HOW TO MAKE PACHINKO IN ANDROID

Introduction

Android is an operating system for mobile devices such as smartphones and tablet computers. It is developed by the Open Handset Alliance led by Google.

Developing applications for Android devices is facilitated by a group of tools that are provided with the SDK. You can access these tools through an Eclipse plugin called ADT (Android Development Tools) or from the command line. Developing with Eclipse is the preferred method because it can directly invoke the tools that you need while developing applications.

However, you may choose to develop with another IDE or a simple text editor and invoke the tools on the command line or with scripts. This is a less streamlined way to develop because you will sometimes have to call command line tools manually, but you will have access to the same number of features that you would have in Eclipse.

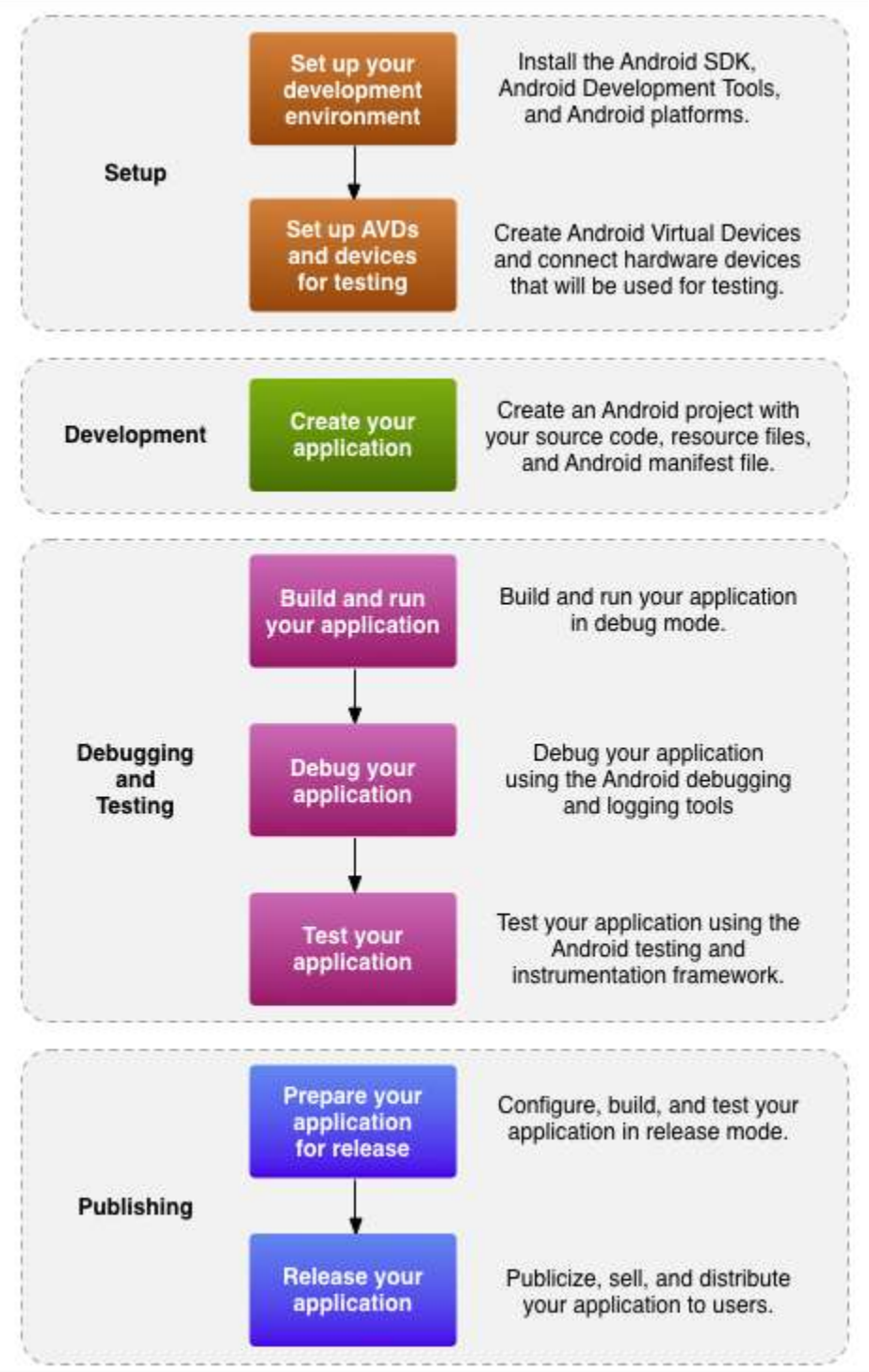

Figure 1. The development process for Android applications.

The basic steps for developing applications (with or without Eclipse) are shown in figure 1. The development steps encompass four development phases, which include:

- **Setup**

  During this phase you install and set up your development environment. You also create Android Virtual Devices (AVDs) and connect hardware devices on which you can install your applications.

- **Development**

  During this phase you set up and develop your Android project, which contains all of the source code and resource files for your application.

- **Debugging and Testing**

  During this phase you build your project into a debuggable `.apk` package that you can install and run on the emulator or an Android-powered device. If you are using Eclipse, builds are generated each time you project is saved. If you're using another IDE, you can build your project using Ant and install it on a device using adb.

  Next, you debug your application using a JDWP-compliant debugger along with the debugging and logging tools that are provided with the Android SDK. Eclipse already comes packaged with a compatible debugger.

  Last, you test your application using various Android SDK testing tools.

- **Publishing**

  During this phase you configure and build your application for release and distribute your application to users.

Getting Started

Here's an overview of the steps you must follow to set up the Android SDK:

1. Prepare your development computer and ensure it meets the system requirements.
2. Install the SDK starter package from the table above. (If you're on Windows, download the installer for help with the initial setup.)
3. Install the ADT Plugin for Eclipse (if you'll be developing in Eclipse).
4. Add Android platforms and other components to your SDK.
5. Explore the contents of the Android SDK (optional).

Installing the SDK

This part describes how to install the Android SDK and set up your development environment for the first time.

If you encounter any problems during installation, see the Troubleshooting section at the bottom of this page.

Updating?

If you already have an Android SDK, use the Android SDK and AVD Manager tool to install updated tools and new Android platforms into your existing environment. For information about how to do that, see Adding SDK Components.

Step 1. Preparing Your Development Computer

Before getting started with the Android SDK, take a moment to confirm that your development computer meets the System Requirements. In particular, you might need to install the JDK, if you don't have it already.

If you will be developing in Eclipse with the Android Development Tools (ADT) Plugin—the recommended path if you are new to Android—make sure that you have a suitable version of Eclipse installed on your computer as described in the System Requirements document. If you need to install Eclipse, you can download it from this location:

http://www.eclipse.org/downloads/

The "Eclipse Classic" version is recommended. Otherwise, a Java or RCP version of Eclipse is recommended.

Step 2. Downloading the SDK Starter Package

The SDK starter package is not a full development environment—it includes only the core SDK Tools, which you can use to download the rest of the SDK components (such as the latest Android platform).

If you haven't already, get the latest version of the SDK starter package from the SDK download page.

If you downloaded a `.zip` or `.tgz` package (instead of the SDK installer), unpack it to a safe location on your machine. By default, the SDK files are unpacked into a directory named `android-sdk-<machine-platform>`.

If you downloaded the Windows installer (`.exe` file), run it now and it will check whether the proper Java SE Development Kit (JDK) is installed (installing it, if necessary), then install the SDK Tools into a default location (which you can modify).

Make a note of the name and location of the SDK directory on your system—you will need to refer to the SDK directory later, when setting up the ADT plugin and when using the SDK tools from the command line.

Step 3. Installing the ADT Plugin for Eclipse

Android offers a custom plugin for the Eclipse IDE, called Android Development Tools (ADT), that is designed to give you a powerful, integrated environment in which to build Android applications. It extends the capabilites of Eclipse to let you quickly set up new Android projects, create an application UI, debug your applications using the Android SDK tools, and even export signed (or unsigned) APKs in order to distribute your application. In general, developing in Eclipse with ADT is a highly recommended approach and is the fastest way to get started with Android.

If you'd like to use ADT for developing Android applications, install it now. Read Installing the ADT Plugin for step-by-step installation instructions, then return here to continue the last step in setting up your Android SDK.

If you prefer to work in a different IDE, you do not need to install Eclipse or ADT. Instead, you can directly use the SDK tools to build and debug your application. The Introduction to Android application development outlines the major steps that you need to complete when developing in Eclipse or other IDEs.

Step 4. Adding Platforms and Other Components

The last step in setting up your SDK is using the Android SDK and AVD Manager (a tool included in the SDK starter package) to download essential SDK components into your development environment.

The SDK uses a modular structure that separates the major parts of the SDK—Android platform versions, add-ons, tools, samples, and documentation—into a set of separately installable components. The SDK starter package, which you've already downloaded, includes only a single component: the latest version of the SDK Tools. To develop an Android application, you also need to download at least one Android platform and the associated platform tools. You can add other components and platforms as well, which is highly recommended.

If you used the Windows installer, when you complete the installation wizard, it will launch the Android SDK and AVD Manager with a default set of platforms and other components selected for you to install. Simply click **Install** to accept the recommended set of components and install them. You can then skip to Step 5, but we recommend you first read the section about the Available Components to better understand the components available from the Android SDK and AVD Manager.

You can launch the Android SDK and AVD Manager in one of the following ways:

- From within Eclipse, select **Window > Android SDK and AVD Manager**.
- On Windows, double-click the `SDK Manager.exe` file at the root of the Android SDK directory.
- On Mac or Linux, open a terminal and navigate to the `tools/` directory in the Android SDK, then execute:

  ```
  android
  ```

To download components, use the graphical UI of the Android SDK and AVD Manager to browse the SDK repository and select new or updated components (see figure 1). The Android SDK and AVD Manager installs the selected components in your SDK environment. For information about which components you should download, see Recommended Components.

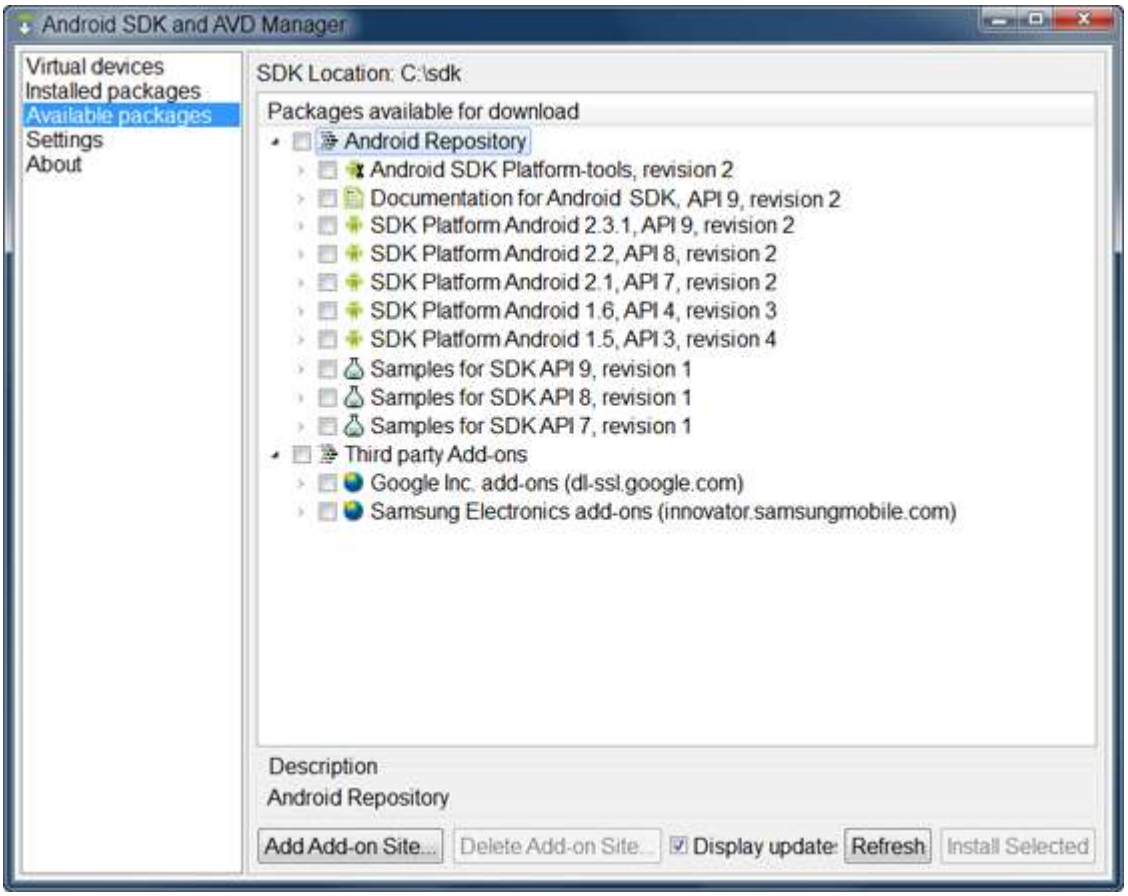

Figure 1. The Android SDK and AVD Manager's **Available Packages** panel, which shows the SDK components that are available for you to download into your environment.

Available Components

By default, there are two repositories of components for your SDK: *Android Repository* and *Third party Add-ons*.

The *Android Repository* offers these types of components:

- **SDK Tools —** Contains tools for debugging and testing your application and other utility tools. These tools are installed with the Android SDK starter package and receive periodic updates. You can access these tools in the `<sdk>/tools/` directory of your SDK. To learn more about them, see SDK Tools in the developer guide.
- **SDK Platform-tools —** Contains platform-dependent tools for developing and debugging your application. These tools support the latest features of the Android platform and are typically updated only when a new platform becomes available. You can access these tools in the `<sdk>/platform-tools/` directory. To learn more about them, see Platform Tools in the developer guide.
- **Android platforms —** An SDK platform is available for every production Android platform deployable to Android-powered devices. Each SDK platform component includes a fully

compliant Android library, system image, sample code, and emulator skins. To learn more about a specific platform, see the list of platforms that appears under the section "Downloadable SDK Components" on the left part of this page.

- **USB Driver for Windows** (Windows only) — Contains driver files that you can install on your Windows computer, so that you can run and debug your applications on an actual device. You *do not* need the USB driver unless you plan to debug your application on an actual Android-powered device. If you develop on Mac OS X or Linux, you do not need a special driver to debug your application on an Android-powered device. See Using Hardware Devices for more information about developing on a real device.
- **Samples** — Contains the sample code and apps available for each Android development platform. If you are just getting started with Android development, make sure to download the samples to your SDK.
- **Documentation** — Contains a local copy of the latest multiversion documentation for the Android framework API.

The *Third party Add-ons* provide components that allow you to create a development environment using a specific Android external library (such as the Google Maps library) or a customized (but fully compliant) Android system image. You can add additional Add-on repositories by clicking **Add Add-on Site**.

Recommended Components

The SDK repository contains a range of components that you can download. Use the table below to determine which components you need, based on whether you want to set up a basic, recommended, or full development environment:

Environment	SDK Component	Comments
Basic	SDK Tools	If you've just installed the SDK starter package, then you already have the latest version of this component. The SDK Tools component is required to develop an Android application. Make sure you keep this up to date.
	SDK Platform-tools	This includes more tools that are required for application development. These tools are platform-dependent and typically update only when a new SDK platform is made available, in order to support new features in the platform. These tools are always backward compatible with older platforms, but you must be sure that you have the latest version of these tools when you install a new SDK platform.
	SDK platform	You need to download **at least one platform** into your environment, so that you will be able to compile your application and set up an Android Virtual Device (AVD) to run it on (in the emulator). To start with, just download the latest

		version of the platform. Later, if you plan to publish your application, you will want to download other platforms as well, so that you can test your application on the full range of Android platform versions that your application supports.
	+	
Recommended (plus Basic)	Documentation	The Documentation component is useful because it lets you work offline and also look up API reference information from inside Eclipse.
	Samples	The Samples components give you source code that you can use to learn about Android, load as a project and run, or reuse in your own app. Note that multiple samples components are available — one for each Android platform version. When you are choosing a samples component to download, select the one whose API Level matches the API Level of the Android platform that you plan to use.
	Usb Driver	The Usb Driver component is needed only if you are developing on Windows and have an Android-powered device on which you want to install your application for debugging and testing. For Mac OS X and Linux platforms, no special driver is needed.
	+	
Full (plus Recommended)	Google APIs	The Google APIs add-on gives your application access to the Maps external library, which makes it easy to display and manipulate Maps data in your application.
	Additional SDK Platforms	If you plan to publish your application, you will want to download additional platforms corresponding to the Android platform versions on which you want the application to run. The recommended approach is to compile your application against the lowest version you want to support, but test it against higher versions that you intend the application to run on. You can test your applications on different platforms by running in an Android Virtual Device (AVD) on the Android emulator.

Once you've installed at least the basic configuration of SDK components, you're ready to start developing Android apps. The next section describes the contents of the Android SDK to familiarize you with the components you've just installed.

For more information about using the Android SDK and AVD Manager, see the Adding SDK Components document.

Step 5. Exploring the SDK (Optional)

Once you've installed the SDK and downloaded the platforms, documentation, and add-ons that you need, we suggest that you open the SDK directory and take a look at what's inside.

The table below describes the full SDK directory contents, with components installed.

Name	Description
`add-ons/`	Contains add-ons to the Android SDK development environment, which let you develop against external libraries that are available on some devices.
`docs/`	A full set of documentation in HTML format, including the Developer's Guide, API Reference, and other information. To read the documentation, load the file `offline.html` in a web browser.
`platform-tools/`	Contains platform-dependent development tools that may be updated with each platform release. The platform tools include the Android Debug Bridge (`adb`) as well as other tools that you don't typically use directly. These tools are separate from the development tools in the `tools/` directory because these tools may be updated in order to support new features in the latest Android platform.
`platforms/`	Contains a set of Android platform versions that you can develop applications against, each in a separate directory.
`<platform>/`	Platform version directory, for example "android-11". All platform version directories contain a similar set of files and subdirectory structure. Each platform directory also includes the Android library (`android.jar`) that is used to compile applications against the platform version.
`samples/`	Sample code and apps that are specific to platform version.
`tools/`	Contains the set of development and profiling tools that are platform-independent, such as the emulator, the Android SDK and AVD Manager, `ddms`, `hierarchyviewer` and more. The tools in this directory may be updated at any time using the Android SDK and AVD Manager and are independent of platform

	releases.
`SDK Readme.txt`	A file that explains how to perform the initial setup of your SDK, including how to launch the Android SDK and AVD Manager tool on all platforms.
`SDK Manager.exe`	Windows SDK only. A shortcut that launches the Android SDK and AVD Manager tool, which you use to add components to your SDK.

Optionally, you might want to add the location of the SDK's `tools/` and `platform-tools` to your `PATH` environment variable, to provide easy access to the tools.

How to make simple Pachinko using Android

Shake detection

If we want to detect android shake motion, we can use SensorEventListener. In this tutorial, I use a class for managing shake event. I named it ShakeEventListener.

Code for ShakeEventListener :

```
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;


/**
 * Listener that detects shake gesture.
 */
public class ShakeEventListener implements SensorEventListener {


  /** Minimum movement force to consider. */
  private static final int MIN_FORCE = 10;

  /**
   * Minimum times in a shake gesture that the direction of movement needs to
   * change.
   */
  private static final int MIN_DIRECTION_CHANGE = 3;

  /** Maximum pause between movements. */
  private static final int MAX_PAUSE_BETHWEEN_DIRECTION_CHANGE = 200;

  /** Maximum allowed time for shake gesture. */
  private static final int MAX_TOTAL_DURATION_OF_SHAKE = 400;

  /** Time when the gesture started. */
  private long mFirstDirectionChangeTime = 0;

  /** Time when the last movement started. */
  private long mLastDirectionChangeTime;
```

```
/** How many movements are considered so far. */
private int mDirectionChangeCount = 0;

/** The last x position. */
private float lastX = 0;

/** The last y position. */
private float lastY = 0;

/** The last z position. */
private float lastZ = 0;

/** OnShakeListener that is called when shake is detected. */
private OnShakeListener mShakeListener;

/**
 * Interface for shake gesture.
 */
public interface OnShakeListener {

  /**
   * Called when shake gesture is detected.
   */
  void onShake();
}

public void setOnShakeListener(OnShakeListener listener) {
  mShakeListener = listener;
}

@Override
public void onSensorChanged(SensorEvent se) {
  // get sensor data
  float x = se.values[SensorManager.DATA_X];
  float y = se.values[SensorManager.DATA_Y];
  float z = se.values[SensorManager.DATA_Z];

  // calculate movement
  float totalMovement = Math.abs(x + y + z - lastX - lastY - lastZ);

  if (totalMovement > MIN_FORCE) {

    // get time
    long now = System.currentTimeMillis();

    // store first movement time
    if (mFirstDirectionChangeTime == 0) {
      mFirstDirectionChangeTime = now;
      mLastDirectionChangeTime = now;
    }

    // check if the last movement was not long ago
    long lastChangeWasAgo = now - mLastDirectionChangeTime;
    if (lastChangeWasAgo < MAX_PAUSE_BETHWEEN_DIRECTION_CHANGE) {

      // store movement data
```

```
        mLastDirectionChangeTime = now;
        mDirectionChangeCount++;

        // store last sensor data
        lastX = x;
        lastY = y;
        lastZ = z;

        // check how many movements are so far
        if (mDirectionChangeCount >= MIN_DIRECTION_CHANGE) {

          // check total duration
          long totalDuration = now - mFirstDirectionChangeTime;
          if (totalDuration < MAX_TOTAL_DURATION_OF_SHAKE) {
            mShakeListener.onShake();
            resetShakeParameters();
          }
        }

      } else {
        resetShakeParameters();
      }
    }
  }

  /**
   * Resets the shake parameters to their default values.
   */
  private void resetShakeParameters() {
    mFirstDirectionChangeTime = 0;
    mDirectionChangeCount = 0;
    mLastDirectionChangeTime = 0;
    lastX = 0;
    lastY = 0;
    lastZ = 0;
  }

  @Override
  public void onAccuracyChanged(Sensor sensor, int accuracy) {
  }

}
```

After create that class, create main class that will implement the shake detection. I named it ShakeActivity.

Code For ShakeActivity :

```
private SensorManager mSensorManager;

private ShakeEventListener mSensorListener;
```

```
import com.phonegap.DroidGap;

import android.os.Bundle;


public class ShakeActivity extends DroidGap {

@Override

protected void onResume() {

   super.onResume();

   mSensorManager.registerListener(mSensorListener,

       mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),

       SensorManager.SENSOR_DELAY_UI);

 }


 @Override

 protected void onStop() {

   mSensorManager.unregisterListener(mSensorListener);

   super.onStop();

 }


   @Override

public void onCreate(Bundle savedInstanceState) {

      super.onCreate(savedInstanceState);

     mSensorListener = new ShakeEventListener();

     mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

   mSensorManager.registerListener(mSensorListener,

       mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
```

```
      SensorManager.SENSOR_DELAY_UI);



   mSensorListener.setOnShakeListener(new ShakeEventListener.OnShakeListener() {


     public void onShake() {

       Toast.makeText(KPBActivityImpl.this, "Shake!", Toast.LENGTH_SHORT).show();

     }

   });

}
```

Crop Image

Next step is the ability to crop the image. We can use canvas to place our new image. We use matrix and Bitmap to resize our image. Also we need path to set position of our image. It is very important to set position of our image because we will change image according to the shake and we need to make simple animation for that image. I make my own function to handle this problem.

Code :

```
public void addGbr(ImageView gbr,float posY,Bitmap bitGbr1,Bitmap bitGbr2){

                try{

                   Paint paint = new Paint();

                   paint.setFilterBitmap(true);



                   RectF rectf = new RectF(0,0,100,66);

                   RectF rectf2 = new RectF(0,0,100,100);



                   Canvas canvas = new Canvas(targetBitmap);
```

```
                Path path = new Path();
                Path path2 = new Path();
                path.addRect(rectf, Path.Direction.CW);
                path2.addRect(rectf2, Path.Direction.CW);

                canvas.clipPath(path);
                canvas.drawBitmap( bitGbr1, 0f,67-posY, paint);//ngatur posisi supaya nempel dgn apel
                canvas.clipPath(path2);
                canvas.drawBitmap( bitGbr2, 0f, -posY, paint);

                Matrix matrix = new Matrix();
                matrix.postScale(1f, 1f);
                Bitmap resizedBitmap = Bitmap.createBitmap(targetBitmap, 0, 0, 100, 100, matrix, true);

                /*convert Bitmap to resource */
                BitmapDrawable bd = new BitmapDrawable(resizedBitmap);
                gbr.setBackgroundDrawable(bd);
             }
             catch(Exception e){
                System.out.println("Error1 : " + e.getMessage() + e.toString());
            }
  }
```

The code that I use is using value that suitable to my application like 66f value. So, that is not exact value for another application.

Manipulate image

Manipulate image that I mean here is the ability to change one image to another image and the ability to change the image that is being shake.

Code :

```
private Runnable myRunnable = new Runnable() {



        @Override

        public void run() {

                if(flag<=3){


    if(flag==2){gbrUbah=gbr2;pnh1.setBackgroundColor(color.black);pnh2.setBackgroundResource(
R.drawable.arrowdown);}

                        else
if(flag==3){gbrUbah=gbr3;pnh2.setBackgroundColor(color.black);pnh3.setBackgroundResource(R.drawa
ble.arrowdown);}

                        if(bykGetar<0)bykGetar=0;


                        if(_letakY<=0){

                                mSensorListener.penanda=flag;

                                bykGetar--;

                                _letakY=66f;

                                if(q==3)q=0;

                                else q++;

                        }

                        if(q==3){

                                addGbr(gbrUbah,_letakY,bitAllGbr[3],bitAllGbr[0]);

                        }

                        else{
```

```
                                        addGbr(gbrUbah,_letakY,bitAllGbr[q],bitAllGbr[q+1]);


                                    }

                                    //set max, ga bs berenti

                                    int speed=6000/((bykGetar)+8);

                                    if(speed<100)speed=100;

                                    Shake2Activity.this.myHandler.postDelayed(myRunnable,speed );

                                        if(bykGetar>0){_letakY-=10;mSensorListener.penanda=0;}



                                    //}

                                        Log.d("bykGetar",bykGetar+", penanda
"+mSensorListener.penanda + ", flag "+ mSensorListener.f);


            //Toast.makeText(Shake2Activity.this,""+bykGetar,Toast.LENGTH_SHORT).show();



                                }

                        }

            };
```

OnShake action

To control the image movement according to shaking, I use thread to read the shake value first, and with that shake value, move the image and change to another if the image already shaken.

Code :

```
myHandler = new Handler();



    mSensorListener.setOnShakeListener(new ShakeEventListener.OnShakeListener() {
```

```
        public void onShake() {

          //Toast.makeText(Shake2Activity.this, mSensorListener.nilai+"", Toast.LENGTH_SHORT).show();

          bykGetar=mSensorListener.nilai*3;

          flag=mSensorListener.f;

          myHandler.postDelayed(myRunnable, 1000);

          //flag++;

        }

      });
```

Shake detection modification

ShakeEventListener that I use is a bit different than shakeEventListener that I explain before. This is because I need simple change and some return value form the class.

Code for new ShakeEventListener :

```
package z.z;


import android.hardware.Sensor;

import android.hardware.SensorEvent;

import android.hardware.SensorEventListener;

import android.hardware.SensorManager;

import android.util.Log;



/**

 * Listener that detects shake gesture.

 */

public class ShakeEventListener implements SensorEventListener {
```

```
public int nilai=0,f=0,penanda=1;
 /** Minimum movement force to consider. */
 private static final int MIN_FORCE = 5;

 /**
  * Minimum times in a shake gesture that the direction of movement needs to
  * change.
  */
 private static final int MIN_DIRECTION_CHANGE = 3;

 /** Maximum pause between movements. */
 private static final int MAX_PAUSE_BETHWEEN_DIRECTION_CHANGE = 400;

 /** Maximum allowed time for shake gesture. */
 private static final int MAX_TOTAL_DURATION_OF_SHAKE = 400;

 /** Time when the gesture started. */
 private long mFirstDirectionChangeTime = 0;

 /** Time when the last movement started. */
 private long mLastDirectionChangeTime;

 /** How many movements are considered so far. */
 private int mDirectionChangeCount = 0;
```

```
/** The last x position. */
private float lastX = 0;

/** The last y position. */
private float lastY = 0;

/** The last z position. */
private float lastZ = 0;

/** OnShakeListener that is called when shake is detected. */
private OnShakeListener mShakeListener;

/**
 * Interface for shake gesture.
 */
public interface OnShakeListener {

  /**
   * Called when shake gesture is detected.
   */
  void onShake();
}

public void setOnShakeListener(OnShakeListener listener) {
```

```
  mShakeListener = listener;
}

@Override
public void onSensorChanged(SensorEvent se) {
  // get sensor data
  float x = se.values[SensorManager.DATA_X];
  float y = se.values[SensorManager.DATA_Y];
  float z = se.values[SensorManager.DATA_Z];

  // calculate movement
  float totalMovement = Math.abs(y + z - lastY - lastZ);

  if (totalMovement > MIN_FORCE) {

    // get time
    long now = System.currentTimeMillis();

    // store first movement time
    if (mFirstDirectionChangeTime == 0) {
      mFirstDirectionChangeTime = now;
      mLastDirectionChangeTime = now;
    }

    // check if the last movement was not long ago
```

```
long lastChangeWasAgo = now - mLastDirectionChangeTime;
if (lastChangeWasAgo < MAX_PAUSE_BETHWEEN_DIRECTION_CHANGE) {

 // store movement data
 mLastDirectionChangeTime = now;
 mDirectionChangeCount++;

 // store last sensor data
 lastX = x;
 lastY = y;
 lastZ = z;

 // check how many movements are so far
 if (mDirectionChangeCount >= MIN_DIRECTION_CHANGE) {

  // check total duration
  long totalDuration = now - mFirstDirectionChangeTime;
  if (totalDuration < MAX_TOTAL_DURATION_OF_SHAKE) {
      nilai++;
   mShakeListener.onShake();
   resetShakeParameters();
  }
 }

} else {
```

```
    resetShakeParameters();nilai=0;
    //Log.d("tanda",""+penanda);
    if(penanda>0)
    f++;
   }
  }
}

/**
 * Resets the shake parameters to their default values.
 */
private void resetShakeParameters() {
  mFirstDirectionChangeTime = 0;
  mDirectionChangeCount = 0;
  mLastDirectionChangeTime = 0;
  lastX = 0;
  lastY = 0;
  lastZ = 0;

}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
}
```

```
}
```

Reference

http://stackoverflow.com/questions/2317428/android-i-want-to-shake-it

http://developer.android.com/

http://en.wikipedia.org/wiki/Android_%28operating_system%29