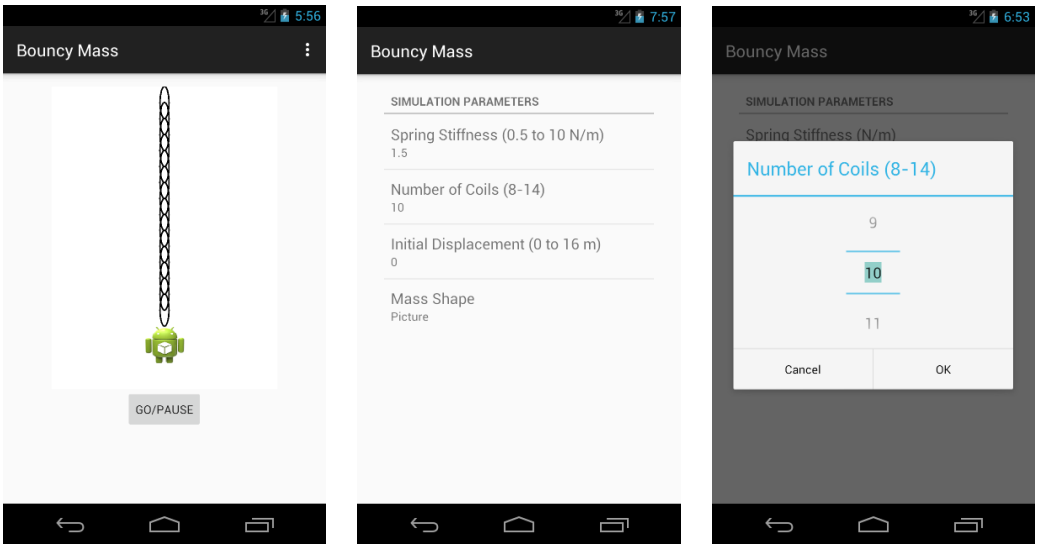
CSCD 372: Android Development
Lab 7: Preferences for the Bouncy Mass

Lab Overview

In this lab we add preferences to our bouncy mass that control the spring constant, the number of spring coils, the initial y position, and the shape drawn for the mass, as shown below.



Following are some suggestions on how to proceed. Because we are adding preferences to an already functioning project (which is a common development sequence for me), and that project consisted of a single Activity with no Fragments, I am suggesting that you follow the method of wrapping a Preference Fragment in a simple Preference Activity. If you ever find yourself adding Preferences to a project that contains one or more Fragments (as in the Fragments Lab), you would want to consider the option of swapping your Preference Fragment directly into a Fragment placeholder (another version of this lab covers that case, but I can't reasonably ask you to practice every variation in 10 weeks now, can I?).

Task 1: Copy your Bouncy Mass project and add a Preferences Activity

1. Copy the previous lab to a new directory, and open it in Android Studio. You will need to manually change your app name to reflect "Lab7". Refactor your package name likewise.
2. Right-click on your java source directory and choose New→Activity→Settings Activity. Set the hierarchical parent to your main activity. In Android Studio 1.5.x there a bug here in that you will have to enter your package name manually in order to get the wizard to accept your chosen Activity name. Or you can right-click one directory lower, on your non-test package.

Task 2: Create a preferences.xml file

1. Right-click on your res directory and choose New→Android Resource File. Set the type to XML, and the Root element to PreferenceScreen. As you work on this file, note that you can get a preview by pulling a pane out from the right hand side of your IDE.
2. Add the preferences shown above, with the following constraints:

a. For the Spring Stiffness, use an EditTextPreference, with a default value of 1.5. You'll need to specify a numeric attribute that allows decimal entries. It would make sense to limit the user's input to some reasonable range here, but I'm not going to require it. I will consider giving extra credit for using an input filter to do that (and can give some guidance and warnings if you like).

b. For the number of coils, use the custom NumberPickerPreference that I have posted on the course website. Give it a default value of 11 and set the range to 8-14, with no wraparound. Note that the min, max, and range are custom attributes and you'll need to add the attrs.xml file to your values resources as indicated in the source comments. I also have a custom seekbar preference widget if you'd like to see that.

c. Use a number picker for the initial displacement as well, with a range of 0 to 16 (meters).

d. For the Mass Shape, use a ListPreference, with options of Rectangle, Rounded Rectangle, Circle, and Picture.

Task 3: Add a Preference Activity and Preference Fragment

1. Right-click on your java source package directory and add a Blank Fragment with no options and no layout. Call it something like SettingsFragment.
2. Modify the generated code to extend PreferenceFragment, remove onCreateView, implement onSharedPreferenceChangeListener, and add overrides of onCreate, onResume, and onPause. Examples for these changes are shown in the Preferences lecture module. Leave onSharedPreferenceChanged blank for now.
3. Create an Empty Activity called something like SettingsActivity and include a layout XML file. If the compiler complains that it can't find AppCompatActivity, look in your build.gradle file (for Module:app) and make sure the dependency for appcompat-v7 matches your build SDK version (or see me for help). This can also be checked in your Project Structure dialog under dependencies.
4. Modify the layout XML to contain a static fragment, and specify your Fragment class in the name attribute.
5. Back in your main activity, respond to the settings action from your menu by launching an Intent on your settings activity class.
6. Run a test to make sure that the preferences launch and can be controlled as specified. They don't yet update the summaries (except for the ListPreference, if you set the summary attribute to "%s"), or control anything, but changes should be persistent between runs.

Task 4: Update Preference Summaries in the Change Listener

1. Add code to your OnSharedPreferenceChanged method that updates the preference summaries. This is done at runtime by calling setSummary() on the Preference object, which you can get from the key by calling findPreference(). Oddly enough, you cannot get the current value from that same object, but you can get it from the SharedPreference object passed into the method. An example is shown in the Preferences module. Note that the custom NumberPickerPreference supports getInt(), but the built-in preferences only support getString().
2. Run a test to make sure the summaries are modified when the corresponding preference is changed. Note that you won't get the call unless you actually change what the preference is currently set to. For example, backspacing over an entry and retyping it does not count as a change. The API is smarter than that.

3. Now back out of the preference screen to the main screen, and select Settings again. Hmmm, the settings fragment reinitializes each time, and doesn't reflect the current settings. It is unfortunate that the API doesn't have an option to set the UI from the values stored in the preference file. Maybe in the future it will, but in the meantime we can easily "cheat" by reusing the code we just wrote ...
4. In the onCreate() method of your Settings Fragment, get a reference to the SharedPreferences by calling getPreferenceScreen().getSharedPreferences(). Pass that into four calls to your onSharedPreferenceChanged() method, once for each of your preferences. That will force the summary fields to update based on the current value.
5. Test again and make sure it works.

Task 5: Modify your animation to respond to user preferences

1. In your main activity, add an override for onResume(), making sure to call the superclass version. Make a call to PreferenceManager.getDefaultSharedPreferences() in order to get a reference to the SharedPreferences, and read the 4 values. Doing this on onResume() assures that the preferences are read on either app launch OR when the settings Activity returns without a full re-create (see the Activity life cycle diagram).
2. Pass the settings you read along to your animated view, and modify your animation code to make use of them. For the mass shape preference, draw the matching shape when you draw the mass. For the picture option, use BitmapFactory to decode a drawable resource. As can be seen above, I simply used the launcher icon, but feel free to add some picture for that (perhaps an image of your face). Then draw that using canvas.drawBitmap(), which will allow you to specify a rectangle into which the bitmap should be scaled. As will be seen shortly, it is important for your animation to reset the y velocity to 0 anytime the initial y displacement or the spring constant changes. I simply reset both y and y velocity anytime any of the preferences are changed, restarting the simulation.
3. Perform test runs until you have all the options working. When you think it is all working, try setting the spring constant to 1.5 and the initial displacement to 7. The mass should barely bounce in this case. Do you understand why that is? This is why it is rather important to reset the y velocity when we change the physics parameters.
4. As a final sanity check on the simulation, note that the bounce period is determined by:

   $$T = 2\pi\sqrt{\frac{m}{k}}$$

   Does your period roughly match that prediction?
5. Make sure that nothing breaks when you rotate. Turn in your apk file, along with the source for your fragment, your main activity, your custom view, and your preferences xml.