A290/A590 – Meeting 8 Guide
Building a "Real App" — Part 2

Goal: We want continue work on our Android version of the Sudoku game as an example of building a more "real" application. We will try to get our "New Game" button working and then begin to layout the actual game grid.

The important thing to know is that we will be building some version of a grid that will be 9x9 (81 squares) and allow the user to "place" numerical values 1-9 in any of the open squares or tiles. Some will already have values present, which will be different every time you start a New Game.

I. Adding a Menu and Settings (continued).

One of the features most Android apps include is a menu that appears with you press the menu "button" on your particular Android device. There are menus that can also be activated by pressing and holding the screen, but we won't worry about that option at this time.

To get started, we will do things in a slightly different order this time, and create a group of string resources that we'll then be using in our menu. Here are the 7 that we want to create (following whatever naming convention you have chosen):

```
<string name="SettingsLabel">Settings&#8230;</string>
<string name="SettingsTitle">A290 First Real Application Settings</string>
<string name="SettingsShortcut">s</string>
<string name="MusicTitle">Music</string>
<string name="MusicDetail">Play Background Music</string>
<string name="HintsTitle">Hints</string>
<string name="HintsDetail">Show Hints During Play</string>
```

Note that we would like the string for SettingsLabel to display as "Settings…" but this will only happen, if at all, by using the actual XML markup for the "…" special character, ellipsis, which is …. Again, this may not display at all in the emulator.

Next, we need to create a menu we can use, so we need to add a "menu" file to the res/menu folder. We use the same process as we did when we created our layout file for About. Use File/New/AndroidXML/Menu (or equivalent) and name your file menu_first_real_app_settings. **[NOTE:** While there is already a "pre-built" menu file for main, we want to create ours from "scratch." When we are done, feel free to see how ours compares to the one that was already provided.**]**

The initial file will be very simple:

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


</menu>
```

To add a single menu item (for now) we need to name it, identify the title string, and, in this case, the shortcut string as well.

```
<item android:id="@+id/SettingsMenu"
        android:title="@string/SettingsLabel"
        android:alphabeticShortcut="@string/SettingsShortcut" />
```

You should be getting comfortable enough with this process to realize that our next step needs to be another modification of our main public class, to include this new item. This means adding some libraries and then, once again, overriding the main onCreate.

We should already have `import android.view.Menu;` and `import android.view.MenuItem;` but we need to add:

```
import android.view.MenuInflater;
```

to have access to these methods/libraries.

We need to be able to create the menu, so we will be doing another override. Remember, the “@Override” is there for debugging purposes.

If things have not been altered, you should actually find another version of what you need at the end of your FirstRealAppMainActivity.java file.

```
	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		// Inflate the menu; this adds items to the action bar if it is present.
		getMenuInflater().inflate(R.menu.first_real_app_main, menu);
		return true;
	}
```

We can adapt this for our use, by making it look like this:

```
	public boolean onCreateOptionsMenu(Menu menu) {
		super.onCreateOptionsMenu(menu);
		MenuInflater inflater = getMenuInflater();
		inflater.inflate(R.menu.menu_first_real_app_settings, menu);
		return true;
	}
```

Then we also need to modify the method for “onOptionsItemSelected” which will look like this when it is done:

```
	@Override
	public boolean onOptionsItemSelected (MenuItem item){
		switch (item.getItemId()) {
		case R.id.SettingsMenu:
			startActivity (new Intent(this, FirstRealAppPrefsActivity.class));
			return true;
/*We can add more items to this switch-case construct, if and when we need them*/
		}
		return false;
	}
```

Once we have this **[NOTE: You will have an error for FirstRealAppPrefsActivity.class, since we have not created it yet.]** we need to create all the resources for our Settings/Preferences.

We want to exploit the fact that Android is designed to make it easy to control your applications preferences. We just need two things: an XML file to set the preferences, and a new class to create things as needed.

To use these “built in” features of Android, we need a new directory in “res” to hold our settings.xml file for Preferences. Use File/New/AndroidResourceDirectory and choose "xml" for the "Resource Type" to create the folder named “xml.” **BE SURE it is created within the "res" folder.**

Then use File/New/XMLResourceFile/ and name the file “settings.xml”. It should be created in the “res/xml” folder you just created. Notice while the filename is "settings.xml," the XML element in the file is "PreferenceScreen" (shown in Component Tree) which is what we need.

Now we need to add content to our PreferenceScreen element in our settings.xml file. For now, we will add a couple of checkboxes. This is what your file should look like when you are done:

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

```
        android:title="@string/MusicTitle"
        android:summary="@string/MusicDetail"
        android:defaultValue="true" />

    <CheckBoxPreference
        android:key="hints"
        android:title="@string/HintsTitle"
        android:summary="@string/HintsDetail"
        android:defaultValue="true" />

</PreferenceScreen>
```

Nothing will display without a new “activity” to display it, so we need to create a new Class in our “src” folder and name it FirstRealAppPrefsActivity. Make sure that the new class is part of your package, which may not be set for you automatically.

Again it will be very simple to begin with. We will add two libraries:

```
import android.os.Bundle;
import android.preference.PreferenceActivity;
```

We will use another deprecated method here, primarily because I’m not quite ready for us to deal with the whole issue of “fragments.” I will try to get us there before the course is over, but I cannot guarantee it. You are, of course, both welcome to and encouraged to investigate them on your own. However, be aware that if you want or need “backward compatibility” with earlier versions of Android, you must use these deprecated methods. Whatever replaced this method will only “work” with those versions of Android released after this new method was introduced.

We simply need to add the details about our preferences to this activity by referring to our “settings.xml” file. It looks like this:

```
public class FirstRealAppPrefsActivity extends PreferenceActivity {
@Override
protected void onCreate(Bundle savedInstanceState){
       super.onCreate(savedInstanceState);
       addPreferencesFromResource(R.xml.settings);
       }
}
```

Remember that we have to **extend** this activity method. The deprecated component will still work and you can use “Quick Fix” to suppress the warnings, if needed. We will review how to do this in class if needed.

As we have already seen, if this new class/activity is not added to our “manifest” it will not be available, so we need to make this addition to AndroidManifest.xml

```
        <activity android:name=".FirstRealAppPrefsActivity"
            android:label="@string/SettingsTitle">

        </activity>
```

Notice I elected to use a relative reference to the location of the *.java file, instead of the absolute reference we used for the About activity.

Once you think you have all this complete go ahead and test this in the emulator, just to be sure we know how it looks. You have to click the “Menu” button in the emulator to reveal the “Settings…” banner at the bottom of the screen. You have to click on that banner to reveal our Preferences menu. Make sure yours works. I plan to make time to return to this screen/activity to show you how we can actually store these Preferences. For now, having this setup is what we want, so let’s move on.

II. Making the “New Game” button offer “levels of difficulty.”

We really need to do more than merely have our “New Game” button start a new game. If we want our app to be one people will keep using, we need to offer different levels of difficulty to keep it challenging for a wider audience. We will try to develop 3 levels of difficulty, Easy, Medium, and Hard, so our button will need to reveal those options first. Add 4 string resources to your strings.xml. Based on my naming convention, mine look like this:

```
<string name="NewGameTitle">Difficulty Level</string>
<string name="EasyLabel">Easy</string>
<string name="MediumLabel">Medium</string>
<string name="HardLabel">Hard</string>
```

Next, we will create a new file on “res/values” named “arrays.xml.” Be sure you create it as a “values” type of Android XML file. If you are successful, it will look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

</resources>
```

We need to add our particular array to this “values” file. We will name this array “difficulty” and then add 3 items to correspond to our three levels.

When you are done, it should look like this, contained in the `<resources></resources>` element:

```
    <array name="difficulty">
        <item>@string/EasyLabel</item>
        <item>@string/MediumLabel</item>
        <item>@string/HardLabel</item>
    </array>
```

Now we need to add another “case” to the onClick() method in our FirstRealAppMainActivity.java file. Remember, all the primary code is already there as is the “case” for our About button.

Because we are going to have a dialog for the “difficulty” level, the code is simpler and refers to a new method we will have to create:

```
            case R.id.btn_New:
                  openNewGameDialog();
                  break;
```

Before we get to our new method, we need to add several libraries because we also want to track the activities/states of the game. This means we need `import android.app.AlertDialog;` and `import android.content.DialogInterface;` and `import android.util.Log;`.

The new method will call “AlertDialog.” It will also call a new DialogInterface. We will also add some debugging info, very much like we did for MyFirstActivity. We will add the TAG and we will add code to show, in LogCat, when we start the game activity.

For the TAG, let’s use: `private static final String TAG = "StartMainActivity";`

The method for openNewGameDialog will look like this:

```
private void openNewGameDialog() {
      new AlertDialog.Builder(this)
            .setTitle(R.string.NewGameTitle)
/*If you have an icon you added, and you want to have it appear as part
 * of this dialog, add this next line*/
            .setIcon(R.drawable.ic_launcher)
```

```
                .setItems(R.array.difficulty,
                new android.content.DialogInterface.OnClickListener() {
                      public void onClick(DialogInterface dialoginterface,
                            int i) {
                      startGame(i);
                      }
                })
                .show();
}
```

The method to track our states using TAG will look like this:

```
private void startGame(int i){
      Log.d(TAG, "Clicked on " + i);
      /*Start game here ...*/
}
```

Let's test all this out. We'll just try to "Run" it first. If that is all OK, then try to "Debug" and see if our added code allows us to track what happens with New Game in LogCat.

Finally, let's make our Exit button do something. Using the same switch-case for onClick, we will add

```
case R.id.btn_ExitMy First:
      finish();
      break;
```

which actually terminate our activity/application and return all the resources.

We are ready to start creating the screens for our game grid, but first we need a quick primer on Android Graphics.

III. Introduction to Android Graphics: Getting Prepared to Create the Game Grid.

Today, I think we also need to spend time on the basics of Android graphics. While there is a whole native graphics library in android.graphics, we need to learn the real basics before we start applying anything to our game.

There are several classes we need to be aware of, including Color, Paint, Canvas, Path and Drawable. Let's look at each one.

Color: We've looked at this class briefly already. While we can create a color directly is some of our code, our preferred strategy will always be to create each one we need in our colors.xml resource file. We already have our background color defined there in our A290 First Real Project, as an example. One thing to note, we can access these colors in our .java files as well as in our layout and other XML files. To access a color in a .java file, you need to use

```
color = getResource().getColor(R.color.background);
```

would get the resource manager to lookup our color "background" and use it in the Project we will start now, if we needed it. We do not. Always keep in mind that your color definition includes four (4) hexadecimal pair values that represent Alpha, Red, Green, Blue, and each has a range of values from 0-255, or 0-ff.

Paint: The Paint class is important because we can use it to draw bitmaps, geometric shapes, and even text. It holds style, color, and related information. We can setup something like the following to paint in a solid, pre-defined color:

```
Paint cPaint;

cPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
cPaint.setColor(Color.LTGRAY);
cPaint.setStyle(Paint.Style.STROKE);
cPaint.setStrokeWidth(3);
```

What this does is set our color to Light Gray, make sure that whatever is drawn will have a solid border of width 3, and the ANTI_ALIAS_FLAG instructs the system to do the best it can to smooth out any diagonal lines.

We can also set a variety of options for text. We'll return to that when we work directly on our game.

Canvas: The Canvas class represents your drawing surface and it starts out blank. Various methods are available to draw lines, circles, rectangles and other shapes. It turns out that our Activity has a View (as we've seen sort of indirectly) and that View has a Canvas. We can override the `View.onDraw()` method and that allows us to draw on the canvas.

Path: The Path class holds a set of vector-drawing commands. These allow us to create a variety of geometric shapes, like lines, rectangles, curves and circles. Because they are vector based, they are efficient and can be easily resized without distortion or loss of resolution.

Drawable: While Path gives us access to vector-based graphics, the Drawable class is what we use if we want to use raster-based graphics like bitmaps or a solid color that will be used of display only, like a background. There are a variety of forms for drawables, including PNG or JPEG images, states, levels, layers, and scale. These can work together to allow us to create a zoom or stretch feature, or various images, one on top of another.

Because we need to see at least some of these in action, what we will do next is start a very simple, new project that will use a variety of these graphics tools. Then we will be in a better position to actually start creating the grid interface for our game. Everyone should be comfortable with the fact this will require a new Class (.java file) and **may** also require a new layout (.xml file). Since we will be using the graphics system in Android, you should be comfortable with the idea what no .xml layout file will be needed, because we will be drawing what we need right there on the screen.

IV. Quick Aside: Building a simple Graphics Project.

We want to take advantage of the cPaint we laid out above, and make something that actually shows what the results are of creating graphics with code alone. This means our XML layout file(s) will be irrelevant to what is displayed on screen.

NOTE: You will not be asked to submit this Project as part of any Homework PROJECT. If you wish to just follow along in class and then build this on your own if and when you wish, that is entirely up to you.

Start a new project called My First 2D Graphics Example, or similar. Name the various files appropriately and choose "Basic Activity."

We are going to this entirely with code, so we will not be using or altering our XML layout file(s) or any other XML files.

We need to add some libraries:

```
import android.view.View;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Path;
import android.graphics.Path.Direction;
import android.graphics.Paint;
import android.graphics.LinearGradient;
import android.graphics.Shader;
```

Note how "specific" some of these libraries are. "LinearGradient" and "Shader" have their own libraries.

We need to extend the Activity, and in this case, it will then call a GraphicsView method that we will also extend:

```
public class MyFirst2DMainActivity extends AppCompatActivity {

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(new GraphicsView(this));

	}

//GraphicsView will go here and contain our drawing commands
}
```

We need to create a View, and then we need to draw what we need within that view. The drawing will be done with onDraw and it is crucial to remember that **all drawing**, including any updates, must be done/controlled within your onDraw event or method.

First, we need to create the Canvas we wish to draw on with the onDraw method. Then we will draw a background gradient.

Second, we will add the cPaint custom class from above, to create the outline around our circle of stroke weight 3 and color LTGREY (consider trying YELLOW if you try the "blue" gradient I use below).

Third, we need to create a tPaint custom class that will control the appearance of our text, including "scaling" it to fit into our circle. Depending on the text you put into your "String QUOTE," you may need to adjust this so it actually fits in the circle without overlapping (obscuring) part of your text with itself.

Fourth, and finally, we need to create a custom Path class to actually create the circle and use the cPaint and tPaint classes to control its appearance. The "density" property uses Android's density-independent-pixel system to set the center of the circle at 150, 150 and give it a radius of 100. Remember, Path is the class that contains the vector-based graphics options. The "CW" for direction means the circle will be draw clock-wise, and that is how the text will be displayed. Then we need to draw the circle with drawPath and add the text with drawTextOnPath. Note that these refer to cPaint and tPaint, respectively.

```
static public class GraphicsView extends View {
		public GraphicsView(Context context) {
			super(context);
		}
		@Override
		protected void onDraw(Canvas canvas) {
		//Drawing commands will go here

			Paint p = new Paint();
			// start at 0,0 and go to 0,max to use a vertical
			// gradient the full height of the screen.
			p.setShader(new LinearGradient(0, 0, 0, getHeight(), Color.rgb(00,00,100),
			Color.rgb(48,76,255), Shader.TileMode.MIRROR));

			canvas.drawPaint(p);

			Paint cPaint;

			cPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
			//Consider YELLOW if you use the BLUE gradient above.
			cPaint.setColor(Color.LTGRAY);
			cPaint.setStyle(Paint.Style.STROKE);
			cPaint.setStrokeWidth(3);

			Paint tPaint;

			tPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
			tPaint.setStyle(Paint.Style.FILL_AND_STROKE);
```

```
                tPaint.setColor(Color.BLACK);
                //Consider MAGENTA if you use the BLUE gradient above.
                float scale = getResources().getDisplayMetrics().scaledDensity;
                tPaint.setTextSize(18*scale);

                Path circle;

                circle = new Path();
                float density=getResources().getDisplayMetrics().density;
                circle.addCircle(175*density,  300*density, 100*density, Direction.CW);

                String QUOTE = "That's one small step for [a] man, "
                             + "one giant leap for mankind. - July 21, 1969";
                canvas.drawPath(circle,  cPaint);
                canvas.drawTextOnPath(QUOTE,  circle,  0,  tPaint.getTextSize(),  tPaint);

        }
    }
```

The final “takeaway” here is that we have created this graphic with the use of two custom classes (cPaint, tPaint), and an instance of a circle from the Path class, all draw on an instance of Canvas, controlled and created by an instance of GraphicsView, within our main activity.

We are now in a position to begin creating our game screen, which will also rely on code instead of XML layout.

V. What’s Next?

First, we will complete the New Game “Select” screen, which we started in this guide. Then we will begin to create our Game class that will actually display the game grid. The next Guide will probably be our last on this project, even though it will be incomplete. We want to move on to some other aspects by creating several other, basic applications.

BE SURE TO SAVE YOUR PACKAGE FOLDER TO YOUR THUMB DRIVE BEFORE YOU DISMOUNT IT.

Here is the entire graphics “aside” in one “piece” so it is easier to see how all the pieces fit together. Notice that this does not show the two public classes for the “Menu” that were automatically created. I did not mention deleting them in the guide (above), but they can be removed as they serve no purpose for this app. If you delete them, then you can also remove the Menu and MenuItem “view” libraries, as they will not be used. You also need to remove the “button” for email from the “activity” layout file to avoid warnings or even errors. Again, if you are becoming comfortable with our environment, all of this should make sense without extensive explanation by me.

```
package edu.indiana.jwhitmer.myfirst2dgraphicsexample;

import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;


import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Path;
import android.graphics.Path.Direction;
import android.graphics.Paint;
import android.graphics.LinearGradient;
import android.graphics.Shader;
```

```
public class MyFirstGraphicsMainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new GraphicsView(this));

    }

//GraphicsView will go here and contain our drawing commands

       static public class GraphicsView extends View {
            public GraphicsView(Context context) {
                super(context);
            }
            @Override
            protected void onDraw(Canvas canvas) {
                //Drawing commands will go here

                Paint p = new Paint();
                // start at 0,0 and go to 0,max to use a vertical
                // gradient the full height of the screen.
                p.setShader(new LinearGradient(0, 0, 0, getHeight(), Color.rgb(00,00,100),
                         Color.rgb(48,76,255), Shader.TileMode.MIRROR));

                        canvas.drawPaint(p);

                         Paint cPaint;

                         cPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
                         cPaint.setColor(Color.YELLOW);
                         cPaint.setStyle(Paint.Style.STROKE);
                         cPaint.setStrokeWidth(3);

                         Paint tPaint;

                         tPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
                         tPaint.setStyle(Paint.Style.FILL_AND_STROKE);
                         tPaint.setColor(Color.MAGENTA);
                         float scale = getResources().getDisplayMetrics().scaledDensity;
                         tPaint.setTextSize(18*scale);

                         Path circle;

                         circle = new Path();
                         float density=getResources().getDisplayMetrics().density;
                         circle.addCircle(175*density,  300*density, 100*density, Direction.CW);

                         String QUOTE = "That's one small step for [a] man, "
                                 + "one giant leap for mankind. - July 21, 1969";
                         canvas.drawPath(circle,  cPaint);
                         canvas.drawTextOnPath(QUOTE,  circle,  0,  tPaint.getTextSize(),  tPaint);

                    }
                }

        }
```