Extracted from:

Hello, Android

Introducing Google's Mobile Development Platform

The Pragmatic Programmers

Hello, Android

Introducing Google's Mobile Development Platform

Ed Burnette

Edited by Susannah Davidson Pfalzer

ISBN-10: 1-934356-17-4

ISBN-13: 978-1-934356-17-3

Printed on acid-free paper.

P1.3 printing, June 22, 2009

Version: 2009-6-23

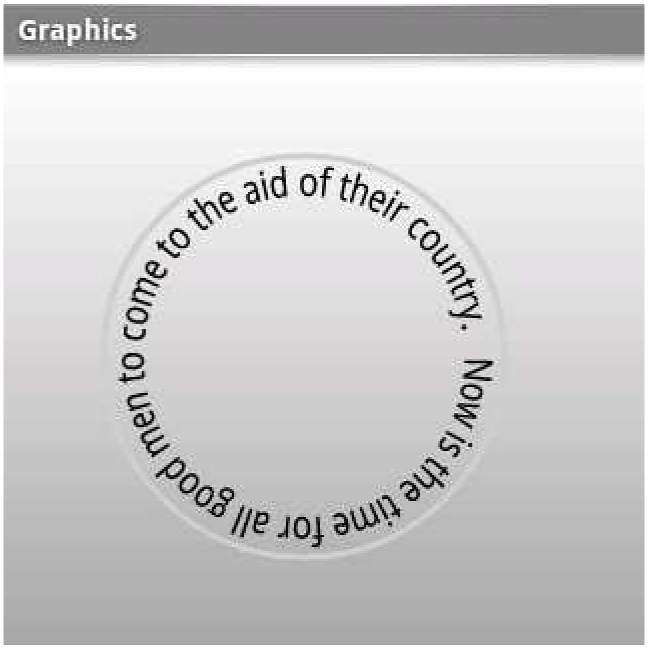

Graphics
Now is the time for all good men to come to the aid of their country.

Figure 4.2: Using a gradient background defined in XML

4.2 Adding Graphics to Sudoku

It's time to apply what we've learned to our Sudoku example. When we left it at the end of Chapter 3, the Sudoku game had an opening screen, an About dialog box, and a way to start a new game. But it was missing one very important part: the game! We'll use the native 2D graphics library to implement that part.

Starting the Game

First we need to fill in the code that starts the game. startGame() takes one parameter, the index of the difficulty name selected from the list. Here's the new definition:

Download Sudokuv2/src/org/example/sudoku/Sudoku.java

```java
/** Start a new game with the given difficulty level */
private void startGame(int i) {
   Log.d(TAG, "clicked on " + i);
   Intent intent = new Intent(Sudoku.this, Game.class);
   intent.putExtra(Game.KEY_DIFFICULTY, i);
   startActivity(intent);
```

Sudoku Trivia

A few years after it was published in the United States, Number Place was picked up by the Japanese publisher Nikoli, who gave it the much cooler-sounding name Sudoku (which means "single number" in Japanese). From there it was exported around the world, and the rest is history. Sadly, Garns died in 1989 before getting a chance to see his creation become a worldwide sensation.

```
}
```

The game part of Sudoku will be another activity called Game, so we create a new intent to kick it off. We place the difficulty number in an extraData area provided in the intent, and then we call the startActivity() method to launch the new activity.

The extraData area is a map of key/value pairs that will be passed along to the intent. The keys are strings, and the values can be any primitive type, array of primitives, Bundle, or a subclass of Serializable or Parcelable.

Defining the Game Class

Here's the outline of the Game activity:

Download Sudokuv2/src/org/example/sudoku/Game.java

```java
package org.example.sudoku;

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

public class Game extends Activity {
   private static final String TAG = "Sudoku";

   public static final String KEY_DIFFICULTY =
      "org.example.sudoku.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[] = new int[9 * 9];
   private PuzzleView 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 PuzzleView(this);
      setContentView(puzzleView);
      puzzleView.requestFocus();
   }
   // ...
}
```

The onCreate() method fetches the difficulty number from the intent and selects a puzzle to play. Then it creates an instance of the PuzzleView class, setting the PuzzleView as the new contents of the view. Since this is a fully customized view, it was easier to do this in code than in XML.

The calculateUsedTiles() method, which is defined in Section 4.4, *The Rest of the Story*, on page 95, uses the rules of Sudoku to figure out, for each tile in the nine-by-nine grid, which numbers are not valid for the tile because they appear elsewhere in the horizontal or vertical direction or in the three-by-three subgrid.

This is an activity, so we need to register it in AndroidManifest.xml:

Download Sudokuv2/AndroidManifest.xml

```
<activity android:name=".Game"
      android:label="@string/game_title"/>
```

We also need to add a few more string resources to res/values/strings.xml:

Download Sudokuv2/res/values/strings.xml

```
<string name="game_title">Game</string>
<string name="no_moves_label">No moves</string>
<string name="keypad_title">Keypad</string>
```

Defining the PuzzleView Class

Next we need to define the PuzzleView class. Instead of using an XML layout, this time let's do it entirely in Java.

Here's the outline:

What Size Is It Anyway?

A common mistake made by new Android developers is to use the width and height of a view inside its constructor. When a view's constructor is called, Android doesn't know yet how big the view will be, so the sizes are set to zero. The real sizes are calculated during the layout stage, which occurs after construction but before anything is drawn. You can use the onSizeChanged( ) method to be notified of the values when they are known, or you can use the getWidth( ) and getHeight( ) methods later, such as in the onDraw( ) method.

Download Sudokuv2/src/org/example/sudoku/PuzzleView.java

```java
package org.example.sudoku;

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;

public class PuzzleView extends View {
   private static final String TAG = "Sudoku";
   private final Game game;
   public PuzzleView(Context context) {
      super(context);
      this.game = (Game) context;
      setFocusable(true);
      setFocusableInTouchMode(true);
   }
   // ...
}
```

In the constructor we keep a reference to the Game class and set the option to allow user input in the view. Inside PuzzleView, we need to implement the onSizeChanged() method. This is called after the view is created and Android knows how big everything is.

Download Sudokuv2/src/org/example/sudoku/PuzzleView.java

```
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));
}
```

We use onSizeChanged() to calculate the size of each tile on the screen (1/9th of the total view width and height). Note this is a floating-point number, so it's possible that we could end up with a fractional number of pixels. selRect is a rectangle we'll use later to keep track of the selection cursor.

At this point we've created a view for the puzzle, and we know how big it is. The next step is to draw the grid lines that separate the tiles on the board.

Drawing the Board

Android calls a view's onDraw() method every time any part of the view needs to be updated. To simplify things, onDraw() pretends that you're re-creating the entire screen from scratch. In reality, you may be drawing only a small portion of the view as defined by the canvas's clip rectangle. Android takes care of doing the clipping for you.

Start by defining a few new colors to play with in res/values/colors.xml:

Download Sudokuv2/res/values/colors.xml

```
<color name="puzzle_background">#ffe6f0ff</color>
<color name="puzzle_hilite">#ffffffff</color>
<color name="puzzle_light">#64c6d4ef</color>
<color name="puzzle_dark">#6456648f</color>
<color name="puzzle_foreground">#ff000000</color>
<color name="puzzle_hint_0">#64ff0000</color>
```

Other Ways to Do It

When I was writing this example, I tried several different approaches such as using a button for each tile or declaring a grid of ImageView classes in XML. After many false starts, I found that the approach of having one view for the entire puzzle and drawing lines and numbers inside that proved to be the fastest and easiest way for this application.

It does have its drawbacks, though, such as the need to draw the selection and explicitly handle keyboard and touch events. When designing your own program, I recommend trying standard widgets and views first and then falling back to custom drawing only if that doesn't work for you.

```
<color name="puzzle_hint_1">#6400ff80</color>
<color name="puzzle_hint_2">#2000ff80</color>
<color name="puzzle_selected">#64ff8000</color>
```

Here's the basic outline for onDraw():

Download Sudokuv2/src/org/example/sudoku/PuzzleView.java

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

   // Draw the board...
   // Draw the numbers...
   // Draw the hints...
   // Draw the selection...
}
```

The first parameter is the Canvas on which to draw. In this code, we're just drawing a background for the puzzle using the puzzle_background color.

Now let's add the code to draw the grid lines for the board:

Download Sudokuv2/src/org/example/sudoku/PuzzleView.java

```
// Draw the board...
// Define colors for the grid lines
Paint dark = new Paint();
```

```
dark.setColor(getResources().getColor(R.color.puzzle_dark));

Paint hilite = new Paint();
hilite.setColor(getResources().getColor(R.color.puzzle_hilite));

Paint light = new Paint();
light.setColor(getResources().getColor(R.color.puzzle_light));

// Draw the minor grid lines
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);
}

// Draw the major grid lines
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);
}
```

The code uses three different colors for the grid lines: a light color between each tile, a dark color between the three-by-three blocks, and a highlight color drawn on the edge of each tile to make them look like they have a little depth. The order in which the lines are drawn is important, since lines drawn later will be drawn over the top of earlier lines. You can see what this will look like in Figure 4.3, on the next page. Next, we need some numbers to go inside those lines.

Drawing the Numbers

The following code draws the puzzle numbers on top of the tiles. The tricky part here is getting each number positioned and sized so it goes in the exact center of its tile.

Download Sudokuv2/src/org/example/sudoku/PuzzleView.java

```
// Draw the numbers...
```

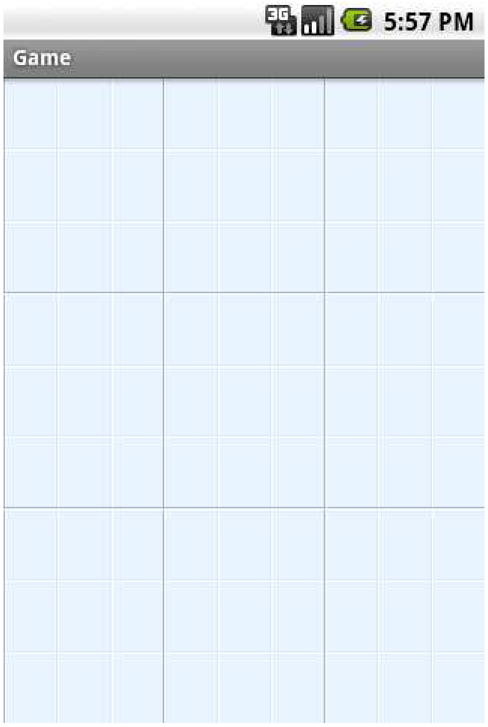
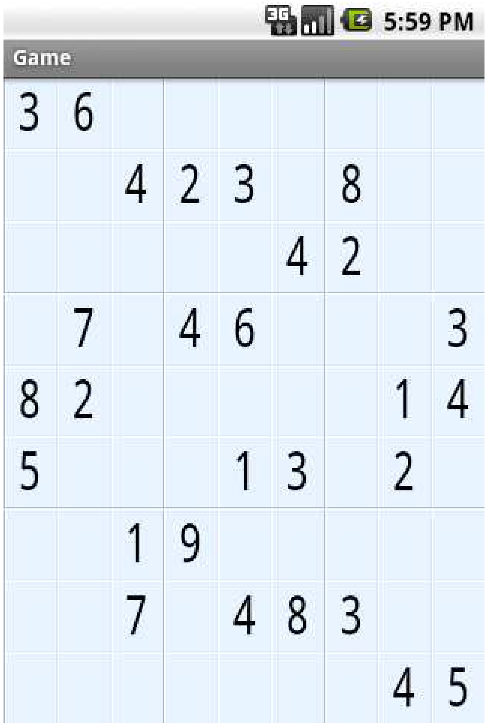


Figure 4.3: Drawing the grid lines with three colors for effect

```
// Define color and style for numbers
Paint foreground = new Paint(Paint.ANTI_ALIAS_FLAG);
foreground.setColor(getResources().getColor(
      R.color.puzzle_foreground));
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);
   }
```



3	6							
		4	2	3		8		
					4	2		
	7		4	6				3
8	2						1	4
5				1	3		2	
		1	9					
		7		4	8	3		
							4	5

Figure 4.4: Centering the numbers inside the tiles

```
}
```

We call the getTileString() method (defined in Section 4.4, *The Rest of the Story*, on page 95) to find out what numbers to display. To calculate the size of the numbers, we set the font height to three-fourths the height of the tile, and we set the aspect ratio to be the same as the tile's aspect ratio. We can't use absolute pixel or point sizes because we want the program to work at any resolution.

To determine the position of each number, we center it in both the x and y dimensions. The x direction is easy—just divide the tile width by 2. But for the y direction, we have to adjust the starting position downward a little so that the midpoint of the tile will be the midpoint of the number instead of its baseline. We use the graphics library's FontMetrics class to tell how much vertical space the letter will take in total, and then we divide that in half to get the adjustment. You can see the results in Figure 4.4.

That takes care of displaying the puzzle's starting numbers (the givens). The next step is to allow the player to enter their guesses for all the blank spaces.

4.3 Handling Input

One difference in Android programming—as opposed to, say, iPhone programming—is that Android phones come in many shapes and sizes and have a variety of input methods. They might have a keyboard, a D-pad, a touch screen, a trackball, or some combination of these.

A good Android program, therefore, needs to be ready to support whatever input hardware is available, just like it needs to be ready to support any screen resolution.

Defining and Updating the Selection

First we're going to implement a little cursor that shows the player which tile is currently selected. The selected tile is the one that will be modified when the player enters a number. This code will draw the selection in onDraw():

Download Sudokuv2/src/org/example/sudoku/PuzzleView.java

```java
// Draw the selection...
Log.d(TAG, "selRect=" + selRect);
Paint selected = new Paint();
selected.setColor(getResources().getColor(
      R.color.puzzle_selected));
canvas.drawRect(selRect, selected);
```

We use the selection rectangle calculated earlier in onSizeChanged() to draw an alpha-blended color on top of the selected tile.

Next we provide a way to move the selection by overriding the onKeyDown() method:

Download Sudokuv2/src/org/example/sudoku/PuzzleView.java

```java
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
   Log.d(TAG, "onKeyDown: keycode=" + keyCode + ", event="
         + event);
   switch (keyCode) {
   case KeyEvent.KEYCODE_DPAD_UP:
      select(selX, selY - 1);
      break;
   case KeyEvent.KEYCODE_DPAD_DOWN:
      select(selX, selY + 1);
```