# Milestone 4

18 March 2013

**due Sunday 24 March at midnight**

This milestone lasts for *three* weeks, one of them over spring break. The goal is to put a little more "meat" on the program, but it will by no means be complete.

Again, this milestone should be completed individually — I expect a commit from everyone. You may base your work on the same project you used for the `SettingsActivity` in Milestone 3. If you did not get that to work, I can provide a working version for you.

Start early!

## Identify your "game state"

First, we need to decide whether the game mainly takes place in `MyActivity` (and then you can click a button or menu control to switch to `SettingsActivity`), or perhaps instead, `MyActivity` is a sort of "main menu," that can lead both to the `SettingsActivity` or the `GameActivity`.

In the latter case, create a new Activity class and XML layout, and then add a second button to your `MyActivity` that will lead to the new `GameActivity`, as illustrated below. (The technique is the same as transitioning to the `SettingsActivity` described in Milestone 3.)

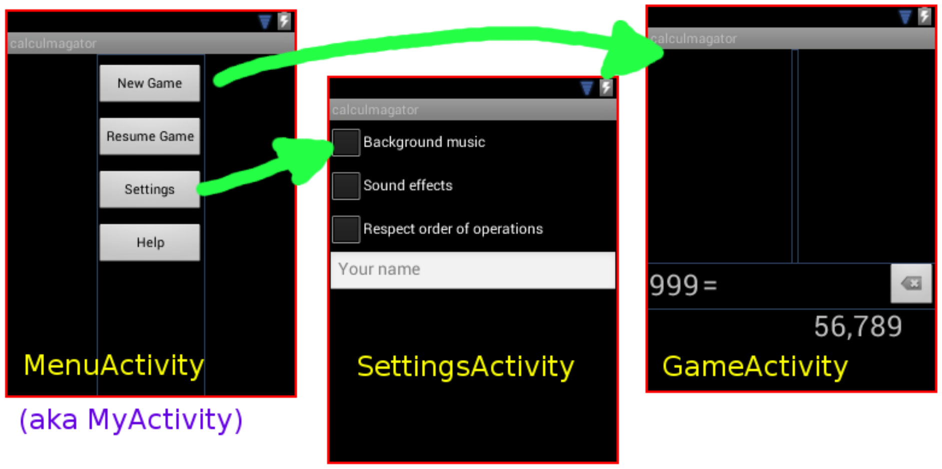

Figure 1:

Now, in `GameActivity`, you want to figure out what variables will be needed to represent your entire "game state." That means that, if a phone call comes in, or the user tilts the phone's orientation, or switches temporarily to another app — what will we need to store so that the game can continue seamlessly?

Here are some sample questions, to assist your brain-storming about game state: does the game keep score? Do multiple players take turns? Is the player working within a particular level? Are there positions of different characters or other game elements on the screen? Are certain elements of the game disabled, or do they have other states?

You will want to transform all of this game state into a set of *instance variables* in the `GameActivity` class. Each instance variable must have a Java type, such as `int`, `float`, `String`, or an array of any of these. If you have more complicated types, that should be okay, as long as they can be converted to a `String` and then read back from a `String`.

## Show & manipulate game state

Next, you want to put some widgets on the `GameActivity` screen that can display (part of) the game state, and allow the user to manipulate that game state in certain ways. This will by no means be a complete game, but just a starting point. A dirt-simple initial draft is to use a `TextView` that displays the current score, and a `Button` that increases the score.

To help you make the game a bit fancier, I will be posting notes on the following:

- Using 'drawables' and sub-classing `ImageView`
- Drawing your own widgets – see MyView in the Canvas notes.
- Positioning widgets

If you have the dirt-simple version working and feel you need help with some additional way to manipulate the game state, do get in touch with your ideas and goals.

## Save & restore game state

This is similar to how we save preference values in the `SettingsActivity`, but it's a slightly different API. You save the game state into what is called a `Bundle`. Whenever Android wants you to save the state of the activity (maybe because a phone call comes in, or the user is switching away to something else), it will call a method `onSaveInstanceState` that you can override:

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    // save your game state here...
}
```

Like `SharedPreferences`, the `Bundle` class has methods such as `putInt`, `putFloat`, `putString`, `putIntArray`, `putStringArray`, etc. You define key strings to which the data will be mapped; I recommend defining them as `static final` variables in your class.

Restoring the game state is part of the job of `onCreate` which we've seen before. It is given a `Bundle` that is usually called `savedInstanceState`. If that bundle is `null`, then there is no state existing, so we should create a new game. Otherwise, we can use methods like `getInt`, `getFloat` to retrieve and restore the game state from the bundle.

A complete example follows, with just one instance variable representing the game state – the current score.

```
public class GameActivity extends Activity
{
    static final String scoreKey = "net.liucs.mygame.score";

    int score;  // Game state

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        if(savedInstanceState == null) {  // Create new game
            score = 0;
        } else {   // Restore previous game
            score = savedInstanceState.getInt(scoreKey);
        }
    }

    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putInt(scoreKey, score);
    }
}
```

With all of this in place, you should be able to start a new game, view and manipulate the game state, then leave the game and return to it in the same state you left it.