ISEL / ADEETC	Master in Communication Networks and Multimedia Engineering **Interactive Multimedia Applications**

Tutorial 3 Interactive Multimedia Applications

Universe Explorer

Rui Jesus

Introduction

This work aims to introduce and explore the application development environment for Android devices namely the XML editor to build the user interface. The tutorial starts from the "Hello World" application that is changed in order to build a new application with a more elaborate user interface. Below is a link that you should consult during the development of this work.

Android Developers

http://developer.android.com/training/index.html

Note1: the first part of this tutorial (points: 1-25) should be done in class and the code of the resulting Android projects must be delivered through the Moodle platform until the 25th of March.

Note2: the second part of this tutorial (points: 26-36) should be done in class and the code of the resulting Android projects must be delivered through the Moodle platform until the 2nd of April.

Laboratory Work

Universe Explorer

1. Create a new project in **Android Studio** by repeating the steps you took in Tutorial 1.

2. Create a new class within the same package. Give the name **"WorldGen"** to the class and create the methods and attributes as shown in Figure 1. The **"WorldGen"** class will be used to generate a new world/planet consisting of several attributes and methods that describe the characteristics of a planet.

3. Run the program and check the result that should be the same as the “Hello World” application.

```
package com.example.hello_world_t1;
public class WorldGen {
        String planetName = "Earth";
        int planetMass;
        double planetGravity;
        int planetColonies;
        long planetPopulation;
        int planetBases;
        int planetMilitary;
        boolean planetProtection;
        public WorldGen (String name, int mass, double gravity) {
                planetName = name;
                planetMass = mass;
                planetGravity = gravity;
                planetColonies = 0;
                planetPopulation = 0;
                planetBases = 0;
                planetMilitary = 0;
                planetProtection = false;
        }
        void setPlanetColonies (int numColonies) {
                // incrementar planetColonies de numColonies unidades
        }
        int getPlanetColonies() {
                // retorna  planetColonies;
        }
        void setPlanetMilitary (int numBases) {
                // incrementa planetBases de numBases unidades
        }
        int getPlanetMilitary() {
                // retorna planetBases
        }
        void turnForceFieldOn() {
                // coloca planetProtection a true
        }
        void turnForceFieldOff() {
                // coloca planetProtection a false
        }
        boolean getForceFieldState() {
                // retorna o valor planetProtection
        }
        void setColonyImmigration (int numColonists) {
                // incrementa planetPopulation de numColonists unidades
        }
        long getColonyImmigration() {
                // retorna planetPopulation
        }
        void setBaseProtection (int numForces) {
                // incrementa planetMilitary de numForces unidades
        }
        int getBaseProtection() {
                // retorna planetMilitary
        }
}
```

4. At the end of the **onCreate()** method of the "**MainActivity**" class create an object of class "**WorldGen**", for instance: "`static WorldGen earth = new WorldGen("Earth", 5973, 9.78);`" Put the line of code, "`earth.setPlanetColonies(1);`" below to delete the "warning".

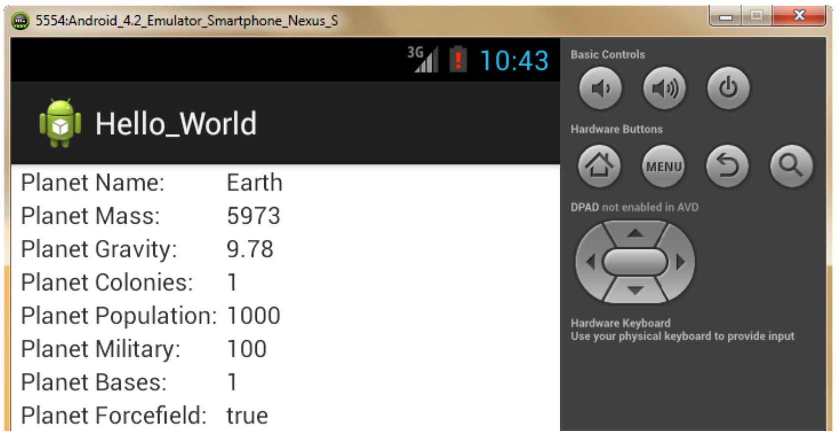
5. Following the previous point, initialize the remaining attributes of the "`earth`" object. For instance, "`planetColonies = 1; planetPopulation = 1000; planetBases = 1; planetMilitary = 100; planetProtection = true;`".

Figure 1. "WorldGen" class.

6. Create the "`protected void setStartUpWorldValues()`" method in the "**MainActivity**" class to initialize the attributes of a new planet. Use this method instead of the lines of code that you performed in the previous point.

7. Run the program to check for errors.

8. The next step is to build the layout shown in Figure 2. Begin by creating the necessary strings in the "*strings.xml*" file.



Planet Name:	Earth
Planet Mass:	5973
Planet Gravity:	9.78
Planet Colonies:	1
Planet Population:	1000
Planet Military:	100
Planet Bases:	1
Planet Forcefield:	true

Figure 2. Main Layout of the Application.

9. Use the strings from the previous point in the file "*content_main.xml*". Use the **TextView** tag (or graphic editor) to display the necessary strings in the layout, for instance,

```
 <TextView
        android:id="@+id/textView1"
        android:layout_marginLeft="36dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/planet_name_label"
 />
<TextView
        android:id="@+id/textView2"
        android:layout_marginLeft="36dp"
        android:layout_below="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/planet_mass_label"
/>
```

Run the program. Do not forget to remove the "Hello World" string.

10. To write the value of each attribute on the screen, create another method in the "**MainActivity**" class, "`private void setStartUpScreenText(…)`". To put the value of the attributes on the screen use the following code:

```
TextView planetNameValue = (TextView) this.findViewById(R.id.dataView1);
planetNameValue.setText(earth.planetName);
TextView planetMassValue = (TextView) this.findViewById(R.id.dataView2);
planetMassValue.setText(String.valueOf(earth.planetMass));
```

11. Run de program. Do not forget that the "`setStartUpWorldValues()`" and "`setStartUpScreenText()`" methods of class "**MainActivity**" should be used inside of the **onCreate()** method.

12. Search the Web for an image of the universe with a resolution of approximately 2400x1440 pixels, to be used as the background image. In an image editing program, generate the remaining images for each generalized density group.

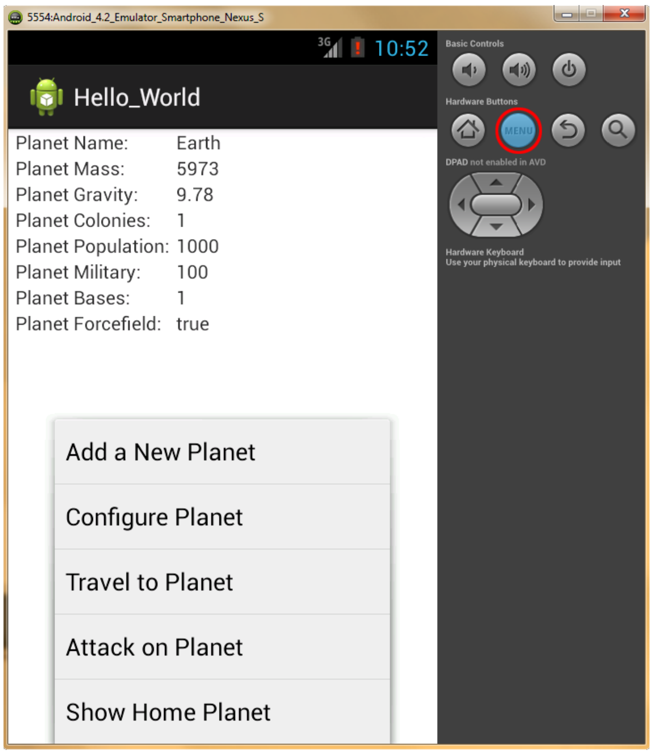
13. Do as in tutorial 2 and put the previous images in their **drawable** directories. Place the image as the background image of the "**MainActivity**".

14. Now let's create a menu (see Figure 4). Copy to the file "*menu_main.xml*" of the menu directory, the code in Figure 3. Run the program. Check that the menu is correct (see Figure 4).

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.example.hello_world_t1.MainActivity" >

    <item android:id="@+id/menu_add"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="@string/menu_add_planet"/>
    <item android:id="@+id/menu_config"
        android:orderInCategory="200"
        android:showAsAction="never"
        android:title="@string/menu_config_planet"/>
    <item android:id="@+id/menu_travel"
        android:orderInCategory="300"
        android:showAsAction="never"
        android:title="@string/menu_travel_planet"/>
    <item android:id="@+id/menu_attack"
        android:orderInCategory="400"
        android:showAsAction="never"
        android:title="@string/menu_attack_planet"/>
    <item android:id="@+id/menu_contact"
        android:orderInCategory="500"
        android:showAsAction="never"
        android:title="@string/menu_home_planet"/>
</menu>
```

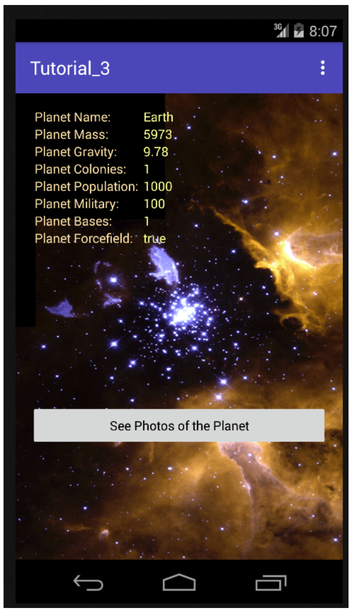
Figure 3. Code for the “*menu_main.xml*” file.



Planet Name:	Earth
Planet Mass:	5973
Planet Gravity:	9.78
Planet Colonies:	1
Planet Population:	1000
Planet Military:	100
Planet Bases:	1
Planet Forcefield:	true

Figure 4. Menu Layout.

15. In the layout "*content_main.xml*" put a "**button**" element below, after the planet information. In the "text" attribute of the "**button**" element place a string (defined in the file "*strings.xml*") with the following text: "Planet Photos" (see Figure 5).



Planet Name:	Earth
Planet Mass:	5973
Planet Gravity:	9.78
Planet Colonies:	1
Planet Population:	1000
Planet Military:	100
Planet Bases:	1
Planet Forcefield:	true

Figure 5. Layout “content_main.xml”.

16. To view images of a planet we will use another activity. Proceed by enabling the **Onclick** event of the "**button**" element created the previous point. In the **Onclick** method (...) put the following code:

```
Intent intent_add = new Intent(this, GridViewPhotos.class);
this.startActivity(intent_add);
```

17. Create the "**GridViewPhotos**" class with the code below.

```
public class GridViewPhotos extends AppCompatActivity {
    private GridView gridView;
    private GridViewAdapter gridAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_gridph);

      }
}
```

18. In order to create the activity "**GridViewPhotos**" it is necessary to create the "*activity_gridph.xml*" layout (for now, it can be left blank). Comment the line of code with the variable "gridAdapter". Run the program and check what happens when you click on **button "Planet Photos…"**.

19. To launch a new activity, add the new activity after the activity "**MainActivity**" in the file "*AndroidManifest.xml*".

```
<activity android:name=".GridViewPhotos"/>
```

20. Run the program and verify that the new activity is blank.

21. The next step consist of building the "*activity_gridph.xml*" layout (see Figure 6). Start by putting a <GridView> element inside the <RelativeLayout> element. For this activity you need another background image so repeat points 12 and 13. Change the value of the following attributes of the <GridView> element:

    android:layout_width - "match_parent"

    android:layout_height - "wrap_content"

android:layout_centerHorizontal - “true"

android:gravity - “center"

android:stretchMode -“columnWidth"

android:drawSelectorOnTop -“true"

android:focusable - “true"

android:clickable - “true"

android:columnWidth - “100dp"

android:numColumns - “auto_fit"

android:verticalSpacing - “5dp"

android:layout_marginLeft - “5dp"

android:layout_marginRight - “5dp"

android:layout_marginTop - “5dp"

android:layout_marginBottom - “40dp”

Tip: Use the visual editor and try to understand the effect of each attribute.

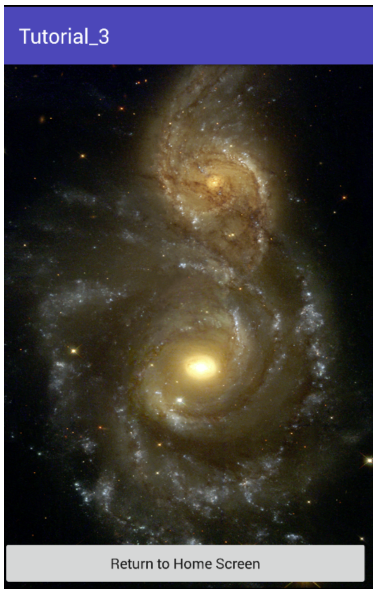


Figure 6. Layout “*activity_gridph.xml*”.

22. To complete the layout of Figure 6 it is necessary to put the "**button**" below. Run the program.

23. Register the **OnClick**() event and in the method that responds to the event place *finish()*; Run the program.

```
package com.example.ruimfjesus.gridview_photos;

import android.graphics.Bitmap;


public class ImageItem {
    private Bitmap image;
    private String title;

    public ImageItem(Bitmap image, String title) {
        super();
        this.image = image;
        this.title = title;
    }

    public Bitmap getImage() {
        // retorna image;
    }

    public void setImage(Bitmap image) {
        // Atribui ao atributo image o valor image
passado como parâmetro
    }

    public String getTitle() {
        // retorna title;
    }

    public void setTitle(String title) {
        // Atribui ao atributo title o valor title
passado como parâmetro

    }
}
```

24. At each position of the <GridView> element we will place an image with text underneath, so let's create an "**ImageItem"** class as described in the box below.

25. Search the Web for 9 images of the planet Earth (for example, images of space in which you see the earth). Each image must be at least 400x400 pixels. Repeat steps 12 and 13. It should be named "image_1.png", "image_1.png", ..., "image_9.png" to the image files.

```
<string-array name="image_ids">
    <item>@drawable/image_1</item>
    <item>@drawable/image_2</item>
      …
    <item>@drawable/image_9</item>
    <item>@drawable/image_1</item>
      …
    <item>@drawable/image_9</item>
</string-array>
```

26. Proceed by creating in XML an array of strings in the file "*strings.xml*" as the "paths" of

the images (see the box below).

27. Add the *getData (...)* method to the "**GridViewPhotos**" class to read from the file the images to memory.

```
private ArrayList<ImageItem> getData() {
    final ArrayList<ImageItem> imageItems = new ArrayList<>();
    TypedArray imgs =
getResources().obtainTypedArray(R.array.image_jds);
    for (int i = 0; i < imgs.length(); i++) {
        Bitmap bitmap =
BitmapFactory.decodeResource(getResources(), imgs.getResourceId(i,
-1));
        imageItems.add(new ImageItem(bitmap, "Image#" + i));
    }
    return imageItems;
}
```

28. The following step consist of creating the layout "*grid_item_layout*" (will be used to format the image and text) for each element of the GridView container. Use the <LinearLayout>, <ImageView> and <TextView> elements. The dimensions of the <LinearLayout> element should be "wrap_content", the <ImageView> element should be 100dp x 100dp, and the text should be 12sp. Figure 7 illustrates the layout.

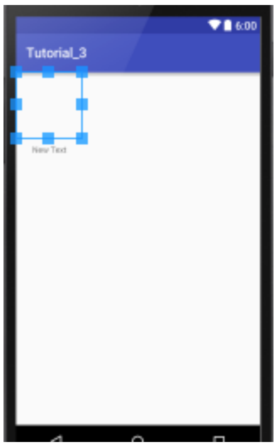


Figure 7. Layout "*grid_item_lauout.xml*".

29. As mentioned in the classes to use the <GridView> container, you must use an "Adapter" object to connect to the data to be placed in <GridView>. Thus, proceed by inserting the line you had put in point 18 in comment. And in the "**GridViewPhotos**"

activity in the **OnCreate** method (...) place the lines of code in the box below.

```
gridView = (GridView) findViewById(R.id.gridViewph);
gridAdapter = new GridViewAdapter(this, R.layout.grid_item_layout,
getData());
gridView.setAdapter(gridAdapter);
```

30. Create the **GridViewAdapter** class where the adapter for this <GridView> container example will be implemented. Copy the code in the box below. Do not forget to confirm the **ID's**.

```
public class GridViewAdapter extends ArrayAdapter {
    private Context context;
    private int layoutResourceId;
    private ArrayList<ImageItem> data = new ArrayList<ImageItem>();

    public GridViewAdapter(Context context, int layoutResourceId,
ArrayList<ImageItem> data) {
        super(context, layoutResourceId, data);
        this.layoutResourceId = layoutResourceId;
        this.context = context;
        this.data = data;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup
parent) {
        View row = convertView;
        ViewHolder holder;

        if (row == null) {
            LayoutInflater inflater = ((Activity)
context).getLayoutInflater();
            row = inflater.inflate(layoutResourceId, parent, false);
            holder = new ViewHolder();
            holder.imageTitle = (TextView)
row.findViewById(R.id.text);
            holder.image = (ImageView) row.findViewById(R.id.image);
            row.setTag(holder);
        } else {
            holder = (ViewHolder) row.getTag();
        }

        ImageItem item = data.get(position);
        holder.imageTitle.setText(item.getTitle());
        holder.image.setImageBitmap(item.getImage());
        return row;
    }

    static class ViewHolder {
        TextView imageTitle;
        ImageView image;
    }

}
```

31. Run the program and verify that it works as expected (see Figure 8).

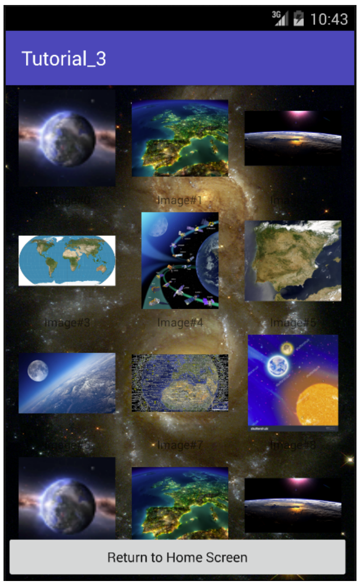


Figure 8. Layout of the activity "**GridViewPhotos**".

32. When the user clicks/touches in an image (Figure 8) on the screen, the application should create another activity to show a preview of the image (see Figure 9). Activate the event in the **GridViewPhotos** class, according to the box below.

```
gridView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View v, int
position, long id) {
        ImageItem item = (ImageItem)
parent.getItemAtPosition(position);

        Intent intent = new Intent(GridViewPhotos.this,
PreviewActivity.class);
        intent.putExtra("title", item.getTitle());
        intent.putExtra("position", position);
        //intent.putExtra("image", item.getImage());

        startActivity(intent);
    }
});
```

33. Create the activity "**PreviewActivity**" (see Figure 9). In the **onCreate()** method include code to fetch the data passed through the "**Intent**" ("title" and "position"). Use the "position" variable to access the array of images.

34. Create the "preview_activity" layout (see Figure 9). Use a <FrameLayout> element with an <ImageView> and <TextView>. Use the "gravity" attribute to organize the two previous contents in <FrameLayout>. In the **onCreate**() method of activity "**PreviewActivity**" place this layout. Put in the <ImageView> and <TextView> the respective image and text (“title").

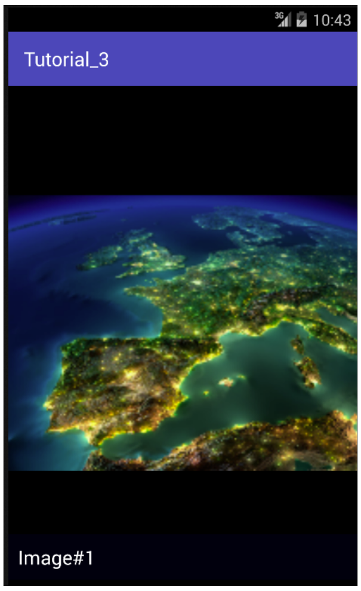


Figura 9. Layout of the “**PreviewActivity**” activity.

35. When the user clicks on the image preview, the application should return to the previous screen (see Figure 8). Make the respective code. Run the program.

36. To give feedback to the user about the functionality of the previous point, place a **Toast** in the **onCreate**() method of the activity "**PreviewActivity**".

```
CharSequence str = "Touch on the screen to return back";
 Toast toast = Toast.makeText(PreviewActivity.this,str,Toast.LENGTH_SHORT);
toast.show();
```

37. For each menu option (Point 14), we will create a different layout through the XML editor/Visual editor and a different activity. Create a new class within the same package. Name the class "**NewPlanet**".

38. Copy the code from the box below to the "**NewPlanet**" class file. For now, the class is composed only by the **OnCreate** method. In this method, it is indicated the layout of

this activity, "`setContentView(R.layout.activity_add);`". This Activity will be used to create a new planet.

```java
package com.example.hello_world_t1;

import android.app.Activity;
import android.os.Bundle;

public class NewPlanet extends Activity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_add);
	}
}
```

39. Select the project folder, "res/layout" in the project panel, right-click with mouse over the folder and select **New -> Layout resource file**. Next, add the file name (*activity_add*), select the "RelativeLayout" layout type and click the **OK**.

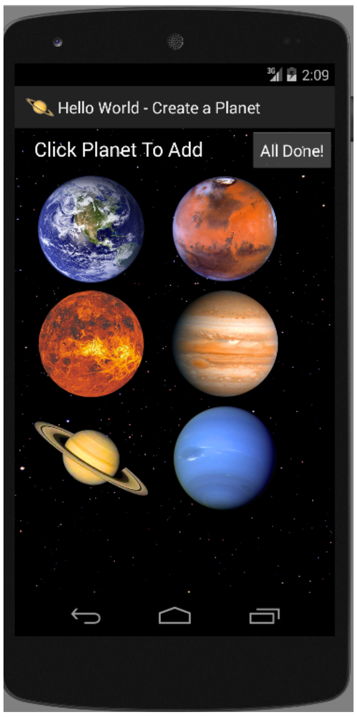


Figure 10. Layout of the "**NewPlanet**" activity.

40. Search the Web for one image of 6 different planets (see Figure 10). Each image must be at least 400x400pixels. For the background image use the same image as for point 21. Repeat for all images the points 12 and 13. Images of the planets should occupy 0.6 inches of physical space on the display in all generalized groups.

41. Build the layout shown in Figure 10. Do not forget the best practices used in previous points or previous tutorials.

42. To launch the second activity you must first change the **onOptionsItemSelected()** method of the "**MainActivity**" class and then use an "**Intent**" object. Change the code of the **onOptionsItemSelected()** method of the "**MainActivity**" class according to the code in the box below.

```
public boolean onOptionsItemSelected(MenuItem item) {
	switch(item.getItemId()) {
		case R.id.menu_add:
			Intent intent_add = new Intent(this, NewPlanet.class);
			this.startActivity(intent_add);
			break;
		case R.id.menu_config:

			break;
		case R.id.menu_travel:

			break;
		case R.id.menu_attack:

			break;
		case R.id.menu_contact:

			break;
		default:
			return super.onOptionsItemSelected(item);
	}
	return true;
}
```

43. When the user selects the "Add a New Planet" menu option, the "**NewPlanet**" activity will be started and the layout of figure 10 will be presented to the user. Execute the program and check the result. Do not forget to add the activity in the file "**AndroidManifest.xml**".

44. When the user clicks the **button**, "All Done!" in figure 10, you must end the activity and return to the "**MainActivity**". Make the code necessary to fulfill this functionality. Run the program.

45. For the menu option "Configure Planet" we will create a new activity. Repeat the steps to create the "**NewPlanet**" activity, now for the "**ConfigPlanet**" class.

46. To create the layout (XML file), do as in step 39. Add the file name (activity_config), select the "LinearLayout" layout type and click the **OK** button.

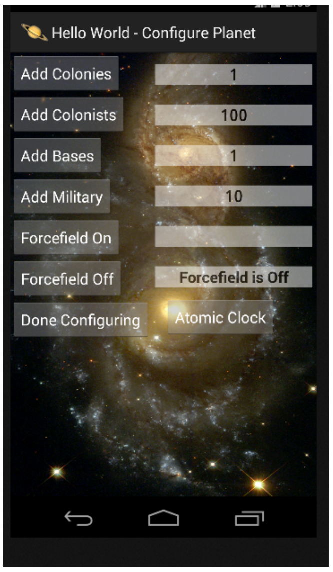


Figure 11. Layout of the "**ConfigPlanet**" activity.

47. Open the file "*activity_config.xml*" and select the tab related to **Design**. Drag "**Button**" elements and "**TextView**" elements into the layout to construct the layout represented in Figure 11. The first six elements on the right-hand side of Figure 11 should be editable "TextView" (EditText). They should appear with the information that is in the "earth" object.

48. When you change the editable values of the layout shown in Figure 11, the respective attributes of the "earth" object must be changed. Implement the code needed to add this functionality (for instance, use a singleton object to communicate between activities). Run the program.

49. The button "Done Configuring" of Figure 11 when pressed allows to end the activity. The "Atomic Clock" button for now has no effect. We will return to this button later.

50. Create a new activity named "**TravelPlanet**" (repeat the steps you used to create the previous activities). Do not forget to create the file "*activity_travel.xml*". Choose the

"**FrameLayout**" container. Drag a **VideoView** element and a **button** element into the layout.

51. The "**TravelPlanet**" activity is initiated when the user selects the "Travel to Planet" option from the "Options" menu. The "**button**" serves to terminate the activity. Add the required code for these features.

52. Create the fourth and last activity named "**AttackPlanet**." The name of the XML file is "*activity_attack.xml*" and use a "LinearLayout" (horizontal) container.

53. Drag the **ImageButton** and **TextView** elements layout to create the layout represented in Figure 12. Arrange the **ImageButtons** in a "LinearLayout" (vertical) container. Also place the **TextViews** in a different "LinearLayout" (vertical) container.

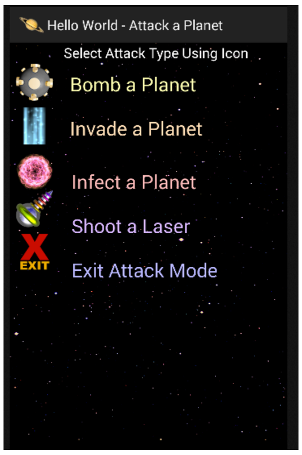


Figure 12. Layout of the "**AttackPlanet**" activity.

54. Browse the Web for images similar to those in Figure 12 to place in the "**ImageButton**" elements. These images should have a density of 48dp. Do not forget to generate the images with the resolutions for the various generalized groups of density.

55. The "**ImageButton**" "exit" (Figure 12) is used to terminate the activity. Implement the code and run the program.

56. The "**AttackPlanet**" activity is initiated when the user chooses the "Attack on a Planet" option in the "Options" menu.

57. With the exception of the "**ImageButton**" relative to the "exit" option in Figure 12, make the "**ImageButtons**" into buttons with states (see Figure 13). Start by editing/ creating the images in an image-editing program. Then create the XML files and change the "scr" attribute of the respective “**ImageButtons**".

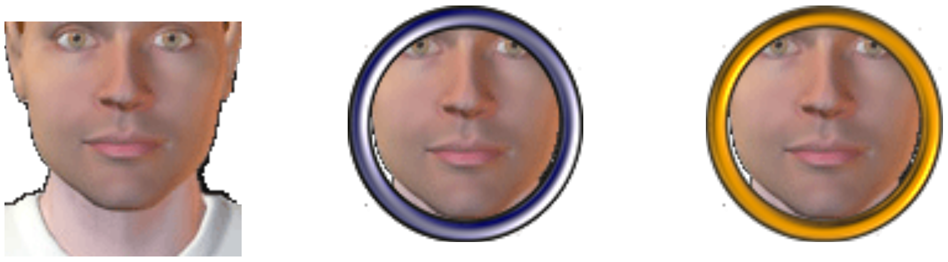


Figure 13. **“ImageButton”** with three states.

58. Create a "Drawable Animation" for the "**ImageButton**" for the "infect a planet" option in Figure 12. Start by creating the images and then the XML file.

59. Create two "View Animation" for the "**ImageButtons**", "bomb a planet" and "shoot a laser" of Figure 12. The first "**ImageButton**" must perform a **rotation** and the second a **translation**.

60. In the activity "**TravelPlanet**" make the code for the reproduction of the video provided by the teacher when the activity is created.

61. In the activity "**NewPlanet**" implement the code to reproduce a sound when the button for the planet Mars is pressed. Use the audio file, "mars.m4a" provided by the teacher. Use a "**MediaPlayer**" object.

62. With the exception of the "**ImageButton**" relative to the "exit" option in Figure 12, make the code so that each time an **ImageButton** is pressed one of the sounds provided by the teacher is played (files "blast.ogg", "transport.m4a" , "Virus.m4a" and "laser.m4a"). Use a "SoundPool" object.

63. Pressing the "Atomic Clock" button of the activity **ConfigPlanet** creates the activity **TimePlanet**. Make the code necessary to create the activity.

64. Figure 14 shows the "*activity_main.xml*" layout of activity **TimePlanet**. Build the layout and declare the activity in the "AndroidManifest.xml" file.

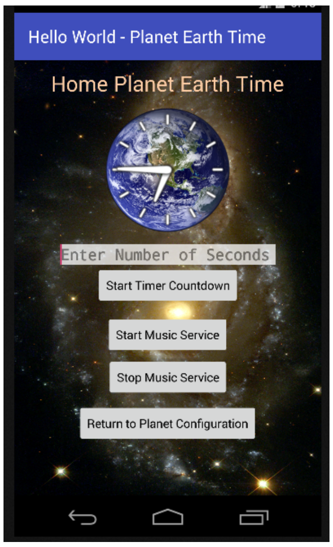


Figure 14. Layout of the "**TimePlanet**" activity.

65. The activity **Time Planet** should return a result when the user presses the "Return to Planet Configuration" button to the initiating activity (**ConfigPlanet**). Use the code below.

```
Intent returnIntent = new Intent();
setResult(RESULT_OK, returnIntent);
finish();
```

66. To start the **TimePlanet** activity use the code below. Run the program.

```
Intent callTimeIntent = new Intent(view.getContext(), TimePlanet.class);
startActivityForResult(callTimeIntent, 0);
```

67. Create the "**AlarmReceiver**" class to implement a **BroadcastReceiver** with the code below. The goal is to receive the message sent by the user-created alarm by pressing the "Start Timer Countdown" button (Figure 14).

```
public class AlarmReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context arg0, Intent arg1) {
                Toast.makeText(arg0, "ALARM NOTIFICATION",
Toast.LENGTH_SHORT).show();
        }
}
```

68. Make the code to receive the **onClick** event of the "Start Timer Countdown" button (Figure 14). When the user presses this button, an alarm must be issued after the number of seconds entered by the user in **EditText** has elapsed. Use the code below. Run the program.

```
public void startTimer(View view) {
    EditText alarmText = (EditText) findViewById(R.id.setAlarm);
    assert alarmText != null : "startTimer (TimePlanet): alarmText";
    int i = 0;
    if (!(alarmText.getText().toString().isEmpty())) {
        i = Integer.parseInt(alarmText.getText().toString());
    }
    if (i == 0 ) {
        Toast.makeText(this, "Insert a positive integer different from zero to 
set the alarm", Toast.LENGTH_LONG).show();
    }
    else {
        Intent intent = new Intent(this, AlarmReceiver.class);

        PendingIntent alarmIntent = 
PendingIntent.getBroadcast(this.getApplicationContext(), 234324243, intent, 0);
        AlarmManager alarmManager = (AlarmManager) 
getSystemService(ALARM_SERVICE);
        alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() + 
(i * 1000), alarmIntent);
        Toast.makeText(this, "Alarm set in " + i + " seconds", 
Toast.LENGTH_LONG).show();
    }
}
```

69. Create the "**MusicService**" class to implement a **Service** to start playing a background song.

```
public class MusicService extends Service {
       public static final String ServiceTag  = "MusicService";

       MediaPlayer musicPlayer;
       @Override
       public IBinder onBind(Intent arg0) {
              return null;
       }

       @Override
       public void onCreate() {
              Toast.makeText(this, "Music Service has been Created", 
Toast.LENGTH_SHORT).show();
              musicPlayer = MediaPlayer.create(this, R.raw.music);
              musicPlayer.setLooping(true);
       }

       @Override
       public int onStartCommand(Intent intent, int flags, int startid) {
              Toast.makeText(this, "Music Service is Started", 
Toast.LENGTH_SHORT).show();
              musicPlayer.start();
        return super.onStartCommand(intent, flags, startid);
       }

       @Override
       public void onDestroy() {
              Toast.makeText(this, "Music Service has been Stopped", 
Toast.LENGTH_LONG).show();
              musicPlayer.stop();
       }
}
```

70. When the user presses the "Start Music Service" **button** (figure 14) the Service (**MusicService**) must be started. Use the code below.

```
Intent intent = new Intent(TimePlanet.this, MusicService.class);
intent.addCategory(MusicService.ServiceTag);
startService(intent);
```

71. When the user presses the "Stop Music Service" **button** (figure 14), the Service (**MusicService**) must be stopped. Use the code below.

```
stopService(new Intent(TimePlanet.this, MusicService.class));
```

72. Lack destroy the **Service** when the app is destroyed. In the **onDestroy**() method of the activity **MainActivity** use the code below. Run the program. Do not forget to declare all app components in the "*AndroidManifest.xml*" file.

```
Intent intent = new Intent(getApplicationContext(), MusicService.class);
intent.addCategory(MusicService.ServiceTag);
 stopService(intent);
```