CIS 201 Computer Science I
Fall 2009 Lab 06

October 6, 2009

- Build a solution with multiple classes.
- Create new instances of your class and use them in a game.
- Create a class which has state (and implement getter and setter methods).

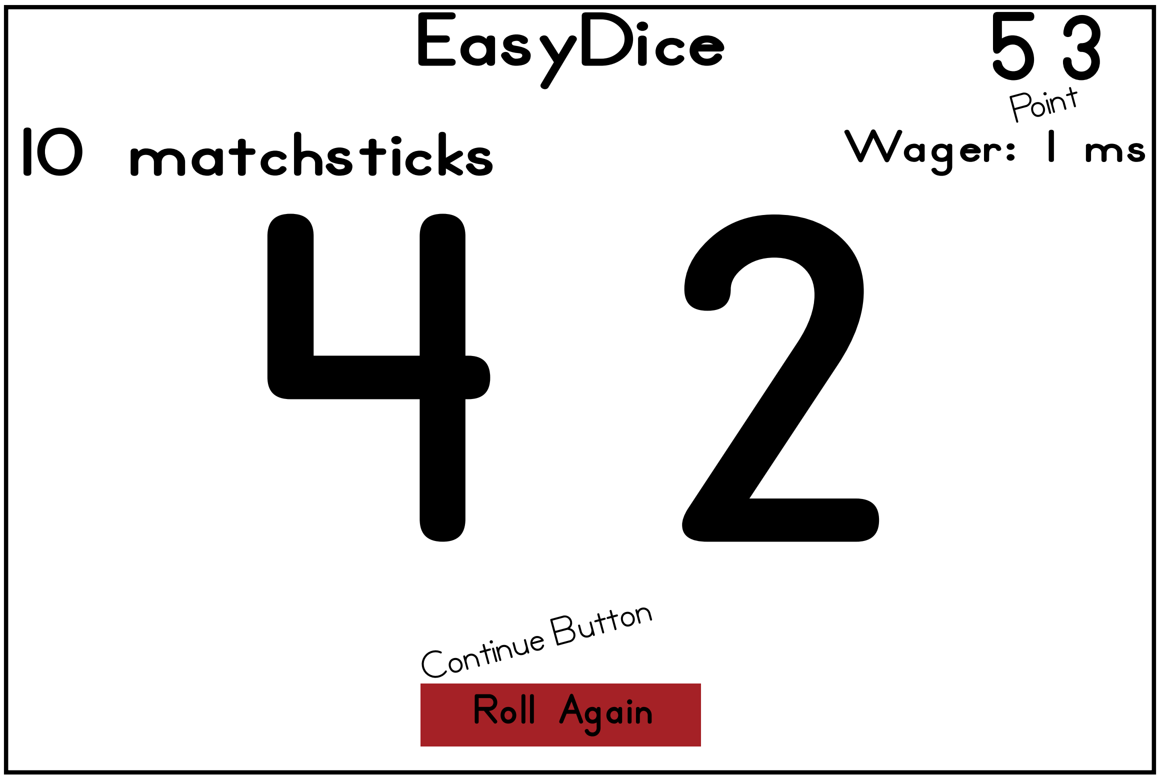

Figure 0.1: EasyDice Design

Checkpoint 1 Getting Started

Create a directory, `Lab06` in your CS1 directory. Copy `EasyDice.java` and `EasyButton.java` (location: `/home/student/Classes/201/Labs/06/EasyDice.java` `/home/student/Classes/201/Labs/06/EasyButton.java`) into the new lab directory.

`EasyDice.java` contains a complete game. Take a look at the code and you will find that one class, `OneDie`, does not exist. Your job is to design and build that class so that `EasyDice` can be played.

Create a new, empty class called `OneDie`. The class `extends StringSprite`. It must be in the lab directory (the same directory as `EasyDice`).

Insert a properly formatted header comment for the `OneDie` class and include both author's names.

Examine the `EasyDice` code. In the header comment for `OneDie`, note each `OneDie` method called by `EasyDice`. Deduce as much of the method header as you can for each one. (Don't forget the *constructor*; it is a method, too.)

Show your work on Checkpoint 1 to the lab monitor, answering any necessary questions for them. Have them sign before continuing.

Checkpoint 2 Implement Stubs

Compare your list of methods of `OneDie` with the documentation for `StringSprite` and divide the list in two: those methods already defined in `StringSprite` and those that are not. You get all the methods in the first list by extending `StringSprite`; you must define the ones in the second list.

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 some number in the expected range. 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 work on Checkpoint 2 to the lab monitor, answering any necessary questions for them. Have them sign before continuing.

Checkpoint 3 Add State

The big extension of `OneDie` over `StringSprite` is that the die has a face value. Add a field to hold the value.

Add comments to the getter and setter methods for the die's face value. Also modify them so that they interact with the state you added.

Modify the setter so that the displayed text of `OneDie` is updated if the value is changed. Also make sure that the face value of the die is only changed if the number is in the range [1-6].

Why would you worry about an out of range face value?

Show your work on Checkpoint 3 to the lab monitor, answering any necessary

questions for them. Have them sign before continuing.

Checkpoint 4 Roll the Dice

The roll method of `OneDie` must pick a random integer on the range [1-6]. How do we get a random number?

If you said "Call `randomInt`," give yourself a cookie. There is a problem: `StringSprite` does not have a `randomInt` method. That method is defined in the `Game` class.

How can we get the currently running game object? It is not enough to get the `Game` class; we need an actual `Game` object. What to do... `Game.getCurrentGame()` is a method defined for the `Game` class which returns a reference to the current game. We can use it to call `randomInt` (what parameters do you pass to `randomInt` to get numbers on the range [1-6]?).

Write `roll` so that it rolls `OneDie`.

Compile, run, and play a little bit of the game (to convince yourself that it works).

Be prepared to explain the line which calls `randomInt`, both the parameter(s) and the `Game.getCurrentGame()` part.

Show your work on Checkpoint 4 to the lab monitor, answering any necessary questions for them. Have them sign before continuing.

Log off of the lab computer you are using before leaving they lab. Anyone entering the lab has unlimited access to your files if you remain logged on. **DO NOT** turn off lab computers! They are a shared resource and there might be someone else logged in to "your" machine.

Clean up your work area, push in the chair, and have a good week.