CIS 201 Fall 2008: Lab 7

October 7, 2008

- Implementing a class
- Using `CompositeSprite`
- Practice using `ArrayList`
- Timed-animation
- Game components with "state"

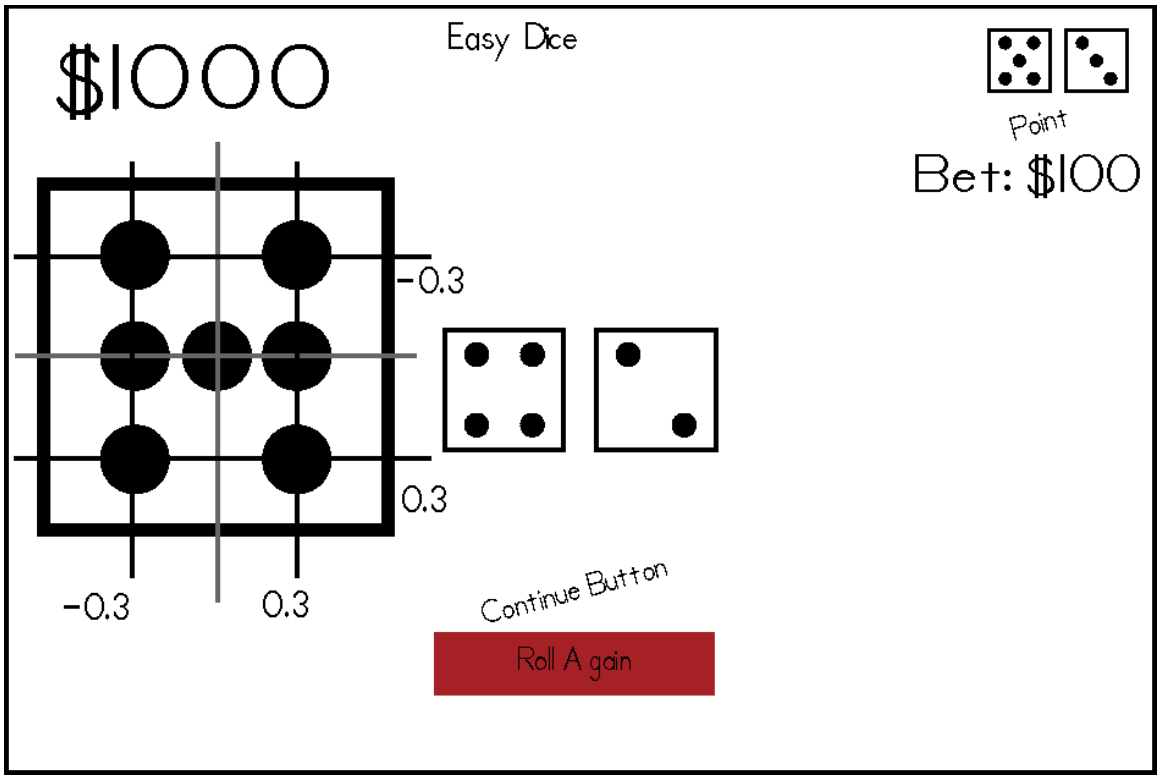

Figure 0.1: EasyDice Design

Checkpoint 1. Download the game `EasyDice.java`.
Create a directory, `Lab7` in the `CS1` directory. Into that directory, download the game `EasyDice.java`, available on Moodle or at `http://cs.potsdam.edu/faculty/laddbc/Teaching/CS1/EasyDice.java`.
Look at the contents of the game. It is functional (though you will have to modify it slightly) *except* that it relies on a class, `OneDie`, that does not exist. Your job, in this lab, is to create the `OneDie` class so that `EasyDice` can be played.
Create a new, empty class `OneDie`; `OneDie` extends `CompositeSprite`. It should be in the same directory as `EasyDice.java`. Insert a header comment for the file with both author's names.
Look at the code for `EasyDice`.
In the header comment for `OneDie`, write the signature for all methods called on any object of type `OneDie` in the game. You may want to label these as the interface methods for the class (in the comment). You know that they must all have *what* access rights? (Hint: The class `EasyDice` must be able to call them from outside the class `OneDie`.)
Don't forget the constructor. It, too, is a method. Include signatures for any constructors used in `EasyDice`.
Before continuing, raise your hand and show the instructors your list of methods.

Checkpoint 2. Split your list.
Take your list, look at the documentation for `CompositeSprite`, and divide your list of methods into two: the first part should be methods defined in `CompositeSprite` or one of its super classes; the second part, separated by a blank line, should be the methods not defined in any super class of `OneDie`.
You will need to implement the "new" methods in your class. You will extend the `CompositeSprite` so you get all of its methods for "free".

Checkpoint 3. Implement stubs.
A "stub" is a method that doesn't actually do anything. It is used to permit code to compile while you actually work on getting the code to function. The stub of a **`void`** method (or a constructor) is easy: it is empty. There is no need to return any value since none is expected.
A stub for a method that returns a value just returns some value. For **`boolean`** you can just return **`false`** and for **`int`** you can just return `0`. You could do the same for **`double`** or **`char`**. Stubs that are supposed to return objects are a bit trickier but you don't have any of those here.
Put stubs for all your functions into the `OneDie` class and get `EasyDice` to compile. You can run it, too, and you won't see much.
Show your instructors that you have it compiling.

Checkpoint 4. Change the color and size of the game. While the game works in the default size of the game, it would work better if the game were larger. Modify `EasyDice` to have a default constructor (a constructor with no parameters; this is the constructor FANG calls) that sets the size of the game to (480, 520) pixels (the extra height is for the buttons below the play field).

Checkpoint 5. Add fields
Your dice will need fields. You need to keep track of the value shown on the die, the game the die is in (for `randomInt`), and the sprites that are the pips and the white background.
As seen in the picture, you will need 7 pips; you can use the `setVisibility` method to turn them on or off depending on the face that is showing.

Checkpoint 6. Add the sprites.

In the constructor for your dice, you should add eight sprites to them. The background is a white rectangle, centered at 0, 0 size 1.0. The pips are 0.2 by 0.2 and centered as shown in the drawing. Note that `addSprite` in the `CompositeSprite` will add the sprites. The `CompositeSprite` is centered at 0, 0.
Adding the `CompositeSprite` will add all of the sprites you added to it to the scene.

Checkpoint 7. Implement get/set for value
Implement `setValue`. It should check which number (between 1 and 6) that value is being set to and set the value field *and* turn on the right pips. You probably want to turn all the pips off first and then turn on just the ones you want.
Have `getValue` return the value set.

Checkpoint 8. Roll the dice.
`roll` takes a time parameter, the amount of time you want the die to roll. It is 2.0 seconds in our case. When I call `roll`, change the face (randomly...this is why you need the `Game`) and start counting down. Every eighth to quarter of a second, change the face. This means you will be running two different countdown timers, one for the current face and one for the roll.
You will need an `advance` method (like the `move` method in `BallSprite`) so that the timers can be updated. Your logic will look like this:

```
if (animationTimer > 0) {
  // decrement animationTimer
  // decrement faceTimer
  if (faceTimer < 0) {
    // show new random face
    faceTimer = // time to show one face
}
```

Also fix up `isRolling` to return true so long as there is time remaining in the current die roll.

Checkpoint 9. Upload Your Java files.
Make sure both authors' names are in all of the header comments!
Go to Moodle; you will find an assignment in week 7 labeled **Lab7**. Go there and upload all of the Java files you created/modified in lab today. Note that there will be other files in your `Lab7` directory. You want to make sure you upload just the `.java` file.