Lab 2: Lights Out (Buttons and MVC)

Goal

The goal for this lab is to practice building simple Android apps that make use of buttons. We will also learn about how to handle screen rotation changes.

Summary

The game you will be developing is called Lights Out. The goal is to click lights and eventually turn them all off, hence the name “Lights Out”. When a user clicks a light it toggles the state of the light that was clicked and toggles the state of all neighbor lights. Instead of lights we’ll just use 0 and a 1 and try to make all the buttons match. In our version we will have a line of exactly seven buttons arranged in a line. We fix the number to keep it simple.

Requirements

I usually present these as a list of user stories, annotated with screenshots to clarify.

1. Initially the buttons have random values.
2. When the user clicks a button, that button and its two neighboring buttons change from 0 to 1, or vice versa.
3. Clicking the first or last button toggles only two buttons; there is no wraparound.
4. The button row and text will be centered. The text size should be 20 sp. The new game button should be right-aligned with the buttons. You can choose reasonable margins. Make the toolbar at the top black and the background color something fun (anything other than black). Research or figure out which color in colors.xml is used for the toolbar and change it.
5. The buttons must have the same width - you can change the button width to a fixed width (in dp) or use stretch columns like we did in the lecture to make them fit.
6. When it starts (or re-starts a new game), the message will display “Make the buttons match”. Otherwise, it will count the number of turns taken so far, using proper pluralization. When the game was won then the message will change to “You Win!” .
7. When a player wins, all the game buttons are disabled. Starting a new game re-enables them. When the buttons are disabled they can’t be clicked and they will gray out slightly. (Hint: we expect you to search the javadoc for buttons to find the method to do this.) The “New Game” button should always be enabled.
8. When you rotate the screen (press ctrl-F12 to do this if using an emulator), it uses a slightly different layout, shown below. It also remembers the game state (buttons and number of turns taken). (Hints below.)
9. You must use a model object, either this LightsOutGame.java or one you make yourself.

Note, you will need to update the package if you use the link above.

Screenshots

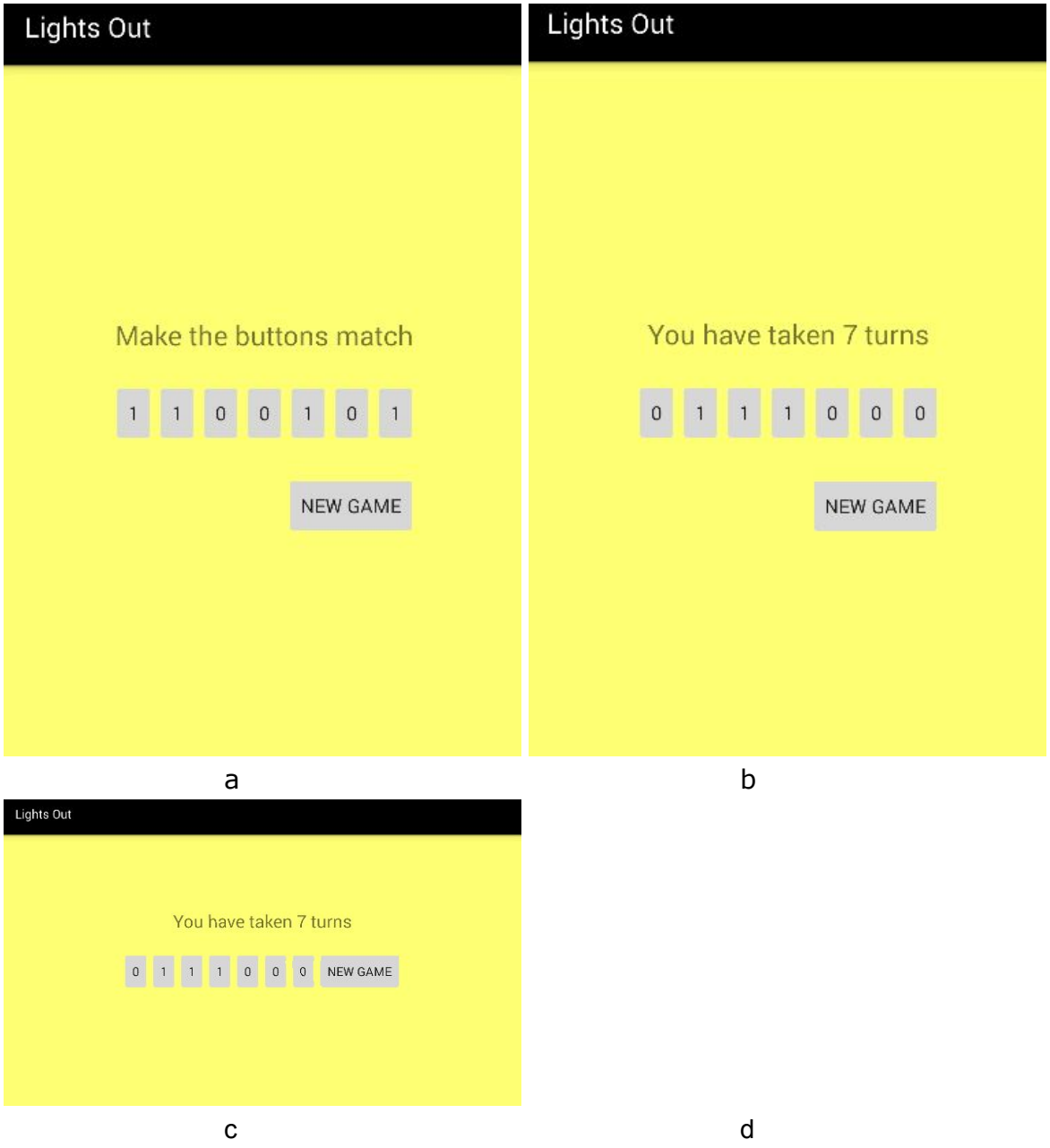

Figure 1: Game play. (a) At start. (b) After 7 button presses (c) After a rotation. Note the change in layout (new game button moves to right of others) and that the board and number of turns persists. (d) After winning the game, the message changes and the buttons are disabled.

More Info

All of these instructions relate to landscape orientation.

1. You specify the different layout for landscape (rotated) mode by creating a folder named `layout-land` in the `res` folder, and creating in it a different `activity_main.xml` (or whatever you named your layout file).
2. Since it loads the layout in `onCreate`, Android must destroy your activity and re-create it when you rotate it. We'll discuss the activity lifecycle later in the course, but for now, just realize that you need to save off the info upon rotation and reload it when it is re-created. You do this using `Bundles`. A bundle is just serializable data that follows a map interface: you just access data using key-value pairs. This leads to 2 places you'll need to add code

   In `MainActivity`, override Activity's `onSaveInstanceState` method:

   ```
   public void onSaveInstanceState(Bundle outState) {
      super.onSaveInstanceState(outState);
      // Save off data using outState.putXX(key, value)
      // Hint: you will use the appropriate methods to store int[] and ints,
      // maybe a String.
   ```

   Warning: Make sure you use the exact method with the arguments as shown above. There are multiple versions of onSaveInstanceState, make sure you get the right one!

   When `onCreate` is called, that data that you put there will be passed into the savedInstanceState parameter. So if that parameter isn't null, call the appropriate savedInstanceState.getXX(key) methods and use what it returns to initialize the game object and screen.

   More information, including another simple information is here.