App Development TA-class 2: Layout

Emil Rose Høeg, Aalborg University - Copenhagen 15/04/2015

1 Activity Layout

We started out by talking about *relativeLayout*, which it the default layout. The advantages is that you can drag-and-drop easily in the design window, but it does not work well if you want to implement an array of buttons (e.g. root-menu).

You can e.g. make two dimensional array of imageButtons to serve as a root-menu of icons (as I showed in class). Pick the GridLayout from the palette [Layouts], and drag it onto the RelativeLayout (release when a red square appears). The size of the array can be defined in the [Properties] or in the text window as code seen in listing 1. If you want the grid of buttons to be centered in the parent (screen activity) add the following code (remember to Wrap Content in GridLayout):

Listing 1: Making a grid to fit in the center of the parent (screen activity).

```
<GridLayout
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:columnCount="3"
android:rowCount="3"
android:layout_centerInParent="true">
```

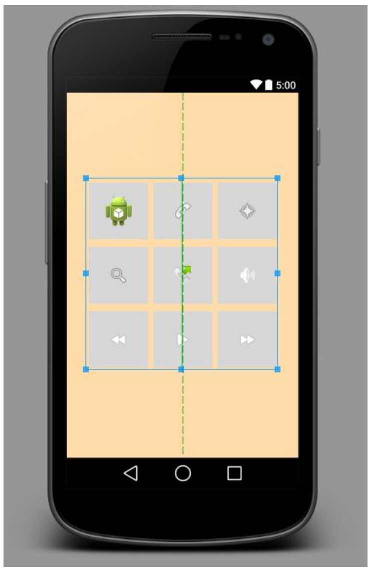

Figure 1: The Android Virtual Device (AVD) with 3x3 grid implemented.

To add an image to the ImageButton, navigate to *src* in the Properties. Click on [· · ·] to bring up resources, and choose an image from the drawable folder (imported or from the android library). See more info on [1].

HINT: Icon for ImageButton (src) can be quick accessed by double clicking on ImageButton

2 Custom buttons (Draw9Patch)

Custom buttons are a bit more comprehensive if you want to be sure to implement them sufficiently. With the Draw9Patch-file you can use a single button design for more purposes and sizes without fearing for deformations and weird artifacts. you can either change the .png in the drawable folder (refactor->rename) to *imagename*.9.png, directly in the folder, or use the draw9patch application tool included in the SDK. Bring up the windows run-function (*windows key* + R) and run: *draw9patch*. You can also access the tool in Android Studio by, by selecting the 9-patch tab when you have selected a .9.png image file from the [drawable] folder.

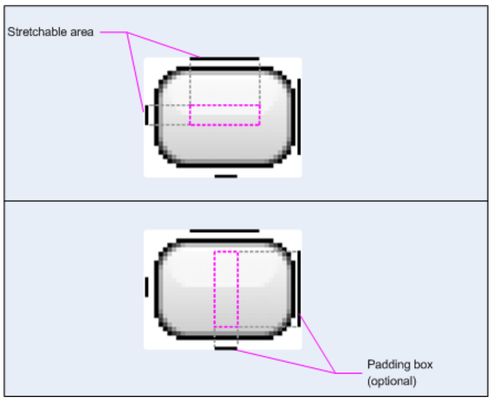

Figure 2: The fundamentals of the draw9patch file format.

Hint: You can find some nice documentation of 9patch files in [2].

The custom-made button is easily implementable into the activity, once the stretchable area and padding box has been defined. This is done by assigning the file to the Background-property, either by writing @drawable/*FILE_NAME (without .png)* or clicking on [· · ·] and locating it manually.

2.1 Applying button-animation (button pressed)

Besides the button, make some visual alteration to the .9.png button-file to simulate key pressed by giving immediate visual feedback. Make it darker, brighter or another color etc. Save the .9.png in the drawable-folder (make sure the two buttons match in stretching and padding). We have to make a selector, and we do that by right clicking on [Drawable], and make a New Drawable Resource File (XML)

Listing 2: newly created *Drawable Resource File* to contain the button-animation.

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">

<item android:state_pressed="true" android:drawable=@drawable/↩
    AndroidButton_pressed"/>
<item android:drawable="@drawable/AndroidButton"/>

</selector>
```

Instead of the .png-file assigned as the Background-property, we now have to assigned the *Drawable Resource File* instead. Simply assign the .xml file you created earlier (seen in listing 2).

Hint: Remember that the state_pressed has to come first, otherwise the compiler will not read it, and the boolean will remain false.

2.2 App-icon

If you want to change the icon of the app (i.e. the icon displayed in your phones apps-menu, navigate to the AndroidManifest.xml file under apps/manifests/AndroidManifest.XML. The standard icon (fig. **??**) is the ip_launcher.png defined in the line: *android:icon="@drawable/ic_launcher"*. To replace with your own icon, simply change the destination/file to the desired icon of your choosing (standard for phones is 48x48p).

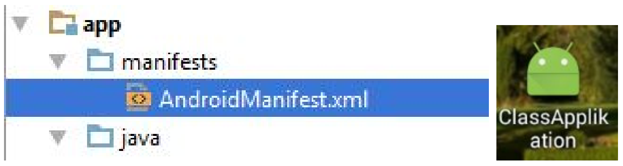

Figure 3: The location of the manifest.XML and the default app-icon

If you want to develop for more platforms, you need to take into account compatibility of the icon (a phone-icon will generally be smaller than e.g. tablet-icon). A nice tool the *Launcher Icon Generator* [3] enables you to upload an image-file (no matter what size), and will automatically convert it into different sized launcher-icons to be used in your application.

3 References

[1] http://developer.android.com/guide/topics/resources/drawable-resource.html

[2] http://developer.android.com/guide/topics/graphics/2d-graphics.html#nine-patch

[3] http://romannurik.github.io/AndroidAssetStudio/icons-launcher.html