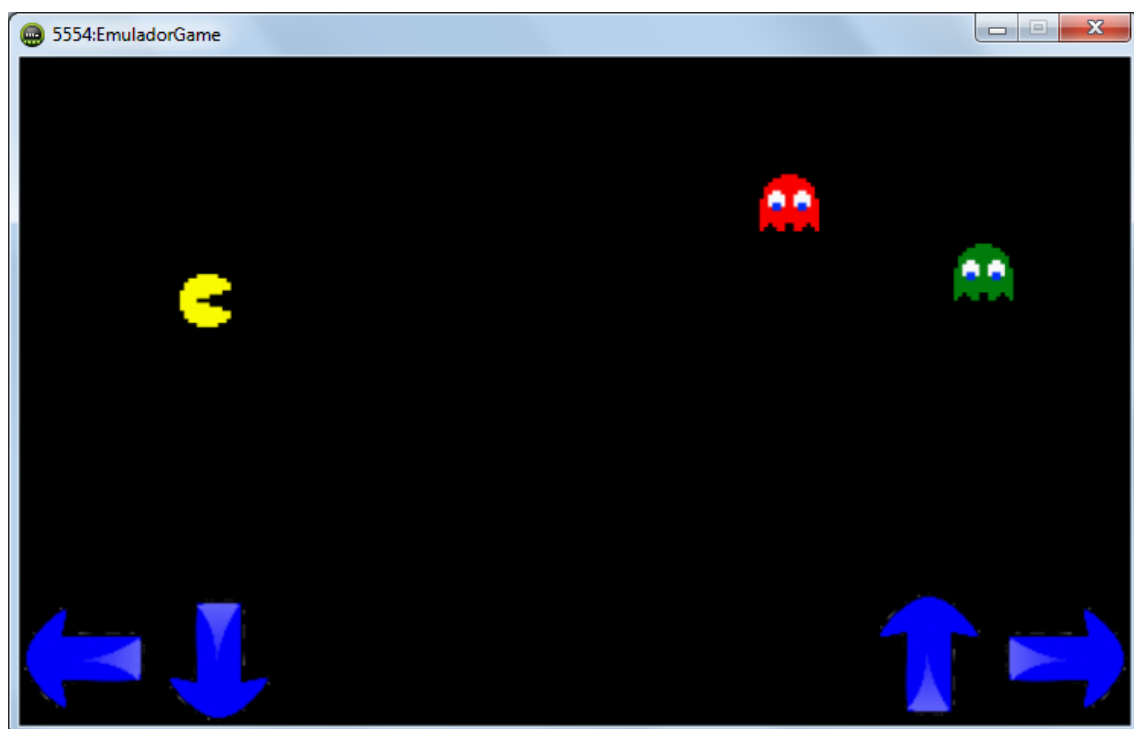
# Apostila de Android

## Programando Passo a Passo

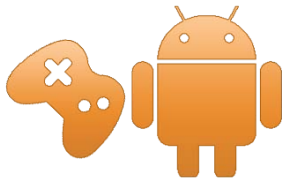
### Desenvolvimento de Jogos (Edição Completa)

```
for(GameElement e : scene.Elements())  
{  
    if(Collision.Check(packman, e))  
    {  
        scene.Remove(e);  
        break;  
    }  
}
```

Execute o jogo novamente e confira os resultados, como demonstra a figura seguinte:



**Jogo em Execução - Inimigos sendo eliminados pelo personagem**



## Capítulo 13 Músicas e sons no jogo

Nenhum jogo, por qualquer que seja seu gênero, passa emoção para o jogador sem uma boa música e sons de efeito. Em todo jogo, na minha opinião, não pode faltar esses itens importantes. Neste capítulo irei mostrar como trabalhar com músicas e sons de efeito nos jogos feitos aqui no Android.

### 13.1) A classe MediaPlayer

A classe **MediaPlayer**, disponível na plataforma Android (situado no pacote “android.media”), é responsável por reproduzir músicas que desejamos reproduzir em nossos aplicativos (jogos). Aqui irei trabalhar com músicas de extensão “.MP3” para esse propósito. Vamos conhecer algumas de suas funções:

Método	Descrição
<code>MediaPlayer.create(Context c, int resid)</code>	Método responsável por carregar a música a ser executada.
<code>public void start()</code>	Executa a música carregada.
<code>public void stop()</code>	Encerra a execução da música
<code>public void setLooping(boolean looping)</code>	Método que define se a música em execução estará em loop (caso <b>true</b> ) ou não (caso <b>false</b> )
<code>public boolean isLooping()</code>	Método que retorna verdadeiro se a música em execução está em loop.

### 13.2) A classe SoundPool

Essa classe (também situado dentro do pacote “android.media”) é responsável por reproduzir os sons de efeitos que utilizamos nos jogos. “Os arquivos que utilizaremos para reproduzir os sons de efeito serão arquivos de áudio de extensão “.WAV”. Essa classe na verdade funciona como um reprodutor e



gerenciador de sons que serão reproduzidos na aplicativos (jogo). Vejamos agora alguns de seus métodos:

Método	Descrição
<code>SoundPool((int maxStreams, int streamType, int srcQuality)</code>	Método responsável por carregar o objeto que gerenciará os sons de efeito.
<code>public int load(Context context, int resId, int priority)</code>	Esse método será responsável por carregar o som de efeito a ser reproduzido, e o mesmo será representado por uma ID, que é gerada e retornada pela função.
<code>public int play(int soundID, float leftVolume, float rightVolume, int priority, int loop, float rate)</code>	Reproduz o som de efeito, utilizando como referência a sua ID.

### 13.3) Desenvolvendo o jogo

Antes de começarmos, vamos importar o modelo de projeto para o ADT e em seguida vamos realizar as seguintes alterações:

Nome do projeto : MusicaSonsDeEfeito

Constante “app\_name” : Música e Sons de Efeito

Constante “title\_game\_activity” : Música e Sons de Efeito

Depois de configurar os parâmetros acima, vamos adicionar as seguintes imagens: “anel\_1.png”, “anel\_2.png”, “anel\_3.png”, “anel\_4.png”, “bloco.png”, “botao\_pulo.png”, “ceu\_azul.png”, “chao.png”, “seta\_direita.png”, “seta\_esquerda.png”, “sonic\_correndo\_1.png”, “sonic\_correndo\_2.png”, “sonic\_correndo\_3.png”, “sonic\_correndo\_4.png”, “sonic\_correndo\_5.png”, “sonic\_parado.png”, “sonic\_pulando\_1.png”, “sonic\_pulando\_2.png”, “sonic\_pulando\_3.png”, “sonic\_pulando\_4.png”.

Observe que ainda não adicionamos as músicas e sons de efeitos que utilizaremos em nosso jogo, isso porque que no Android não podemos adicionar qualquer tipo de arquivo em qualquer tipo de pasta. A estrutura de diretórios do Android segue um conjunto de regras para os tipos de arquivos a serem armazenados dentro de cada pasta.

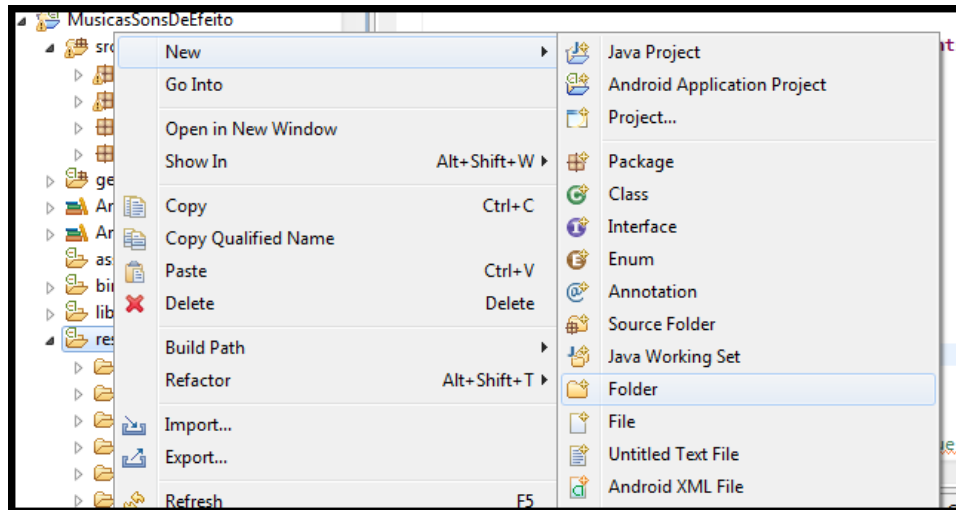


# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

Dentro de um projeto Android existe um diretório, que por padrão não está criado dentro do nosso projeto, que serve para armazenar arquivos de música e sons de efeitos. O nome desse diretório que armazena esses tipos de arquivo se chama “raw”, logo, vamos criar esse diretório dentro da pasta “res” presente em nosso projeto, simples clicando com o botão direito sobre essa pasta, e em seguida selecionando “New” / “Folder”, como demonstra a figura seguinte:



**Criando uma nova pasta**

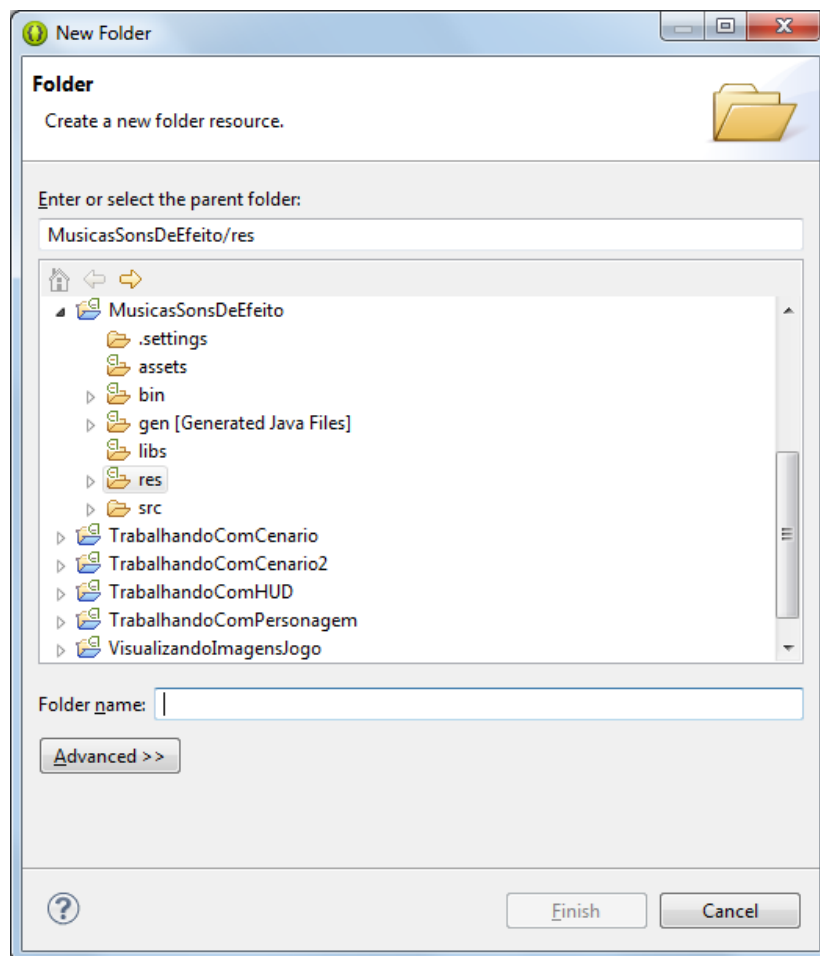
Feito o que se foi mostrado, será exibida a seguinte caixa de diálogo:



# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)



**Caixa de diálogo – New Folder**

No campo “Folder Name” vamos digitar o nome da pasta que iremos criar, que é a pasta “raw” (sem aspas , é claro). Depois de digitar o nome da pasta, vamos clicar no botão “Finish” para que a pasta seja gerada, como você pode conferir na figura seguinte:

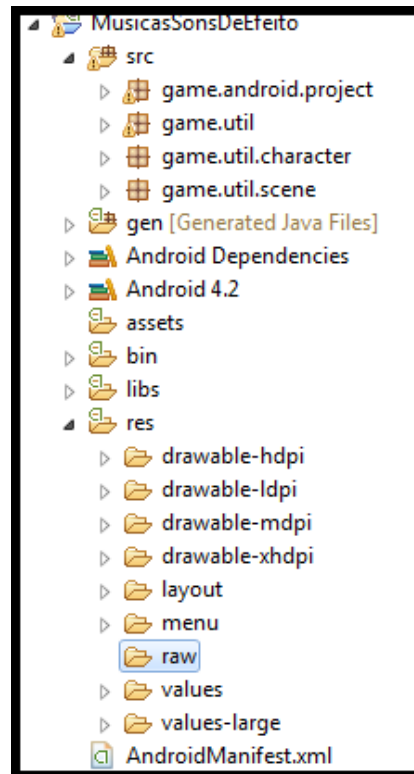




# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



Pasta “raw” criada

Agora dentro da pasta que acabamos de criar, vamos adicionar os seguintes arquivos : “musica\_fase.mp3”, “som\_anel.wav”, “som\_pulo.wav”.

O jogo de demonstração que iremos fazer para enfatizar este capítulo, será baseado no jogo do “Sonic The Hedgehog” (da plataforma Master System).

Para começarmos o desenvolvimento do jogo, vamos criar uma classe chamada **Anel** dentro do pacote “game.android.project”. Depois disso, vamos digitar o código da classe abaixo:

```
package game.android.project;

import android.content.*;
import game.util.*;

public class Anel extends AnimationSprites {

    public Anel(Context c, int x, int y, int width, int height) {
        super(c, x, y, width, height);
        Add(R.drawable.anel_1);
        Add(R.drawable.anel_2);
        Add(R.drawable.anel_3);
        Add(R.drawable.anel_4);
        Start(2, true);
    }
}
```



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

Depois de escrever o código da classe acima, salve. Agora vamos no arquivo “GameMain.java” para escrevermos o código do nosso jogo. Na seção de importação de pacotes, vamos declarar as seguintes instruções abaixo:

```
import game.util.scene.*;
import game.util.character.Character;
```

Agora segue abaixo, o código completo da nossa classe **GameMain** :

```
public class GameMain extends SurfaceView implements Runnable {

    GameState gameState;

    Thread thread = null;
    SurfaceHolder surfaceHolder;
    volatile boolean running = false;
    Context context;

    Character sonic;
    Scene scene;

    Image ceu_azul;

    Image chao, chao2;

    Image seta_direita, seta_esquerda, botao_pulo;

    enum Sentido { DIREITA, ESQUERDA, PARADO }
    Sentido sentidoSonic = Sentido.PARADO;

    int largura_tela;

    public GameMain(Context context) {

        super(context);
        this.context = context;
        surfaceHolder = getHolder();

        setFocusable(true);
    }

    @Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh) {

        super.onSizeChanged(w, h, oldw, oldh);

        largura_tela = w;
    }
}
```



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
scene = new Scene();
ceu_azul = new Image(context, R.drawable.ceu_azul, 0, 0, w, h);
chao = new Image(context, R.drawable.chao, 0, h - 90, w, 90);
chao.SetTag("chao");
chao2 = new Image(context, R.drawable.chao, w, h - 90, w, 90);
chao2.SetTag("chao");

scene.Add(ceu_azul);
scene.Add(chao);
scene.Add(chao2);

sonic = new Character(context, (w / 2) - (91 / 2) , chao.GetY() -
98, 91, 98);
sonic.AddNewSpriteIdle(R.drawable.sonic_parado);

sonic.AddNewSpriteRunning(R.drawable.sonic_correndo_1);
sonic.AddNewSpriteRunning(R.drawable.sonic_correndo_2);
sonic.AddNewSpriteRunning(R.drawable.sonic_correndo_3);
sonic.AddNewSpriteRunning(R.drawable.sonic_correndo_4);
sonic.AddNewSpriteRunning(R.drawable.sonic_correndo_5);

sonic.AddNewSpriteJumping(R.drawable.sonic_pulando_1);
sonic.AddNewSpriteJumping(R.drawable.sonic_pulando_2);
sonic.AddNewSpriteJumping(R.drawable.sonic_pulando_3);
sonic.AddNewSpriteJumping(R.drawable.sonic_pulando_4);

sonic.AddCollisionElementOfFallByTag("chao");

sonic.Idle(3, false);

scene.Add(sonic);

scene.Add(new Anel(context, 600, chao.GetY() - 80, 64, 64));
scene.Add(new Anel(context, 670, chao.GetY() - 80, 64, 64));
scene.Add(new Anel(context, 740, chao.GetY() - 80, 64, 64));
scene.Add(new Anel(context, 810, chao.GetY() - 80, 64, 64));

seta_esquerda = new Image(context, R.drawable.seta_esquerda, 0, h -
96, 96, 96);
seta_direita= new Image(context, R.drawable.seta_direita, w - 96, h
- 96, 96, 96);

botao_pulo = new Image(context, R.drawable.botao_pulo, w - (96 * 2),
h - 96, 96, 96);

}
```

```
@Override
protected void onDraw(Canvas canvas) {

    super.onDraw(canvas);

    Update();
```



# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

```
canvas.drawColor(Color.BLACK);
scene.Draw(canvas);
seta_esquerda.Draw(canvas);
seta_direita.Draw(canvas);
botao_pulo.Draw(canvas);
```

```
}
```

```
public void Update() {
```

```
    if(sentidoSonic == Sentido.DIREITA)
    {
        chao.MoveByX(-15);
        chao2.MoveByX(-15);

        if(chao.GetX() < -(chao.GetWidth()))
            chao.SetX(chao2.GetX() + chao2.GetWidth());

        if(chao2.GetX() < -(chao2.GetWidth()))
            chao2.SetX(chao.GetX() + chao.GetWidth());

        for(int x = 0; x < scene.GetCount() ; x++)
        {
            if(scene.Get(x) instanceof Anel)
            {
                scene.Get(x).MoveByX(-15);
                if(Collision.Check(sonic, scene.Get(x)))
                {
                    scene.Remove(scene.Get(x));
                    x--;
                }
            }
        }
    }
    else if(sentidoSonic == Sentido.ESQUERDA)
    {
        chao.MoveByX(15);
        chao2.MoveByX(15);

        if(chao.GetX() > largura_tela)
            chao.SetX(chao2.GetX() - chao2.GetWidth());

        if(chao2.GetX() > largura_tela)
            chao2.SetX(chao.GetX() - chao.GetWidth());

        for(int x = 0; x < scene.GetCount() ; x++)
        {
            if(scene.Get(x) instanceof Anel)
            {
                scene.Get(x).MoveByX(15);
                if(Collision.Check(sonic, scene.Get(x)))
                {
                    scene.Remove(scene.Get(x));
                    x--;
                }
            }
        }
    }
}
```



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
        }
    }
}

sonic.Update(scene);

}

:

@Override
public boolean onTouchEvent(MotionEvent event) {

    if(event.getAction() == MotionEvent.ACTION_DOWN)
    {
        if(seta_esquerda.IsTouch(event.getX(), event.getY()))
        {
            if((sentidoSonic == Sentido.PARADO) || (sentidoSonic ==
                Sentido.DIREITA))
            {
                sentidoSonic = Sentido.ESQUERDA;
                sonic.RunningToLeft(2, true);
            }
            else {

                sentidoSonic = Sentido.PARADO;
                sonic.Idle(3, false);

            }
        }
        else if(seta_direita.IsTouch(event.getX(), event.getY()))
        {
            if((sentidoSonic == Sentido.PARADO) || (sentidoSonic ==
                Sentido.ESQUERDA))
            {
                sentidoSonic = Sentido.DIREITA;
                sonic.RunningToRight(2, true);
            }
            else {

                sentidoSonic = Sentido.PARADO;
                sonic.Idle(3, false);

            }
        }
        else if(botao_pulo.IsTouch(event.getX(), event.getY()))
        {
            sonic.Jump(2, true);
        }
    }

    return true;
}
}
```

Executando o nosso jogo, nós temos o seguinte resultado.



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



**Jogo em Execução – Sem música e som de efeito**

Se você não percebeu, o tempo para a compilação do projeto demorou um pouco mais do que devia, isso porque nós adicionamos os arquivos de música e sons de efeito que influenciou no tamanho do projeto. Por isso, o tempo para a geração do executável (geração do arquivo “.apk”) foi um pouco mais demorado.

O código do nosso jogo funciona perfeitamente, agora, precisamos aplicar a música e os sons de efeito. Como havia falado, todas as classes que iremos utilizar fazem parte do pacote “android.media”, logo, precisamos importar esse pacote para o nosso código, conforme é conferido em seguida:

```
import android.media.*;
```

Agora na seção de declaração de atributos, vamos escrever o seguinte código abaixo:

```
MediaPlayer mp;  
  
SoundPool sp;  
  
int ID_SOM_PULO;  
int ID_SOM_ANEL;
```



Agora dentro do método **GameMain**, vamos adicionar as seguintes instruções seguintes:

```
public GameMain(Context context) {
    super(context);
    this.context = context;
    surfaceHolder = getHolder();

    mp = MediaPlayer.create(context, R.raw.musica_fase);
    mp.start();
    mp.setLooping(true);

    sp = new SoundPool(2, AudioManager.STREAM_MUSIC, 0);

    ID_SOM_ANEL = sp.load(context, R.raw.som_anel, 0);
    ID_SOM_PULO = sp.load(context, R.raw.som_pulo, 0);

    setFocusable(true);
}
```

Irei comentar aqui algumas linhas de código. A linha:

```
mp = MediaPlayer.create(context, R.raw.musica_fase);
```

Cria a instância do objeto *mp*, do tipo **MediaPlayer** e carrega a música a ser reproduzida, através do método **create** (funciona como se fosse um método construtor da classe). Na linha seguinte:

```
mp.start();
```

Inicia a reprodução da música do jogo. Já a próxima linha:

```
mp.setLooping(true);
```

Coloca a música no estado de “loop”, para que ela possa repetir sempre quando a mesma chegar no final, através do método **setLooping**. A instrução seguinte:

```
sp = new SoundPool(2, AudioManager.STREAM_MUSIC, 0);
```

Cria a instância do objeto *sp*, do tipo **SoundPool**. O construtor possui os seguintes parâmetros : O primeiro (com o valor 2), corresponde ao número de sons de efeitos que serão reproduzidos (aqui no jogo utilizaremos o som de pulo e o som do anel sendo obtido, logo, usaremos 2 sons de efeito) ; O segundo parâmetro corresponde ao tipo de fluxo (Stream Type) que iremos utilizar, para essa situação usamos o argumento





“AudioManager.STREAM\_MUSIC” ; O terceiro parâmetro corresponde a qualidade, que por padrão, deixamos com o valor “0”.

A instruções seguintes:

```
ID_SOM_ANEL = sp.load(context, R.raw.som_anel, 0);  
ID_SOM_PULO = sp.load(context, R.raw.som_pulo, 0);
```

Carrega os sons de efeito que serão reproduzidos, e cada som será representado pela sua ID (cada ID uma variável do tipo **int**).

Agora dentro da classe **GameMain** vamos adicionar o seguinte método :

```
public void executarSom(int ID) {  
    sp.play(ID, 1, 1, 1, 0, 1f);  
}
```

O método **executarSom** criado acima, será responsável por executar o som de efeito, através de sua ID, passada como parâmetro no método.

Agora dentro do método **Update** vamos adicionar as seguintes linhas de código **em negrito**:

```
public void Update() {  
  
    if(sentidoSonic == Sentido.DIREITA)  
    {  
        :  
        if(Collision.Check(sonic, scene.Get(x)))  
        {  
            scene.Remove(scene.Get(x));  
            executarSom(ID_SOM_ANEL);  
            x--;  
        }  
        :  
    }  
    else if(sentidoSonic == Sentido.ESQUERDA)  
    {  
        :  
        if(Collision.Check(sonic, scene.Get(x)))  
        {  
            scene.Remove(scene.Get(x));  
            executarSom(ID_SOM_ANEL);  
            x--;  
        }  
    }  
}
```



```
        :  
    }  
}
```

Agora dentro do método **onTouchEvent** vamos adicionar a seguinte instrução **em negrito**.

```
public boolean onTouchEvent(MotionEvent event) {  
    if(event.getAction() == MotionEvent.ACTION_DOWN)  
    {  
        :  
        else if(botao_pulo.IsTouch(event.getX(), event.getY()))  
        {  
            sonic.Jump(2, true);  
            executarSom(ID_SOM_PULO);  
        }  
    }  
    :  
}
```

Agora para finalizar, dentro do método **onExitGame**, vamos adicionar a seguinte instrução :

```
public void onExitGame() {  
    mp.stop();  
}
```

Esse código adicionado acima é importante, pois, após a aplicação ser encerrada, ele interrompe a música em jogo. Se essa instrução não existisse, mesmo após a execução do jogo, a música continuaria a executar.

Depois de escrever os códigos solicitados acima, vamos executar o nosso jogo para conferir os resultados.



## Capítulo 14 Trabalhando com mais de uma tela no jogo

A te agora os exemplos que demonstrei , tinham somente uma única tela, que era a tela do próprio jogo em si. Mas se em meu jogo eu quiser trabalhar com mais de uma tela, como por exemplo, uma tela de abertura, uma tela de créditos, uma tela de menu e etc. Neste capítulo vamos aprender como criar e gerenciar várias telas que podemos inserir em nossos jogos.

Antes de começarmos, vamos adicionar o nosso modelo de projeto no ADT, e em seguida vamos alterar as seguintes propriedades:

Nome do projeto : GerenciandoTelas

Constante "app\_name" : Gerenciando Telas

Constante "title\_game\_activity" :Gerenciando Telas

Agora dentro do projeto vamos adicionar as seguintes imagens :  
"botao\_iniciar.png", "botao\_pulo.png", "ceu.png", "chao.png",  
"mario\_correndo\_1.png", "mario\_correndo\_2.png", "mario\_parado.png",  
"mario\_pulando.png", "seta\_direita.png", "seta\_esquerda.png",  
"tela\_abertura\_mario\_bros.jpg", "tela\_game\_over.png".

O nosso exemplo irá se basear no jogo do famoso "Super Mario Bros". Como iremos trabalhar com mais de uma tela no jogo, precisamos arrumar uma forma de estruturar e gerenciar essas telas. Para cada tela do jogo, iremos criar uma classe que irá representá-la.

No exemplo que iremos desenvolver agora, o nosso jogo terá as seguintes telas : Tela de Menu, Tela Da Fase 1 e Tela de Game Over. Para cada uma dessas telas, haverá uma classe que irá representá-la. Vamos começar primeiramente criando a tela de menu, logo, iremos criar uma classe chamada **TelaMenu**. Vamos criar a classe dentro do pacote "game.android.project", e depois de criado a nossa classe, vamos escrever o seu código como você pode conferir em seguida:



```
package game.android.project;

import android.content.Context;
import android.graphics.Canvas;
import android.view.MotionEvent;
import game.util.*;

public class TelaMenu {

    public TelaMenu(Context context)
    {

    }

    public void onSizeChanged(int w, int h, int oldw, int oldh)
    {

    }

    public void Update()
    {

    }

    public void Draw(Canvas canvas)
    {

    }

    public void onTouchEvent(MotionEvent event)
    {

    }

}
```

Se observarmos a classe criada acima, ela possui os seguintes métodos:

**TelaMenu:** Corresponde ao construtor da classe (equivale ao construtor da classe principal **GameMain**), onde nele iremos adicionar as instruções que serão inicialmente carregadas.

**onSizeChanged:** Esse método será disparado toda vez que a resolução/orientação do dispositivo sofrer alguma alteração (equivalente ao método **onSizeChanged** da classe principal).

**Update:** Será responsável por “processar” a lógica do jogo (equivalente ao método **Update** da classe principal).



**Draw:** Será responsável por desenhar os elementos do jogo (equivalente ao método **onDraw** da classe principal).

**onTouchEvent:** Será disparado quando algum evento de toque ocorrer sobre a superfície da tela do dispositivo (equivalente ao método **onTouchEvent** da classe principal).

Todas as classes que irão representar as telas do jogo, serão compostas sempre por esses métodos. Será um padrão de estrutura que iremos seguir.

Agora vamos criar uma classe chamada **TelaFase** (dentro do pacote “game.android.project”), que irá representar a tela onde o jogo irá acontecer. Depois de criada a classe, vamos adicionar o seguinte código dela em seguida:

```
package game.android.project;

import android.content.Context;
import android.graphics.Canvas;
import android.view.MotionEvent;

import game.util.*;
import game.util.scene.*;
import game.util.character.Character;

public class TelaFase {

    public TelaFase(Context context)
    {

    }

    public void onSizeChanged(int w, int h, int oldw, int oldh)
    {

    }

    public void Update()
    {

    }

    public void Draw(Canvas canvas)
    {

    }

    public void onTouchEvent(MotionEvent event)
    {

    }

}
```



```
}
```

Agora iremos criar uma classe chamada **TelaGameOver** (dentro do pacote “game.android.project”), que irá representar a tela de “game over” do nosso jogo. O código da classe você confere em seguida:

```
package game.android.project;

import android.content.Context;
import android.graphics.Canvas;
import android.view.MotionEvent;

import game.util.*;

public class TelaGameOver {

    public TelaGameOver(Context context)
    {

    }

    public void onSizeChanged(int w, int h, int oldw, int oldh)
    {

    }

    public void Update()
    {

    }

    public void Draw(Canvas canvas)
    {

    }

    public void onTouchEvent(MotionEvent event)
    {

    }

}
```

Agora , para finalizar, iremos criar uma classe que será responsável por gerenciar todas as classes criadas anteriormente, que irá se chamar **TelasDoJogo** (dentro do pacote “game.android.project”). Depois de criar a classe, vamos adicionar o seu código, conforme você confere em seguida:

```
public class TelasDoJogo {

    public enum TelaJogo { TELA_MENU, TELA_FASE, TELA_GAME_OVER }
    public static TelaJogo tela;
```



```
}
```

Agora vamos no arquivo “GameMain.java” para escrevermos e estruturarmos o nosso jogo de forma que ele trabalhe com o gerenciamento de telas , através das classes que acabamos de criar.

Na seção de declaração de atributos, vamos adicionar as seguintes linhas de comando abaixo:

```
TelaMenu tela_menu;  
TelaFase tela_fase;  
TelaGameOver tela_game_over;
```

Cada tela do jogo será uma instância de classe (objeto), que será executada de acordo com o valor da variável de controle *tela*, definida na classe **TelasDoJogo** que indicará qual tela será “processada” no momento.

Agora dentro do método **GameMain** vamos adicionar as seguintes instruções abaixo:

```
public GameMain(Context context) {  
  
    super(context);  
    this.context = context;  
    surfaceHolder = getHolder();  
  
    tela_menu = new TelaMenu(context);  
    tela_fase = new TelaFase(context);  
    tela_game_over = new TelaGameOver(context);  
  
    TelasDoJogo.tela = TelaJogo.TELA_MENU;  
  
    setFocusable(true);  
}
```

Irei comentar aqui algumas linhas de código. As instruções:

```
tela_menu = new TelaMenu(context);  
tela_fase = new TelaFase(context);  
tela_game_over = new TelaGameOver(context);
```

Carrega todas as telas do jogo. A instrução :

```
TelasDoJogo.tela = TelaJogo.TELA_MENU;
```

Atribui a variável *tela* o valor “TelaJogo.TELA\_MENU”, que indica que a tela que será processada será a tela de menu.





Dentro do método **onSizeChanged**, vamos escrever o seguinte código abaixo:

```
protected void onSizeChanged(int w, int h, int oldw, int oldh) {  
  
    super.onSizeChanged(w, h, oldw, oldh);  
    tela_menu.onSizeChanged(w, h, oldw, oldh);  
    tela_fase.onSizeChanged(w, h, oldw, oldh);  
    tela_game_over.onSizeChanged(w, h, oldw, oldh);  
}
```

Agora dentro do método **onDraw** vamos escrever as seguintes linhas de comando abaixo:

```
protected void onDraw(Canvas canvas) {  
  
    super.onDraw(canvas);  
  
    Update();  
  
    canvas.drawColor(Color.BLACK);  
  
    if(TelasDoJogo.tela == TelaJogo.TELA_MENU)  
        tela_menu.Draw(canvas);  
    else if(TelasDoJogo.tela == TelaJogo.TELA_FASE)  
        tela_fase.Draw(canvas);  
    else if(TelasDoJogo.tela == TelaJogo.TELA_GAME_OVER)  
        tela_game_over.Draw(canvas);  
  
}
```

Se observarmos a estrutura adicionada no método acima, ela irá executar o processamento do desenho das telas, de acordo com o valor da variável *tela*.

Agora dentro do método **Update** vamos adicionar as seguintes comandos:

```
public void Update() {  
  
    if(TelasDoJogo.tela == TelaJogo.TELA_MENU)  
        tela_menu.Update();  
    else if(TelasDoJogo.tela == TelaJogo.TELA_FASE)  
        tela_fase.Update();  
    else if(TelasDoJogo.tela == TelaJogo.TELA_GAME_OVER)  
        tela_game_over.Update();  
  
}
```

Agora dentro do método **onTouchEvent** vamos adicionar as seguintes instruções:

```
public boolean onTouchEvent(MotionEvent event) {
```



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
        if(TelasDoJogo.tela == TelaJogo.TELA_MENU)
            tela_menu.onTouchEvent(event);
        else if(TelasDoJogo.tela == TelaJogo.TELA_FASE)
            tela_fase.onTouchEvent(event);
        else if(TelasDoJogo.tela == TelaJogo.TELA_GAME_OVER)
            tela_game_over.onTouchEvent(event);

        return true;
    }
```

Bom, a estrutura principal do nosso jogo já está concluída. Agora, precisamos definir o código de cada tela do jogo. Vamos começar definindo o código da tela de menu, que é representado pela classe **TelaMenu**. Na classe **TelaMenu**, dentro da seção de declaração de atributos, vamos escrever as seguintes instruções seguintes:

```
Image tela_abertura;

Image botao_iniciar;

Context context;
```

Agora dentro do método **TelaMenu**, vamos adicionar a seguinte instrução:

```
public TelaMenu(Context context)
{
    this.context = context;
}
```

Agora dentro do método **onSizeChanged**, vamos adicionar as seguintes instruções abaixo:

```
public void onSizeChanged(int w, int h, int oldw, int oldh)
{
    tela_abertura = new Image(context,
        R.drawable.tela_abertura_mario_bros, 0, 0, w, h);
    botao_iniciar = new Image(context, R.drawable.botao_iniciar, w -
        250, 300, 200, 70);
}
```

No método **onDraw** vamos adicionar as seguintes instruções:

```
public void Draw(Canvas canvas)
{
    tela_abertura.Draw(canvas);
    botao_iniciar.Draw(canvas);
}
```



```
}
```

Agora no método **onTouchEvent** vamos escrever as seguintes instruções:

```
public void onTouchEvent(MotionEvent event)
{
    if(event.getAction() == MotionEvent.ACTION_DOWN) {
        if(botao_iniciar.IsTouch(event.getX(), event.getY()))
        {
            TelasDoJogo.tela = TelaJogo.TELA_FASE;
        }
    }
}
```

Se lembra que na classe principal **GameMain** no seu método construtor definimos que a tela que será processada inicialmente não seria a tela de menu ? Pois bem, como acabamos de codificar a classe **TelaMenu**, vamos executar então o nosso jogo para conferir o resultado, como demonstra a figura seguinte:



**Jogo em execução – Exibindo a tela de menu**

Agora vamos para a tela da fase do jogo, representado pela classe **TelaFase**. Na classe **TelaFase**, dentro da seção de declaração de atributos, vamos adicionar as seguintes instruções abaixo:

```
Scene scene;
```



# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

```
Character mario;  
  
Image ceu;  
  
Image chao, chao2;  
  
Image seta_direita, seta_esquerda, botao_pulo;  
  
Context context;  
  
int largura_tela;  
  
enum Sentido { DIREITA, ESQUERDA, PARADO }  
Sentido sentidoMario = Sentido.PARADO;
```

Agora dentro do método construtor **TelaFase** vamos adicionar as seguintes instruções:

```
public TelaFase(Context context)  
{  
    this.context = context;  
    scene = new Scene();  
}
```

Agora dentro do método **onSizeChanged** vamos adicionar as seguintes instruções abaixo:

```
public void onSizeChanged(int w, int h, int oldw, int oldh)  
{  
  
    largura_tela = w;  
  
    ceu = new Image(context, R.drawable.ceu, 0, 0, w, h);  
  
    chao = new Image(context, R.drawable.chao, 0, h - 90, w, h);  
    chao.SetTag("chao");  
  
    chao2 = new Image(context, R.drawable.chao, w + 250, h - 90, w, h);  
    chao2.SetTag("chao");  
  
    mario = new Character(context, (w / 2) - (66 / 2), 20, 86, 85);  
    mario.AddNewSpriteIdle(R.drawable.mario_parado);  
    mario.AddNewSpriteRunning(R.drawable.mario_correndo_1);  
    mario.AddNewSpriteRunning(R.drawable.mario_correndo_2);  
    mario.AddNewSpriteJumping(R.drawable.mario_pulando);  
    mario.Idle(3, false);  
  
    mario.AddCollisionElementOfFallByTag("chao");  
  
    seta_esquerda = new Image(context, R.drawable.seta_esquerda, 0, h -  
        96, 96, 96);  
    seta_direita = new Image(context, R.drawable.seta_direita, w - 96, h -  
        96, 96, 96);
```



# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

```
        botao_pulo = new Image(context, R.drawable.botao_pulo, w - (96 * 2), h
        - 96, 96, 96);

        scene.Add(ceu);
        scene.Add(chao);
        scene.Add(chao2);
        scene.Add(mario);
    }
```

Agora dentro do método **Update** vamos adicionar as seguintes instruções:

```
public void Update()
{
    if(sentidoMario == Sentido.DIREITA)
    {
        chao.MoveByX(-10);
        chao2.MoveByX(-10);

        if(chao.GetX() < -chao.GetWidth())
            chao.SetX(chao2.GetX() + chao2.GetWidth() + 250);

        if(chao2.GetX() < -chao2.GetWidth())
            chao2.SetX(chao.GetX() + chao.GetWidth() + 250);
    }
    else if(sentidoMario == Sentido.ESQUERDA) {

        chao.MoveByX(10);
        chao2.MoveByX(10);

        if(chao.GetX() > largura_tela)
            chao.SetX(chao2.GetX() - chao2.GetWidth() - 250);

        if(chao2.GetX() > largura_tela)
            chao2.SetX(chao.GetX() - chao.GetWidth() - 250);
    }

    mario.Update(scene);

    if(mario.GetY() > 500)
    {
        TelasDoJogo.tela = TelaJogo.TELA_GAME_OVER;
    }
}
```

Agora dentro do método **Draw** vamos adicionar as seguintes instruções:

```
public void Draw(Canvas canvas)
{
    scene.Draw(canvas);
    seta_esquerda.Draw(canvas);
    seta_direita.Draw(canvas);
    botao_pulo.Draw(canvas);
}
```

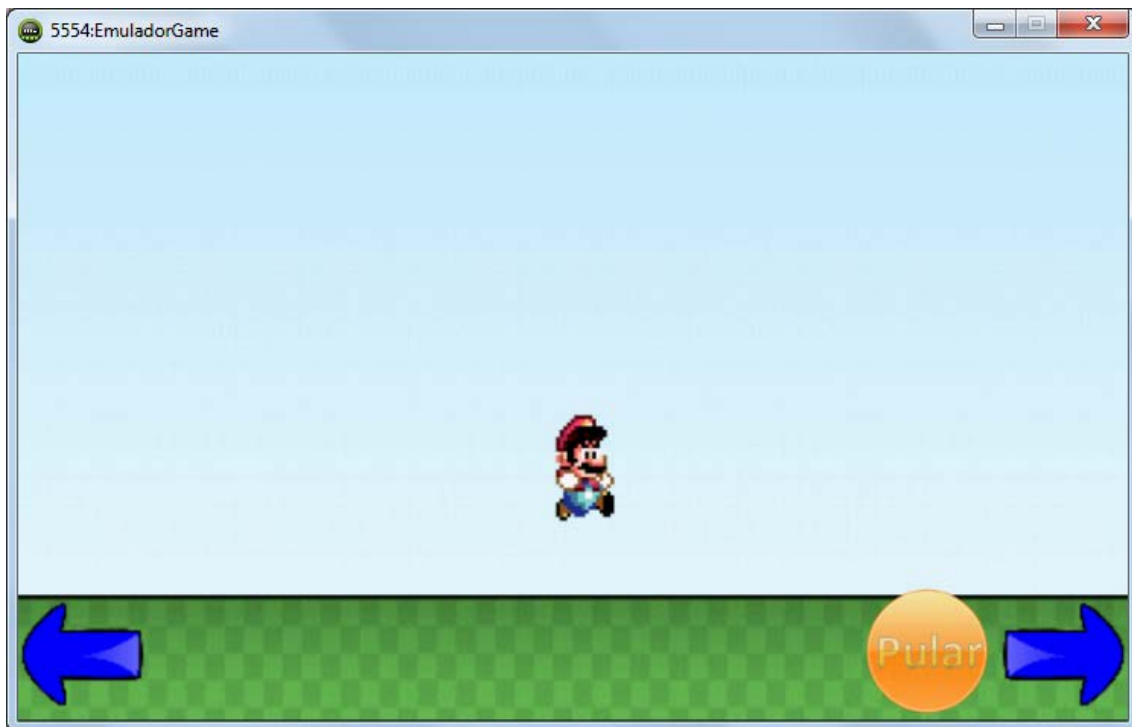


Agora para finalizar, dentro do método **onTouchEvent**, vamos escrever as seguintes linhas de código abaixo:

```
public void onTouchEvent(MotionEvent event)
{
    if(event.getAction() == MotionEvent.ACTION_DOWN)
    {
        if(seta_esquerda.IsTouch(event.getX(), event.getY()))
        {
            if((sentidoMario == Sentido.PARADO) || (sentidoMario ==
                Sentido.DIREITA))
            {
                sentidoMario = Sentido.ESQUERDA;
                mario.RunningToLeft(3, true);
            }
            else {
                sentidoMario = Sentido.PARADO;
                mario.Idle(3, true);
            }
        }
        else if(seta_direita.IsTouch(event.getX(), event.getY()))
        {
            if((sentidoMario == Sentido.PARADO) || (sentidoMario ==
                Sentido.ESQUERDA))
            {
                sentidoMario = Sentido.DIREITA;
                mario.RunningToRight(3, true);
            }
            else {
                sentidoMario = Sentido.PARADO;
                mario.Idle(3, true);
            }
        }
        else if(botao_pulo.IsTouch(event.getX(), event.getY()))
        {
            mario.Jump(3, false);
        }
    }
}
```

Vamos executar o nosso jogo agora. Quando ele é carregado, é mostrado a tela de menu. Para iniciarmos o jogo, basta clicar no botão iniciar. Feito isso, a tela do jogo será executada, conforme você pode conferir em seguida:





**Jogo em execução – Tela do jogo em execução**

Durante o jogo, o nosso personagem (o Mario) pode correr e pular pela fase. Você observar, no nosso jogo existe vários buracos em que o nosso personagem deve “pular” para não sair. Se por acaso o personagem cair, “game over“, ou seja, o jogo termina. Quando isso acontece, nós visualizamos simplesmente uma tela “preta”, pelo fato de ainda não estruturarmos a nossa tela de “game over”. Vamos estruturar agora a nossa tela de “game over”, que será representado pela classe **TelaGameOver**. Vamos na classe **TelaGameOver**, e na seção de declaração de atributos vamos escrever as seguintes linhas de comando:

```
Image tela_game_over;  
Context context;
```

Agora dentro do método construtor **TelaGameOver** vamos adiciona a seguinte instrução:

```
public TelaGameOver(Context context)  
{  
    this.context = context;  
}
```

Agora dentro do método **onSizeChanged** vamos adicionar a seguinte instrução:





# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
public void onSizeChanged(int w, int h, int oldw, int oldh)
{
    tela_game_over = new Image(context, R.drawable.tela_game_over, 0, 0,
    w, h);
}
```

Dentro do método **Draw** , vamos adicionar a seguinte instrução abaixo:

```
public void Draw(Canvas canvas)
{
    tela_game_over.Draw(canvas);
}
```

Vamos executar agora o nosso jogo, para conferir os resultados. Quando você executá-lo, será executado a tela de menu, bastando clicar no botão “Iniciar” para que o jogo comece. Durante o jogo, se você cair no buracão, será mostrado a tela de “game over”, conforme você confere na figura seguinte:



**Jogo em execução – Tela de game over**

Quando o nosso jogo chega na tela do “game over”, ele não retorna mais para nenhuma tela. Vamos modificar o nosso jogo de forma que toda vez que ele chegar na tela de “game over”, o mesmo volte para a tela de menu.

Primeiramente, vamos na classe **TelaFase** para criarmos um método chamado **ReiniciaTela**, que irá ter o seguinte código, como pode conferir em seguida:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
public void ReiniciaTela()
{
    mario.SetX((largura_tela / 2) - (86 / 2));
    mario.SetY(10);
    chao.SetX(0);
    chao2.SetX(largura_tela + 250);
}
```

Agora dentro do método **Update** vamos adicionar a seguinte instrução **em negrito**:

```
public void Update()
{
    :
    if(mario.GetY() > 500)
    {
        ReiniciaTela();
        TelasDoJogo.tela = TelaJogo.TELA_GAME_OVER;
    }
}
```

Antes da tela de “game over” ser exibida, a tela do jogo será reiniciada para começar do início.

Agora dentro da classe **TelaGameOver** vamos declarar o seguinte atributo abaixo:

```
int contaFrame;
```

Essa variável irá funcionar como se fosse um “temporizador”, que vai servir para medirmos (em frames) o tempo em que a tela ficará em exibição, antes de ser trocada para a tela de menu.

Agora dentro do método construtor **TelaGameOver** vamos adicionar a seguinte instrução **em negrito**:

```
public TelaGameOver(Context context)
{
    this.context = context;
    contaFrame = 0;
}
```

Agora dentro do método **Update** vamos adicionar as seguintes instruções:

```
public void Update()
{
    contaFrame++;
}
```



# Apostila de Android

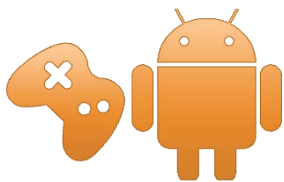
## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

```
if(contaFrame == 20)
{
    contaFrame = 0;
    TelasDoJogo.tela = TelaJogo.TELA_MENU;
}
```

O método acima irá processar o tempo em que a tela de “game over” será exibida (em frames). Quando a variável *contaFrame* atingir o valor 20, ela será reiniciada e a tela de menu será exibida novamente.

Execute agora novamente o jogo, e confira os novos resultados.

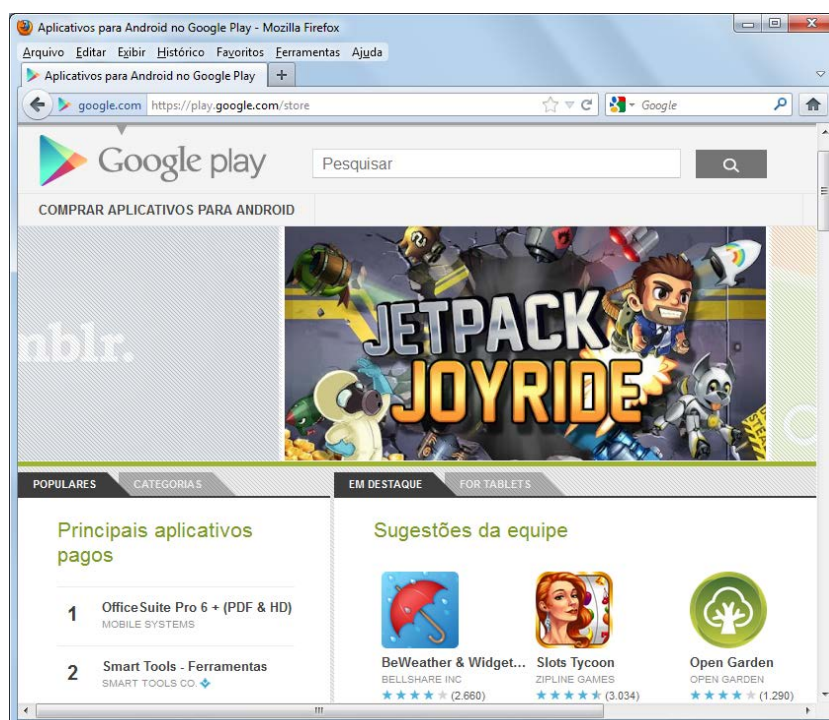


## Capítulo 15 Publicando seus jogos Android no Google Play

**A**té aqui aprendemos a desenvolver nossos jogos voltados para a plataforma Android utilizando a ferramenta Android Developer Tools. Mas agora, como nós fazemos para publicar os nossos jogos na loja virtual de aplicativos do Android, o Google Play. Neste capítulo iremos aprender passo a passo como realizar esta tarefa.

Quem aqui já pensou em desenvolver um jogo Android para distribuí-lo, permitindo que outras pessoas possam utilizá-la (seja um utilitário, jogo e etc.) ? Pois bem, isso já permitindo utilizando um serviço, ou melhor, uma loja virtual de aplicativos do Android, o “Google Play” (o extinto “Android Market”). Para acessarmos o “Google Play” basta acessarmos o seguinte link : <https://play.google.com/store>.

Veja o WebSite conforme é demonstrado na figura seguinte:



Site do “Google Play”



# Apostila de Android

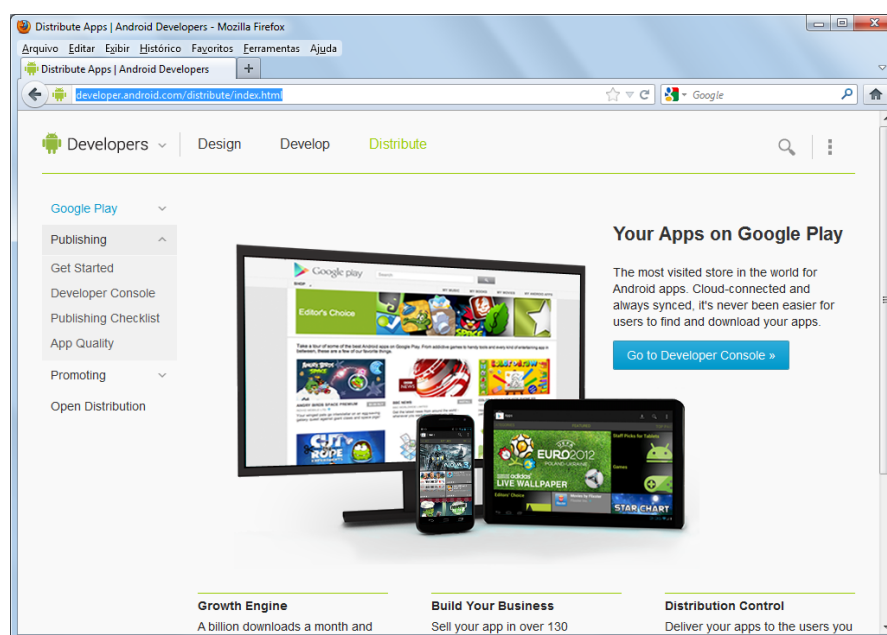
## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

Neste site podemos “publicar” e distribuir nossas aplicações Android, mas , para que suas aplicações possam ser instaladas através deste site, você precisa acessar o site acima através do seu Smartphone ou Tablet (caso contrário, a aplicação não poderá ser instalada).

Para publicarmos um aplicativo no “Google Play”, devemos primeiramente ter uma conta criada no “Google”. Caso você não tenha uma conta no Google é preciso que você crie uma conta para tal propósito. Se você já tem uma conta no Google, basta acessar o seguinte link para começarmos a publicar nossas aplicações: <http://developer.android.com/distribute/index.html>.

Veja o conteúdo na figura seguinte:



**Site dos desenvolvedores do Android**

Depois de carregado o site acima, clique no botão “Go to Developer Console” (que na verdade vai redirecioná-lo para o link <https://play.google.com/apps/publish/>). Caso você não esteja logado, será solicitado que você digite seu login e senha para continuar o processo. Feito isso será aberta a seguinte tela conforme você pode conferir na figura seguinte:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

The screenshot shows the 'Inscrição do desenvolvedor' (Developer Registration) page in the Google Play Developer Console. The browser is Mozilla Firefox, and the URL is <https://play.google.com/apps/publish/signup>. The page title is 'Inscrição do desenvolvedor - Mozilla Firefox'. The Google Play logo and 'ANDROID DEVELOPER CONSOLE' are visible at the top. The main heading is 'Primeiros passos' (First steps). Below it, a note states: 'Antes de publicar um software no Google Play, você deve fazer três coisas:' (Before publishing software on Google Play, you must do three things:). A list of requirements follows: 'Criar um perfil de desenvolvedor' (Create a developer profile), 'Concorde com o Contrato de distribuição do desenvolvedor' (Agree to the Developer Distribution Agreement), and 'Pagar uma taxa de registro (US\$25,00) com cartão de crédito (usando o Google Checkout)' (Pay a registration fee of US\$25.00 with a credit card using Google Checkout). The 'Detalhes da entrada' (Entry details) section explains that the developer profile determines how the app appears to users. It contains several input fields: 'Nome do desenvolvedor' (Developer name), 'Endereço de e-mail' (Email address), 'URL do site' (Website URL), 'Número de telefone' (Phone number), and 'Atualizações por e-mail' (Email updates). The phone number field includes a note: 'Inclua o sinal de adição, o código do país e o código de área. Por exemplo, +1-650-253-0000. [Por que pedimos isso?](#)' (Include the plus sign, country code, and area code. For example, +1-650-253-0000. Why do we ask for this?). The 'Atualizações por e-mail' field has a checked checkbox and the text 'Entrar em contato conosco ocasionalmente sobre desenvolvimento e oportunidades do Google Play' (Occasionally contact us about development and Google Play opportunities). A 'Continuar »' (Continue) button is at the bottom left.

## Android Developer Console

Para publicar nossas aplicações no “Google Play” devemos preencher os campos informados na página acima, aceitar os termos do “Contrato de distribuição do desenvolvedor” do “Google Play” e pagar uma taxa de US\$25,00 (aproximadamente R\$50,00) através do “Google Checkout”. Feito isso você poderá publicar suas aplicações que poderão ser distribuídas tanto de graça quanto pagas.



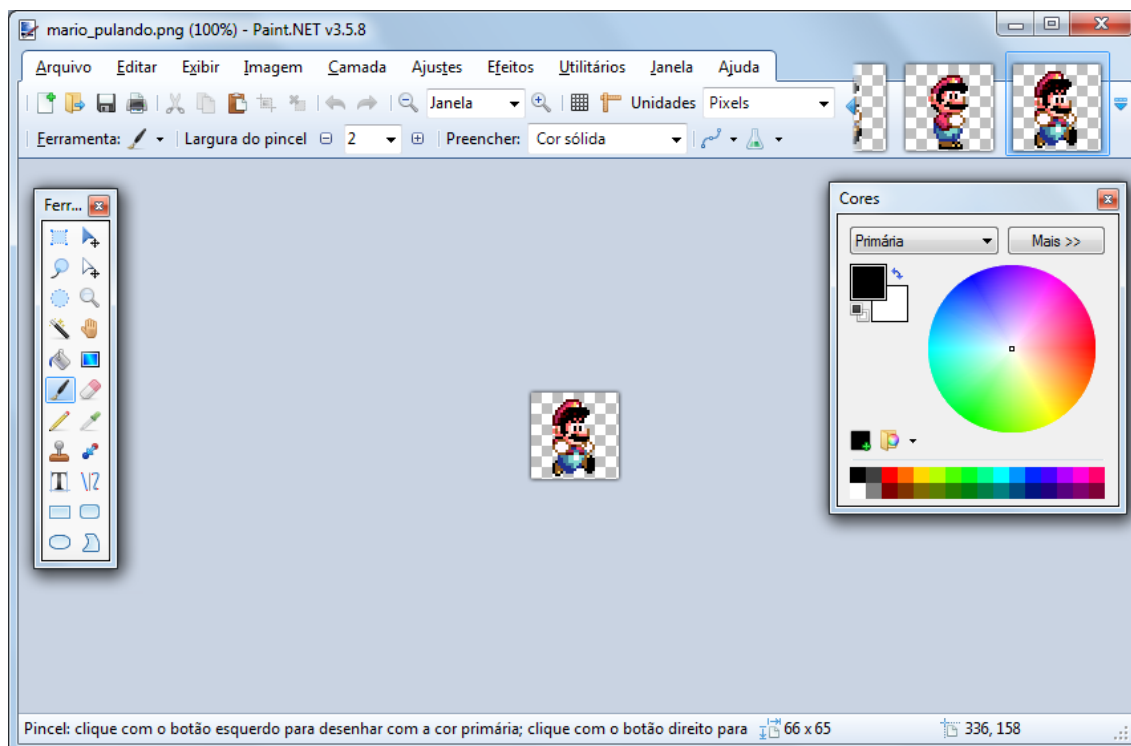


## Apêndice A Sites e ferramentas para a criação de sprites e sons de efeitos

Nesta apostila aprendemos a construir os jogos utilizando as imagens, músicas e sons de efeito fornecidos no próprio material. Mas, e seu eu quisesse criar minhas sprites, escolher meus sons de efeitos e minhas músicas ? Neste tópico, lhe darei dicas “preciosas” de como você criar suas próprias sprites, músicas e sons de efeitos através dos mais variados recursos disponíveis hoje na Web.

### A Ferramenta Paint.NET

Para quem deseja criar suas próprias sprites ou editar um conjunto de sprites já existentes, recomendo “fortemente” o uso da ferramenta Paint.NET. Ela é uma ferramenta muito poderosa, cheia de recursos, é totalmente gratuita. Veja uma demonstração dela na figura abaixo:



**Ferramenta Paint.NET**





O link para download você confere abaixo:

<http://www.getpaint.net/download.html>

### **Sites para obtenção de sprites e backgrounds**

Se você é aquela pessoa que tem a vontade de recriar algum jogo clássico/atual de sucesso (como eu) do Super Nitendo, Master System e etc., para colocar no Android, existe alguns sites que disponibilizam sprites e backgrounds de jogos das mais variadas plataformas (PlayStation, Game Gear, Master System, Super Nintendo e etc.). Irei deixar aqui alguns links destes sites:

#### **The Sprites Resources**

<http://spriters-resource.com/>

#### **Sprite Database**

<http://spritedatabase.net>

#### **Background HQ**

<http://www.bghq.com>

### **A Ferramenta Audacity**

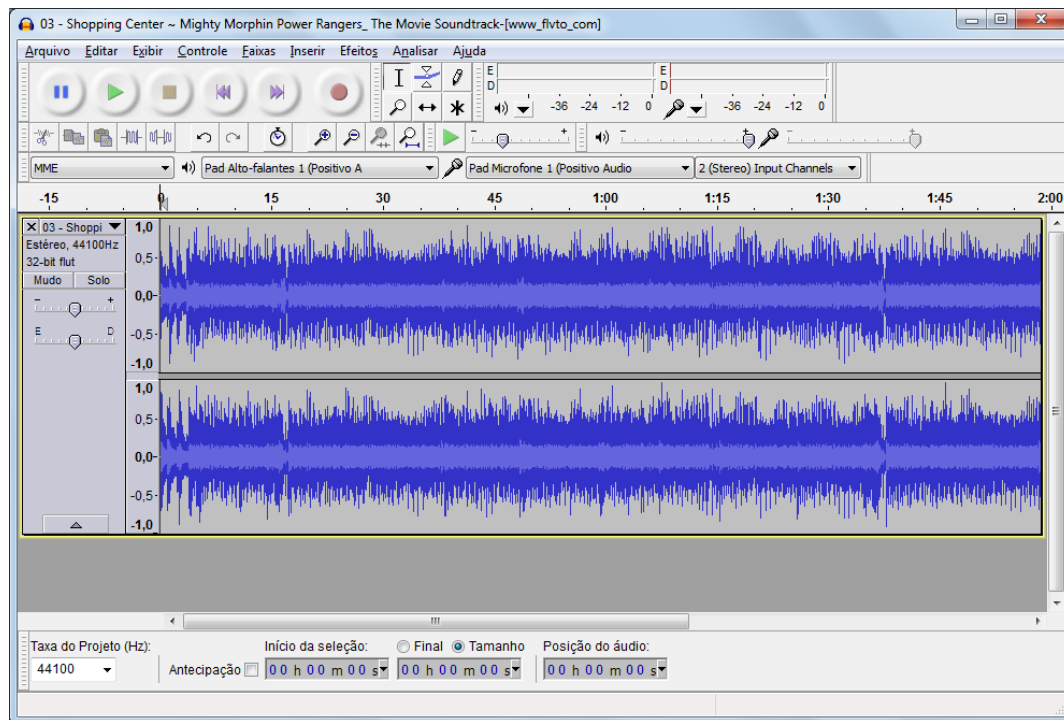
Essa ferramenta é muito útil quando queremos manipular ou editar algum tipo de áudio (como músicas e sons de efeito). Veja a foto dessa ferramenta abaixo:



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)



### A ferramenta Audacity

Com essa ferramenta podemos copiar/recortar e colar trechos de áudio, aplicar efeitos dos mais variados tipos e etc. Essa ferramenta é totalmente gratuita e de código fonte aberto (open source). Seu download pode ser efetuado no seguinte link:

<http://audacity.sourceforge.net/download/?lang=pt>

### Sites para a obtenção de músicas e sons de efeito

Na Internet existe alguns sites úteis onde nele estão disponibilizados músicas e sons de efeito de grandes jogos clássicos que você pode utilizar em seus jogos.

#### **VGMusic**

<http://www.vgmusic.com>

#### **The Midi Shrine**

<http://www.midishrine.com>

#### **SoundBible**

<http://www.soundbible.com>

#### **Sound FX Center**

<http://www.soundfxcenter.com>



## **Apêndice B Guia de referência da biblioteca GameUtil**

Aqui nesta seção você encontra o guia de referência completo da biblioteca GameUtil (que eu desenvolvi), que foi utilizada dentro do nosso modelo de projeto para jogos em Android. Vamos conhecer agora todas as classes e todos os seus métodos:

### **A classe GameElement**

Essa é classe base, que foi utilizada para a construção de todas as outras classes presentes nesse pacote. Vejamos agora seus métodos:

<b>Método</b>	<b>Descrição</b>
<code>public void SetX(int value)</code>	Define o valor da coordenada X do objeto na tela.
<code>public void SetY(int value)</code>	Define o valor da coordenada Y do objeto na tela.
<code>public void SetWidth(int value)</code>	Define a largura do objeto na tela (em pixels).
<code>public void SetHeight(int value)</code>	Define a altura do objeto na tela (em pixels).
<code>public void SetBounds(int x, int y, int w, int h)</code>	Define as coordenadas e o tamanho do objeto na tela.
<code>public void SetTag(String tag)</code>	Define uma "tag" (rótulo) para o objeto.
<code>public int GetX()</code>	Retorna o valor da coordenada X do objeto na tela.
<code>public int GetY()</code>	Retorna o valor da coordenada Y do objeto na tela.
<code>public int GetWidth()</code>	Retorna a largura do objeto.
<code>public int GetHeight()</code>	Retorna a altura do objeto.
<code>public void MoveByX(int value)</code>	Move um objeto na coordenada X, partindo da sua coordenada atual, em pixels. Ex. Se o objeto estiver na coordenada X 30, passando como argumento o valor 10, ele avança para a coordenada X 40. Valores positivos deslocam ele para direita e valores negativos para a esquerda
<code>public void MoveByY(int value)</code>	Move um objeto na coordenada Y, partindo da sua coordenada atual, em pixels. Ex. Se o objeto estiver na coordenada Y 10, passando como argumento o valor 15, ele avança para a coordenada Y 25. Valores positivos deslocam ele para baixo e valores negativos para cima.



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

<pre>public Boolean IsTouch(float x, float y)</pre>	Retorna verdadeiro se o objeto foi “tocado”, nas coordenadas informadas no parâmetro. Usar esse método dentro do evento onTouchEvent.
<pre>public void Draw(Canvas canvas)</pre>	Desenha o objeto na tela

### A classe Image

Essa classe é derivada da classe **GameElement**, e ela tem a função de desenhar imagens na tela. Vamos conhecer agora os seus métodos:

Método	Descrição
<pre>public Image(Context context, int id_image, int x, int y, int w, int h)  public Image(Context context, id_image, int x, int y, int w, int h, String tag)</pre>	Esse método carrega a imagem a ser exibida na tela. No construtor definimos o “context”, a imagem a ser carregada (a <i>Id id_image</i> ), as coordenadas X e Y e o seu tamanho.
<pre>public void SetX(int value)</pre>	Define o valor da coordenada X da imagem na tela.
<pre>public void SetY(int value)</pre>	Define o valor da coordenada Y da imagem na tela.
<pre>public void SetWidth(int value)</pre>	Define a largura da imagem na tela (em pixels).
<pre>public void SetHeight(int value)</pre>	Define a altura da imagem na tela (em pixels).
<pre>public void SetBounds(int x, int y, int w, int h)</pre>	Define as coordenadas e o tamanho da imagem na tela.
<pre>public void SetTag(String tag)</pre>	Define uma “tag” (rótulo) para a imagem.
<pre>public int GetX()</pre>	Retorna o valor da coordenada X da imagem na tela.
<pre>public int GetY()</pre>	Retorna o valor da coordenada Y da imagem na tela.
<pre>public int GetWidth()</pre>	Retorna a largura da imagem.
<pre>public int GetHeight()</pre>	Retorna a altura da imagem.
<pre>public void MoveByX(int value)</pre>	Move uma imagem na coordenada X, partindo da sua coordenada atual, em pixels. Ex. Se a imagem estiver na coordenada X 30, passando como argumento o valor 10, ele avança para a coordenada X 40. Valores positivos deslocam ele para direita e valores negativos para a esquerda
<pre>public void MoveByY(int value)</pre>	Move uma imagem na coordenada Y, partindo da sua coordenada atual, em pixels. Ex. Se a imagem estiver na coordenada Y 10, passando como argumento o valor 15, ele avança para a coordenada Y 25. Valores positivos deslocam ele para baixo e valores



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

	negativos para cima.
<pre>public Boolean IsTouch(float x, float y)</pre>	Retorna verdadeiro se a imagem foi “tocada”, nas coordenadas informadas no parâmetro. Usar esse método dentro do evento onTouchEvent.
<pre>public void Draw(Canvas canvas)</pre> <pre>public void Draw(Canvas canvas, FlipEffect effect)</pre>	Desenha a imagem na tela. Nesta classe, o método Draw possui um segundo parâmetro, que permite que a imagem seja invertida na horizontal.

### A classe AnimationSprites

Essa classe é derivada da classe **GameElement**, e ela tem a função de realizar uma animação de sprites através de um conjunto de imagens. Vamos conhecer agora os seus métodos:

Método	Descrição
<pre>public AnimationSprites(Context context, int x, int y, int w, int h)</pre>	Esse método carrega o objeto na memória. No construtor definimos o “context”, as coordenadas X e Y e o tamanho das sprites que serão exibidas.
<pre>public void Add(int Id)</pre>	Neste método adicionamos uma imagem que fará parte da nossa animação.
<pre>public void Start(int frames, boolean loop)</pre>	Inicia a animação de sprites. O intervalo de troca entre as imagens é dada em frames (primeiro parâmetro), podendo essa animação estar em loop (caso o segundo parâmetro seja true) ou não (caso o segundo parâmetro seja false).
<pre>public void Stop()</pre>	Encerra a animação de sprites.
<pre>public void SetX(int value)</pre>	Define o valor da coordenada X do objeto na tela.
<pre>public void SetY(int value)</pre>	Define o valor da coordenada Y do objeto na tela.
<pre>public void SetWidth(int value)</pre>	Define a largura do objeto na tela (em pixels).
<pre>public void SetHeight(int value)</pre>	Define a altura do objeto na tela (em pixels).
<pre>public void SetBounds(int x, int y, int w, int h)</pre>	Define as coordenadas e o tamanho do objeto na tela.
<pre>public void SetTag(String tag)</pre>	Define uma “tag” (rótulo) para o objeto.
<pre>public int GetX()</pre>	Retorna o valor da coordenada X do objeto na tela.
<pre>public int GetY()</pre>	Retorna o valor da coordenada Y do objeto na tela.
<pre>public int GetWidth()</pre>	Retorna a largura do objeto.
<pre>public int GetHeight()</pre>	Retorna a altura do objeto.
	Move o objeto na coordenada X, partindo da sua coordenada atual, em pixels. Ex. Se o objeto estiver na coordenada X 30, passando



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

<pre>public void MoveByX(int value)</pre>	como argumento o valor 10, ele avança para a coordenada X 40. Valores positivos deslocam ele para direita e valores negativos para a esquerda
<pre>public void MoveByY(int value)</pre>	Move o objeto na coordenada Y, partindo da sua coordenada atual, em pixels. Ex. Se o objeto estiver na coordenada Y 10, passando como argumento o valor 15, ele avança para a coordenada Y 25. Valores positivos deslocam ele para baixo e valores negativos para cima.
<pre>public boolean IsTouch(float x, float y)</pre>	Retorna verdadeiro se o objeto foi “tocado”, nas coordenadas informadas no parâmetro. Usar esse método dentro do evento onTouchEvent.
<pre>public void Draw(Canvas canvas)</pre> <pre>public void Draw(Canvas canvas, FlipEffect effect)</pre>	Mostra a animação na tela. Nesta classe, o método Draw possui um segundo parâmetro, que permite que as animações sejam invertidas na horizontal.

### A classe Collision

Essa é classe é responsável pela detecção de colisão entre dois objeto (ela pode ser usada sem instância). Vamos ser seu método abaixo:

Método	Descrição
<pre>public boolean Check(GameElement obj1, GameElement obj2)</pre>	Este método avalia se ocorreu uma colisão entre os dois objetos do parâmetro. Caso a colisão aconteça, o método retorna true, caso contrário, retorna false.

### A classe Character

Essa classe é derivada da classe **GameElement**, e com ela podemos construir um personagem para o jogo. Essa classe é bastante “rica” e “poderosa”, possui vários métodos e funcionalidades que facilitam a construção de um personagem (animação, física e etc.). Vamos ver seus métodos abaixo:

Método	Descrição
<pre>public Character(Context context, int x, int y, int w, int h)</pre>	Esse método carrega o objeto na memória. No construtor definimos o “context”, as coordenadas X e Y e o tamanho do personagem que será exibido.
<pre>public void AddNewSpriteIdle(int Id)</pre>	Nesse método adicionamos imagens que vão representar a animação dele parado.
<pre>public void AddNewSpriteRunning(int</pre>	Nesse método adicionamos imagens que





# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

Id)	vão representar a animação dele correndo.
public void AddNewSpriteWalking(int Id)	Nesse método adicionamos imagens que vão representar a animação dele andando.
public void AddNewSpriteJumping(int Id)	Nesse método adicionamos imagens que vão representar a animação dele pulando.
public void AddNewSpriteDie(int Id)	Nesse método adicionamos imagens que vão representar a animação dele morrendo.
public void AddNewSpriteDamage(int Id)	Nesse método adicionamos imagens que vão representar a animação dele sofrendo dano.
public void AddNewSpriteAttack1(int Id)	Nesse método adicionamos imagens que vão representar a animação do ataque número 1.
public void AddNewSpriteAttack2(int Id)	Nesse método adicionamos imagens que vão representar a animação do ataque número 2.
public void AddNewSpriteAttack3(int Id)	Nesse método adicionamos imagens que vão representar a animação do ataque número 3.
public void AddNewSpriteAttack4(int Id)	Nesse método adicionamos imagens que vão representar a animação do ataque número 4.
public void AddNewSpriteAttack5(int Id)	Nesse método adicionamos imagens que vão representar a animação do ataque número 5.
public void AddNewSpriteAttack1OnJump(int Id)	Nesse método adicionamos imagens que vão representar a animação do ataque número 1, quando estiver no ar.
public void AddNewSpriteAttack2OnJump(int Id)	Nesse método adicionamos imagens que vão representar a animação do ataque número 2, quando estiver no ar.
public void AddNewSpriteAttack3OnJump(int Id)	Nesse método adicionamos imagens que vão representar a animação do ataque número 3, quando estiver no ar.
public void AddNewSpriteAttack4OnJump(int Id)	Nesse método adicionamos imagens que vão representar a animação do ataque número 4, quando estiver no ar.
public void AddNewSpriteAttack5OnJump(int Id)	Nesse método adicionamos imagens que vão representar a animação do ataque número 5, quando estiver no ar.
public void Attack1(int frames)	Executa a animação do ataque numero 1.
public void Attack2(int frames)	Executa a animação do ataque numero 2.
public void Attack3(int frames)	Executa a animação do ataque numero 3.
public void Attack4(int frames)	Executa a animação do ataque numero 4.
public void Attack5(int frames)	Executa a animação do ataque numero 5.
public void SufferDamage(int frames)	Executa a animação dele sofrendo dano.





# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

<pre>public void Idle(int frames, bool loop)</pre>	Executa a animação dele parado.
<pre>public void Jump(int frames, bool loop)</pre>	Executa a animação dele pulando.
<pre>public void RunningToRight(int frames, bool loop)</pre>	Executa a animação dele correndo para a direita.
<pre>public void RunningToLeft(int frames, bool loop)</pre>	Executa a animação dele correndo para a esquerda.
<pre>public void WalkingToRight(int frames, bool loop)</pre>	Executa a animação dele andando para a direita.
<pre>public void WalkingToLeft(int frames, bool loop)</pre>	Executa a animação dele andando para a esquerda.
<pre>public void AddCollisionElementOfFallByTag(String tag)</pre>	Neste método adicionamos os rótulos nos quais os objetos do cenário possui, que será o ponto de colisão para o nosso personagem, durante a queda.
<pre>public void AddCollisionElementOfSideByTag(String tag)</pre>	Neste método adicionamos os rótulos nos quais os objetos do cenário possui, que será o ponto de colisão para o nosso personagem, enquanto ele estiver andando.
<pre>public boolean CollisionBySide(Scene scene)</pre>	Verifica se houve alguma colisão pelos lados por parte do personagem, baseado nas tags adicionadas através do método acima.
<pre>public void Update(Scene scene)</pre>	Processa o pulo e a física do personagem.
<pre>public void SetX(int value)</pre>	Define o valor da coordenada X do objeto na tela.
<pre>public void SetY(int value)</pre>	Define o valor da coordenada Y do objeto na tela.
<pre>public void SetWidth(int value)</pre>	Define a largura do objeto na tela (em pixels).
<pre>public void SetHeight(int value)</pre>	Define a altura do objeto na tela (em pixels).
<pre>public void SetBounds(int x, int y, int w, int h)</pre>	Define as coordenadas e o tamanho do objeto na tela.
<pre>public void SetTag(String tag)</pre>	Define uma "tag" (rótulo) para a imagem.
<pre>public int GetX()</pre>	Retorna o valor da coordenada X da imagem na tela.
<pre>public int GetY()</pre>	Retorna o valor da coordenada Y da imagem na tela.
<pre>public int GetWidth()</pre>	Retorna a largura da imagem.
<pre>public int GetHeight()</pre>	Retorna a altura da imagem.
<pre>public void MoveByX(int value)</pre>	Move uma imagem na coordenada X, partindo da sua coordenada atual, em pixels. Ex. Se a imagem estiver na coordenada X 30, passando como argumento o valor 10, ele avança para a



# Apostila de Android

## Programando Passo a Passo

### Desenvolvimento de Jogos (Edição Completa)

	coordenada X 40. Valores positivos deslocam ele para direita e valores negativos para a esquerda
<pre>public void MoveByY(int value)</pre>	Move uma imagem na coordenada Y, partindo da sua coordenada atual, em pixels. Ex. Se a imagem estiver na coordenada Y 10, passando como argumento o valor 15, ele avança para a coordenada Y 25. Valores positivos deslocam ele para baixo e valores negativos para cima.
<pre>public boolean IsTouch(float x, float y)</pre>	Retorna verdadeiro se a imagem foi "tocada", nas coordenadas informadas no parâmetro. Usar esse método dentro do evento onTouchEvent.
<pre>public void Draw(Canvas canvas)</pre>	Desenha o objeto na tela.

### A classe Scene

Essa é classe é responsável por gerenciar os elementos que serão visualizados na tela do jogo. Vamos ser seus métodos abaixo:

Método	Descrição
<pre>public Scene()</pre>	Esse método é responsável por carregar o objeto na memória.
<pre>public void Add(GameElement e)</pre>	Adiciona um objeto do tipo GameElement , para que o mesmo possa ser gerenciado.
<pre>public GameElement Get(int index)</pre>	Esse método retorna um elemento do tipo GameElement presente dentro do objeto Scene, baseado em seu índice.
<pre>public ArrayList&lt;GameElements&gt; Elements</pre>	Retorna um array (de GameElement) contendo todos os elementos presentes dentro do objeto Scene.
<pre>public void Remove(int index)</pre> <pre>public void Remove(GameElement element)</pre>	Remove um elemento do objeto Scene, baseado em seu índice ou em sua instância.
<pre>public int GetCount()</pre>	Retorna o número total de elementos dentro do objeto Scene.
<pre>public int GetCountByType(String type)</pre>	Retorna o número total de elementos de um tipo específico, passado pelo parâmetro.
<pre>public int GetCountByTag(String type)</pre>	Retorna o número total de elementos que possua uma tag em específico, passada pelo parâmetro.
<pre>public void Draw(Canvas canvas)</pre>	Desenha todos os elementos presentes dentro do objeto Scene.
	Define a coordenada X e todos os



# Apostila de Android

## Programando Passo a Passo

Desenvolvimento de Jogos (Edição Completa)

<pre>public void SetX(int value)</pre>	elementos presentes dentro do objeto Scene, com o mesmo argumento passado no parâmetro.
<pre>public void SetY(int value)</pre>	Define a coordenada Y e todos os elementos presentes dentro do objeto Scene, com o mesmo argumento passado no parâmetro.
<pre>public void MoveByX(int value)</pre>	Move todos os elementos na coordenada X, partindo da posição atual de cada um.
<pre>public void MoveByY(int value)</pre>	Move todos os elementos na coordenada Y, partindo da posição atual de cada um.



## **Conclusão a respeito do material**

Nesta apostila aprendemos a desenvolver jogos para a plataforma Android. Vimos um pouco sobre a plataforma Android, como ela surgiu e tudo mais. Aprendemos a instalar e configurar Android Developer Tools, que é a ferramenta que utilizamos para criar os nossos jogos. Conhecemos também um pouco da estrutura de projeto de jogos em Android, voltado para um modelo que eu desenvolvi com essa finalidade (utilizando a biblioteca GameUtil, também desenvolvida por mim para facilitar a construção de jogos nessa plataforma).

Depois do conhecimento do modelo apresentado, aprendemos passo a passo a construir um jogo conhecendo os componentes da biblioteca, como Image , AnimationSprites, Collision, Scene, Character e etc. Aprendemos conceitos de programação com Inteligência Artificial, como criar elementos dinâmicos durante a execução do jogo, como criar uma HUD, como trabalhar com mais de uma tela no jogo e etc.

Espero que esse material lhe tenha sido útil.

**Abraços**