Event Driven Computing - Assignment 1

July 26, 2007

1 Objective

This assignment is intended to make you familiar with the problems posed by event-driven systems. You will also build a simple Graphical User Interface. Some of you may not be familiar with the SVN version management software, so this practical exercise provides an opportunity for you to use it. The key for this exercise is: **edc-07-s2-assign1**. If you are unfamiliar with SVN, a document describing it is available on the web at:

http://www.cs.adelaide.edu.au/docs/svn-instr.pdf

2 Description

2.1 Simple rection controller

You have been hired by an electronics company to build the software for a simple Reaction-timer game.

The reaction timer has two inputs: a *coin-slot*, to start the game, and a *start/stop* button that plays the game. There is also a display that indicates to the player what he should do next.

The machine behaves like this: Initially, the display shows **insert coin**, and waits for a player to do so. When the player inserts a coin, the machine displays **press GO!**, and waits for the player to do so. When the player presses the go/stop button, the machine displays **Wait...** for a random time between 1.0 and 2.5 seconds. After the random delay expires, the machine displays a time-value that increments every 10 milliseconds, starting at zero. The player must now press the go/stop button as soon as possible — the object of the exercise is (of course) to have *fast* reactions! When the player presses go/stop, the machine displays the final timer value for three seconds, then the game is over until another coin is inserted.

If the player presses the go/stop during the random delay period (i.e. the player tries to "guess" when the delay will expire, the machine aborts the game, and immediately demands another coin. (No reward for trying to cheat!)

During the period when the machine is displaying the timer value, the machine automatically presses the go/stop button when the timer reaches two seconds.

If the player presses go/stop while the measured reaction-time is being displayed, the machine immediately displays “insert coin”.

3 Components of your program

Your program will consists of three parts:

- A controlling program, that we provide;
- A SimpleReactionController, that you will write. This component must conform to the interface $Controller$;
- A Display (gui component) that you will write. It must conform to the Gui interface.

To ensure that we can automatically test your program, you *Must* comply with the following (minor) constraints:

Obviously, your program needs a source of timing information. You *must* obtain this information by implementing the $tick$ method in your controller. The ReactionMachine class guarantees to call this method every 10 milliseconds.

Your program also needs a source of random numbers. You *must* obtain your random numbers by calling the $getRandom()$ method in the interface $Random$.

3.1 SimpleReactionController component

This is the main body of the exercise. You are free to write this component in any way you choose, so long as it implements the required interfaces.

3.2 Display component

You are required to build a simple GUI to allow you to test your reaction machine. The GUI must have a button labelled *Coin inserted*, and a button labelled *go/stop*.

There must be a display region to show the messages of the machine. Times must be displayed in decimal notation. For example, a value of 1.5 seconds must be shown as “1.50”

Do not go to too much effort to make the display “beautiful”. We are more concerned here with correct operation than we are with your artistic ability!

4 Enhanced reaction controller

After showing the basic reaction timer machine to potential customers, the marketting department has found that some would be prepared to pay for a deluxe reaction timer that allowed multiple games after payment of a single coin.

In this machine, when the coin is inserted, the player is allowed to play three games. The operating sequencee is very similar to before: insert coin, press go/stop, wait random delay, press go/stop, then display the time for three seconds. If the machine has not yet completed three games, it displays: **Wait**, then waits the random delay, etc.

After displaying the time for the last game, the machine displays **Average time= t.tt** for five seconds, then the game is over.

If after inserting the coin, the player fails to prevent go/stop within ten seconds, the game is over.

If the player presses go/stop during the waiting period, the game is aborted, and the average value is *not* displayed. (Once again, no reward for cheating!)

If the player presses go/stop while the machine is displaying a reaction-time value, the machine immediately moves on to the next game, without waiting for the full three seconds.

If the player presses go/stop while the machine is displaying the average time, the game is immediately over.

You are asked to incorporate these feature to make a new product called *EnhancedReactionController*.

5 What we give you

We provide the following java files for you, in a zip-file named `assign1.zip`

Controller.java contains an interface specification. Both your reaction controllers *must* implement this interface.

Gui.java contains an interface specification. Your GUI *must* implement this interface.

Random.java contains an interface specification for the random number generator.

ReactionMachine.java contains a driver program that will allow you to test your reaction controllers.

Please be aware that you *cannot* alter these files, or we will not be able to test your program.

You must write three java classes: *SimpleReactionController.java* that functions like the simple reaction machine described above. *Display.java* that implements the necessary GUI functions; and *Enhanced ReactionController.java* that behaves like the enhanced multi-game reaction machine.

Your display and two controllers *must* work correctly with ReactionMachine.

6 Getting the files

The following Unix commands will get a copy of the files, and create a repository for this assignment. Please note the following:

- Perform these steps in the order written*ONCE*!
- Replace *a123456*, where it appears in the commands, with YOUR student id.
- Some commands are long - they must be typed on *one* line.

Use the Unix “cd” command to change to the place where you want your exercise to be stored, then type these commands:

```
svn mkdir -m"created" https://repos.cs.adelaide.edu.au/svn/a123456/edc-07-s2-assign1
svn co https://repos.cs.adelaide.edu.au/svn/a123456/edc_07_s2_assign1/ edc-07-s2-assign1
```

now unpack the assign1.zip file, then

```
svn add *.java

svn commit -m "initial import"
```

You are now ready to commence working on the exercise.

7 Running the driver

You can run the driver program by typing:

`java ReactionMachine` *cccc*

where *cccc* is the name of the controller you want to test. For example, `java Reactionmachine SimpleReactionController` will run the SimpleReactionController program.

You are, of course, free to modify this program, or build another one to help you with your testing.

8 Assessment

We will test the behaviour of your two controllers using an automated test script. It is therefore vital that you correctly implement the interfaces. We will also read your program, and award marks for neat, well documented, code.

Since this is a level-three subject, we will *not* provide access to our tests *before* the handin deadline. The handin mechanism will perform only rudimentary tests to verify that nothing is catastrophically wrong. We therefore encourage you to thoroughly test your code before the deadline!

The deadline for this assignment 5PM Friday 10th August. Late handins will be penalised 25% per day (including weekends!).

David Knight
Travis Olds

26-July-2007