A290/A590 – Meeting 9 Guide
Building a "Real App" — Part 3 (Wrap-up)

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 options 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. Building the Game and Puzzle classes for Sudoku – Getting Started.

There is more to creating our game grid within the game. We also need to make sure there is a way to start a game. That also means a new class, FirstRealAppGameActivity.

To call this new class in our FirstRealAppMainActivity, we will need:

```
	private void startGame(int i){
		Log.d(TAG, "Clicked on " + i);
		Intent intent = new Intent(this, FirstRealAppGameActivity.class);
		intent.putExtra(FirstRealAppGameActivity.KEY_DIFFICULTY, i);
		startActivity(intent);
	}
```

Clearly, our new class will control most of this. We need a new Intent, and the "putExtra" is part of the intent, designed to store key/value pairs that will be passed along to the new activity created by the new class.

Create a new class file: FirstRealAppGameActivity and add the following libraries:

```
package edu.indiana.a290firstrealapplication;

import android.app.Activity;
import android.app.Dialog;
import android.os.Bundle;
import android.util.Log;
import android.view.Gravity;
import android.widget.Toast;
```

We may not get to the point of using all of these, but let's get them in there anyway.

Now things will get complicated quickly. There are a variety of ways this could have been done. What we will try here is to call a new "FirstRealAppPuzzleView" class "into" our FirstRealAppGameActivity class, and that will become the content of this view. We will select the difficulty from within FirstRealAppGameActivity, and then "load" the view with the custom class for the selected "level" from FirstRealAppPuzzleView.

We will need to extend our activity, add a TAG so we can track a few things in Debug mode (if time permits) and then define the DIFFICULTY levels and define the onCreate method we will need. We will also have to create our SudokuPuzzleView class, and define both the getPuzzle() method and the calculateUsedTiles() method.

```
public class FirstRealAppGameActivity extends Activity {

	private static final String TAG = "FirstRealAppGameActivity";

	public static final String KEY_DIFFICULTY =
			"edu.indiana.a290firstrealapplication.difficulty";

	public static final int DIFFICULTY_EASY = 0;
	public static final int DIFFICULTY_MEDIUM = 1;
	public static final int DIFFICULTY_HARD = 2;
```

```
	private int puzzle[];

	private FirstRealAppPuzzleView puzzleView;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		Log.d(TAG,"onCreate");

		int diff = getIntent().getIntExtra(KEY_DIFFICULTY,
			DIFFICULTY_EASY);
		puzzle = getPuzzle(diff);
		calculateUsedTiles();

		puzzleView = new FirstRealAppPuzzleView(this);
		setContentView(puzzleView);
		puzzleView.requestFocus();

	}
//We have more work to do here...
}
```

[Remember: You will get a large number of “red text” errors if you follow this guide. They will all disappear when everything in this guide is completed successfully.]

II. Expanding the Game and Puzzle classes for our Game.

Next, we want to create our FirstRealAppPuzzleView class and get all the pieces “linked” together. Then we can start to build the game grid with our graphics classes.

To be sure we have the string resources we need, let’s add the following 3 to strings.xml. I’m showing them as they would appear in XML view:

```
<string name="GameTitle">Game</string>
<string name="NoMovesLabel">No moves possible</string>
<string name="KeypadTitle">Keypad</string>
```

Before we forget, we need to add our FirstRealAppGameActivity to our manifest, or nothing will work even after all our code is complete.

```
        <activity
            android:name=".FirstRealAppGameActivity"
            android:label="@string/GameTitle">

        </activity>
```

Now we need to create our Puzzle class: FirstRealAppPuzzleView. Then we need to add an extensive set of libraries, since this is where we will be building our 9x9 grid with Android graphics.

```
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Rect;
import android.graphics.Paint.FontMetrics;
import android.graphics.Paint.Style;
import android.util.Log;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.view.View;
```

```
import android.view.animation.AnimationUtils;
```

We need to extend our View and setup the call for the instance of the game.

```
public class FirstRealAppPuzzleView extends View {

	private static final String TAG = "FirstRealAppPuzzleView";

	private final FirstRealAppGameActivity game;

	public FirstRealAppPuzzleView(Context context) {
		super(context);
		this.game = (FirstRealAppGameActivity) context;
		setFocusable(true);
		setFocusableInTouchMode(true);
	}
//We have more to fill in here

}
```

Next, we will setup the framework for our grid, but only after we know how big the “screen” is. This will insure that our tiles are the proper size for whatever resolution the screen may be. This all goes inside FirstRealAppPuzzleView, after the public method we just added.

```
	private float width; //width of one tile
	private float height; //height of one tile
	private int selX; //X index of selection
	private int selY; //Y index of selection
	private final Rect selRect = new Rect();

	@Override
	protected void onSizeChanged(int w, int h, int oldw, int oldh) {
		width = w/9f;
		height = h/9f;
		getRect(selX, selY, selRect);
		Log.d(TAG, "onSizeChanged: width " + width + ", height " + height);
		super.onSizeChanged(w, h, oldw, oldh);
	}

	private void getRect(int x, int y, Rect rect) {

		rect.set((int) (x * width), (int) (y * height),
				(int) (x * width + width), (int) (y * height + height));
	}
```

Since we are creating a totally custom class and using code to create the graphics, we need to define all the various color resources we will need. Then we will be ready to add the onDraw method and start drawing our grid. Again, I have just selected a related set of colors and I’m displayed them in XML view, from **colors.xml**.

```
<color name="PuzzleBackground">#ffe6f0ff</color>
<color name="PuzzleHiLite">#ffffffff</color>
<color name="PuzzleLight">#64c6d4ef</color>
<color name="PuzzleDark">#6456648f</color>
<color name="PuzzleForeground">#ff000000</color>
<color name="PuzzleSelected">#64ff8000</color>
<color name="PuzzleHint0">#64ff0000</color>
<color name="PuzzleHint1">#6400ff80</color>
<color name="PuzzleHint2">#2000ff80</color>
```

Let's get our onDraw method started, by created the canvas and setting the background color. This is also part of our "…PuzzleView" class.

```
@Override
protected void onDraw(Canvas canvas) {
	//Draw the background first
	Paint background = new Paint();
	background.setColor(getResources().getColor(R.color.PuzzleBackground));
	canvas.drawRect(0,  0, getWidth(), getHeight(), background);

	//Draw the board

	//Draw the numbers, after the game difficulty is selected

	//Draw the hints during the game

	//Draw the tile selected, so it is clearly distinguished from the other tiles

}
```

To draw the board, we need to do things in the correct order. For graphics, whatever you create will "draw over" what is already there. If we want a 9x9 grid to show, but then want to add emphasis to the 3x3 blocks within the larger grid, we need to draw the "minor" [9x9] gridlines first, and then the "major" [3x3] gridlines "over" the minor gridlines.

You may have noticed when we set our colors, that besides "light" for the minor gridlines and "dark" for the major gridlines, I also defined "hilite." We will use this to draw additional lines, offset by "+ 1" dp, to give all the lines a sort of "etched" appearance. The code is somewhat long, tedious, and repetitive, but that is what is required with a completely custom class.

```
//Draw the board
	//First, define the colors for the grid lines.
Paint dark = new Paint();
dark.setColor(getResources().getColor(R.color.PuzzleDark));
Paint hilite = new Paint();
hilite.setColor(getResources().getColor(R.color.PuzzleHiLite));
Paint light = new Paint();
light.setColor(getResources().getColor(R.color.PuzzleLight));

	//Second, draw the minor grid lines (order is important)
for (int i =0; i < 9; i++){
	canvas.drawLine(0, i * height, getWidth(), i * height, light);
	canvas.drawLine(0, i * height + 1, getWidth(), i * height + 1, hilite);
	canvas.drawLine(i * width, 0, i * width, getHeight(), light);
	canvas.drawLine(i * width + 1, 0, i * width + 1, getHeight(), hilite);

}

	//Third, draw the major grid lines (order is important)
for (int i = 0; i < 9; i++) {
	if(i % 3 != 0)
		continue;
	canvas.drawLine(0,  i * height,  getWidth(),  i * height,  dark);
	canvas.drawLine(0, i * height + 1, getWidth(), i * height + 1, hilite);
	canvas.drawLine(i * width,  0,  i * width,  getHeight(),  dark);
	canvas.drawLine(i * width + 1,  0,  i * width + 1,  getHeight(),  hilite);
}
```

We can actually "test" our results, as long as we **comment out** those custom methods we have not yet defined in our FirstRealAppGameActivity. There are two of them:

```
//puzzle = getPuzzle(diff);
//calculateUsedTiles();
```

Once you have commented them out, start your emulator and Run your app. The New Game button should show the three difficulty levels and, at this point, choosing any of them should bring up the blank game grid that we have just created.

III. Setting up the Graphics to draw our Numbers

We have our game grid, now we need to draw the initial set of numbers in the various tiles that will represent the "starting" condition for each of our three difficulty levels. These three options will be hard-coded at this point, so we will get the same "starting" condition every time we select a particular level. We will be using simple strings to populate the tiles either with a number or using "0" as a placeholder in the string, with a blank space.

Because we want to see what the three levels look like populated with numbers, we will do things in a slightly unusual order. First, let's try to get the remaining methods related to the game logic into our classes, so we can actually call the three options and draw the numbers. We need to add the following to FirstRealAppGameActivity, so we can monitor which tiles are "used" and which or "open."

```
	private final int used[][][] = new int[9][9][];

	protected int[] getUsedTiles(int x, int y) {
		return used[x][y];
	}

	private void calculateUsedTiles() {

		for (int x = 0; x < 9; x++) {
			for (int y = 0; y < 9; y++) {
				used[x][y] = calculateUsedTiles(x, y);
				//have more work to do here for both
				//LogCat and toPuzzleString
			}
		}
	}

	private int[] calculateUsedTiles(int x, int y) {

		int c[] = new int[9];
		//check horizontal tiles
		for (int i = 0; i < 9; i++) {
			if (i == x)
				continue;
			int t = getTile(i, y);
			if (t != 0)
				c[t - 1] = t;
		}

		//check vertical tiles
		for (int i = 0; i < 9; i++) {
			if (i == y)
				continue;
			int t = getTile(i, x);
			if (t != 0)
				c[t - 1] = t;
		}
```

```
            //check 3x3 block
            int startx = (x / 3) * 3;
            int starty = (y / 3) * 3;
            for (int i = startx; i < startx + 3; i++) {
                  for (int j = starty; j < starty + 3; j++) {
                        if (i == x && j == y)
                              continue;
                        int t = getTile(i, j);
                        if (t != 0)
                              c[t - 1] = t;
                  }
            }

            //compress the array to save some space
            int nused = 0;
            for (int t : c) {
                  if (t != 0)
                        nused++;
            }

            int c1[] = new int[nused];
            nused = 0;
            for (int t : c) {
                  if (t != 0)
                        c1[nused++] = t;
            }
      return c1;
      }

//We have more work to do here...
```

Next, we need to return to our FirstRealAppPuzzleView, so we can add the code to draw the numbers.

```
//Draw the numbers, after the game difficulty is selected

     // Define color and style for numbers
     Paint foreground = new Paint(Paint.ANTI_ALIAS_FLAG);
     foreground.setColor(getResources().getColor(
           R.color.PuzzleForeground));
     foreground.setStyle(Style.FILL);
     foreground.setTextSize(height * 0.75f);
     foreground.setTextScaleX(width / height);
     foreground.setTextAlign(Paint.Align.CENTER);

      // Draw the number in the center of the tile
      FontMetrics fm = foreground.getFontMetrics();
      // Centering in X: use alignment (and X at midpoint)
      float x = width / 2;
      // Centering in Y: measure ascent/descent first
      float y = height / 2 - (fm.ascent + fm.descent) / 2;
      for (int i = 0; i < 9; i++) {
         for (int j = 0; j < 9; j++) {
            canvas.drawText(this.game.getTileString(i, j), i
                  * width + x, j * height + y, foreground);
         }
      }
```

To wrap things up so we can at least load different sets of numbers into our game grid, depending on what level of difficulty is selected, we need to add a variety of things.

First, we need to return to FirstNewAppGameActivity and setup the “hard-coded” number patterns for the 3 levels. These are just laid out as strings.

```
   private final String easyPuzzle =
      "360000000004230800000004200" +
      "070460003820000014500013020" +
      "001900000007048300000000045";
   private final String mediumPuzzle =
      "650000070000506000014000005" +
      "007009000002314700000700800" +
      "500000630000201000030000097";
   private final String hardPuzzle =
      "009000000080605020501078000" +
      "000000700706040102004000000" +
      "000720903090301080000000600";
```

Second, also in FirstNewAppGameActivity we will use a switch-case construct to control which of the three “strings” is used to populate the game board, depending on the level of difficulty selected.

```
/** Given a difficulty level, come up with a new puzzle */
   private int[] getPuzzle(int diff) {
      String puz;
      // TODO: Continue last game
      switch (diff) {

       case DIFFICULTY_HARD:
          puz = hardPuzzle;
          break;

      case DIFFICULTY_MEDIUM:
          puz = mediumPuzzle;
          break;

      case DIFFICULTY_EASY:
      default:
          puz = easyPuzzle;
          break;
      }
      return fromPuzzleString(puz);
   }

/** Convert a puzzle string into an array */
          static protected int[] fromPuzzleString(String string) {
             int[] puz = new int[string.length()];
             for (int i = 0; i < puz.length; i++) {
                puz[i] = string.charAt(i) - '0';
             }
             return puz;
          }

/** Return the tile at the given coordinates */
          private int getTile(int x, int y) {
             return puzzle[y * 9 + x];
          }

/** Return a string for the tile at the given coordinates */
          protected String getTileString(int x, int y) {
             int v = getTile(x, y);
             if (v == 0)
```

```
            return "";
        else
            return String.valueOf(v);
    }
```

All these together allow us to move back and forth between our hard-coded puzzle strings and the array we will/would eventually use to keep track of which tiles are filled, and what the possible options are for those that are blank. That would be one of our next steps. We would also have to decide how to control the selection of an “empty” tile and then, how the game player would be permitted to enter a value. This would almost certainly involve using the keyboard.

Remember to “un comment” the two lines from FirstRealAppGameActivity

```
puzzle = getPuzzle(diff);
calculateUsedTiles();
```

And then run your app in the emulator. We will only run it, but remember we have inserted some TAG declarations and some LogCat markers that will allow you to follow the Lifecycle if you start in “Debug” mode.

IV. What’s Next?

So, if we had time to finish this game, we would have to create at least one more Activity (as noted above), for a keypad that would allow the game player to enter the number desired in the cell that had been selected. This means we would also have to add to our FirstRealAppPuzzleView so that it would be possible to select any of the “open” tiles but not any of the “used” tiles. I have decided we will stop here, so that you have time to produce your own version of this “real” app to this stage for Homework Project 3.

Next time, we will look at one or more simple projects that involve audio, video and using/accessing the web. These may take more than one meeting to complete.

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