CS 5010: Problem Set 8

Out: Tuesday, November 20, 2012

Due: Wednesday, November 28, 2012 at 600pm.

The goal of this problem set is to help you design simple class systems and methods on them.

As always, you must follow the design recipe.

- As part of the data analysis step, you must create a UML class diagram that can be displayed during the codewalk. You may either create this in ASCII or using your favorite diagramming tool. If you use a tool, you must deliver it in pdf format.
- You should have no structs in your solution: every struct should be replaced by a class.
- Be sure to write an interface for every kind of enumeration or mixed data that you introduce, with a few exceptions:
    - You can treat key events and mouse events as ordinary data definitions, and define alternate enumerations (like AnimationKeyEvent) as you have in the past.
    - Similarly, if you need a list of indefinite length, you can use a Racket list (with ordinary data definitions); you do not need to turn this into an interface and two classes.
- Be sure to write out a purpose statement for each class and an interpretation for each field.
- Write out a contract and purpose statement for each method. For methods declared in an interface, do this there. For methods that are not declared in an interface, do it along with the method definition in the class.
- Specify the design strategy for each method.

Please restrict yourself to the language features discussed in class. In particular, you may NOT use inheritance.

You must use `#lang racket`, `rackunit`, `rackunit/text-ui`, `2htdp/universe`, and `2htdp/image`. So your file should begin something like

```
#lang racket
(require rackunit)
(require rackunit/text-ui)
(require 2htdp/universe)   ; only if you are doing an animation
(require 2htdp/image)      ; only if you are manipulating images
```

You may not `require` any other libraries.

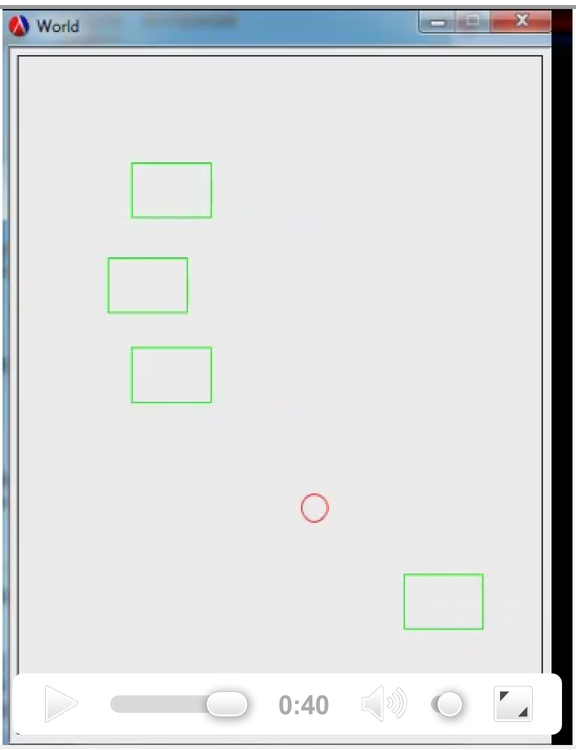
---

1. This problem combines elements of Problem Set 2 Question 2 (Draggable Images) and Problem Set 2 Question 3 (Balls in a Box).

Design and implement a system similar to the ones above, but using classes and methods. The specification is as follows:

- The canvas is 400x500 pixels.
- On the canvas, the system displays a circle of radius 10 in outline mode. The circle initially appears in the center of the canvas. We call this circle the "target"
- The user can move the target around the screen by dragging it with the mouse. During a drag, the position of the mouse *within* the target should stay the same. For example, if the user drags the target by grabbing it near its edge, the mouse should stay near the edge as the mouse moves. Thus the target will move continuously following the mouse. Unlike ps02q2, there is no requirement to display the position of the mouse within the target.
- In addition, the system displays several rectangles. Each rectangle is 30x20 outline.
- When the user types "n", a new rectangle appears, with its center located at the center of target, travelling right.
- Each rectangle travels at a constant rate. When the edge of the rectangle reaches the edge of the canvas, it starts again in the opposite direction, from the edge, as in Problem Set 1.
- The user can move the rectangles around the screen by dragging them with the mouse. This should be a "smooth drag", like the dragging behavior of the target above.

Here's a short video demo:



Your solution should provide the following functions and classes:

```
World% -- a class that satisfies the World<%> interface (shown below).
Rectangle% -- a class that satisfies the Rectangle<%> interface
(shown below).
```

```
make-world : Number -> World%
Creates a world with no rectangles, but in which any rectangles
created in the future will travel at the given speed.

run : Number Number -> World%
Given a frame rate (in seconds/tick) and a speed (in pixels/tick), runs
the world.  Returns the final state of the world

Interfaces:

(define World<%>
  (interface ()

    ;; -> World<%>
    ;; Returns the World<%> that should follow this one after a tick
    on-tick

    ;; Number Number MouseEvent -> World<%>
    ;; Returns the World<%> that should follow this one after the
    ;; given MouseEvent
    on-mouse

    ;; KeyEvent -> World<%>
    ;; Returns the World<%> that should follow this one after the
    ;; given KeyEvent
    on-key

    ;; Scene -> Scene
    ;; Returns a Scene like the given one, but with this object drawn
    ;; on it.
    add-to-scene

    ;; -> Number
    ;; Returns the x and y coordinates of the target
    get-x
    get-y

    ;; -> Boolean
    ;; Is the target selected?
    get-selected?


    ;; -> ListOf<Rectangle<%>>
    get-rectangles

))

(define Rectangle<%>
  (interface ()

    ;; -> Rectangle<%>
    ;; Returns the Rectangle<%> that should follow this one after a tick
    on-tick

    ;; Number Number MouseEvent -> Rectangle<%>
    ;; Returns the Rectangle<%> that should follow this one after the
    ;; given MouseEvent
    on-mouse

    ;; KeyEvent -> Rectangle<%>
    ;; Returns the Rectangle<%> that should follow this one after the
    ;; given KeyEvent
    on-key

    ;; Scene -> Scene
    ;; Returns a Scene like the given one, but with this object drawn
    ;; on it.
    add-to-scene

    ;; -> Number
    ;; Return the x and y coordinates of the center of the rectangle.
    get-x
    get-y

    ;; -> Boolean
    ;; Is the rectangle currently selected?
    is-selected?

))
```

Note: these interfaces list only the methods that we need in order to test your program; your classes are likely to have other methods as well.

When you do this problem, remember the principle of Iterative Development: get something simple working, and then add features as necessary. For what it's worth, my World% class was 168 lines, of which 62 lines were for dragging, and my Rectangle% class was 130 lines, of which 41 lines was for ordinary (undragged) motion, and 46 was for dragging.

Last modified: Tue Nov 20 14:26:31 -0500 2012