Lab: Working with Structures and Universe Programs

COMP 50

Fall 2013

Prelude

This lab has two purposes:

- To give you hands-on experience writing Universe (`big-bang`) programs that interact with the mouse.
- To give you practice using structures (definition by parts) and state machines (definition by choice) to build Universe programs.

Both parts of the lab will prepare you for the interactive user-interface experiment that you will build for your homework.

Structure of the lab

As in most labs, you will work with a partner. Tom Williams will have your partner assignment.

The lab will begin with a demonstration of two programs: a circle-dropping program and a line-drawing program. You will have time to build the full circle-dropping program and to start a prototype of the line-drawing program.

Designing Universe programs

Modeling the state of the world

In any `big-bang` program, your key to success is the *world-state*. The *world-state* determines what you can draw, and it also determines how the mouse and the keyboard are supposed to behave.

1. Working with your partner, think about how you want to represent the *state* of the circle-dropping program. How will you represent the state of the circle-dropping world? If there is more than one state, draw a state diagram like the one we used for the traffic light.

   Write down examples of the world state.

2. Working with your partner, think about how you want to represent the *state* of the line-drawing program. How will you represent the state of the line-drawing world? If there is more than one state, draw a state diagram like the one we used for the traffic light.

   Write down examples of the world state.

Show your state diagrams and your examples to the lab staff.

Interacting with the mouse

These programs don't have any animations that run by themselves. Instead the movement of lines or other things on the screen is driven by interaction with the mouse and/or keyboard. In addition to the `to-draw` method of `big-bang`, which is needed in every `big-bang` program, you will begin with one or two other methods on your wish list:

- The `on-mouse` method is a *mouse event handler*. It reacts both to mouse movement and to mouse clicks. You will need to look up the documentation of *mouse events* in the `2htdp/universe` teachpack and to analyze the conditions under which you think a mouse event is interesting. In different programs you may well be interested in different conditions.

  All mouse-event handlers have the same signature. Here it is, along with a very generic purpose statement and a simple header:

```
; meh : world-state number number mouse-event -> world-state
; based on the given state, to react appropriately to the given
; mouse event at the given (x, y) coordinates, and to return
; the new world state that results
(define (meh world x y event)
  ...)
```

- The line-drawing program stops when the `q` key is pressed. To implement this behavior you will need to implement the `on-key` method.

  Here is a generic key-event handler:

```
; keh : world-state key-event -> world-state
; based on the given state, to react appropriately to the given
; key event, and to return the new world state that results
(define (keh world a-key)
  ...)
```

To understand when the program is over you will have to learn about the `stop-when` method, and you will have to *design a data representation* that includes the idea of a "final" or "stop" state.

Develop your wish list

Your wish list will include a rendering function (passed to `to-draw`), a mouse-event handler (passed to `on-mouse`), and possibly a keyboard-event handler. Your wish list should also include whatever functions you think you might need to change the world state in response to mouse events and keyboard events. *Every wish-list function needs a signature and a purpose statement.*

Write down your wish list!

Designing the bricks

Now you're ready to tackle the "bricks":

1. Begin with your data descriptions and data examples. You should already have a data definition for the world state, but you may not others. Use structure definitions where needed.

   Write data examples!

   Have the lab staff check your data descriptions.

2. Make sure that every function on your wish list has a signature, purpose statement, and header.

   Have the lab staff check your signatures, purpose statements, and headers.

3. Using the signatures, purpose statements, and headers of the functions on your wish list, start devising functional examples.

4. Create function templates. Use the template for conditionals and the template for structures.

5. Code

6. Test

7. Run

Ten minutes before the end of the lab, put the following text at the beginning of a DrRacket file. You may use an empty file or the source code you have been working on:

```
#|
What I did during this lab:
   (you fill in this part)

What I learned during this lab:
   (you fill in this part)

|#
```

The lab staff will help you articulate what you learned.

Finally, *submit this file through the handin server* as `lab2`. You will need to submit it using *two* usernames connected by a + sign, as in

```
Jane.Doe+Richard.Roe
```

You submit using Jane Doe's password.