

# CS5010 - Problem Set 10 - Test Results

pdp-group-dantuluru-rosefox911

March 31, 2014

This test suite tests your implementation of Problem Set 09

## 1 File: 1.rkt

Tests your bouncing rectangles problem

Common Definitions

```
(define CX 200)
```

```
(define CY 250)
```

```
(define SPEED 13)
```

```
(define LBORDER 15)
```

```
(define RBORDER 385)
```

```
(define check-world-selected?  
  (lambda (t w) (equal? t (send w get-selected?))))
```

```
(define check-world-x (lambda (t w) (equal? t (send w get-x))))
```

```
(define check-world-y (lambda (t w) (equal? t (send w get-y))))
```

```
(define check-rectangle-selected?  
  (lambda (t r) (equal? t (send r is-selected?))))
```

```
(define check-rectangle-x (lambda (t r) (equal? t (send r get-x))))
```

```

(define check-rectangle-y (lambda (t r) (equal? t (send r get-y))))

(define check-rectangles
  (lambda (lst w)
    (let ((rmap (map map-rect-components (send w get-shapes))))
      (if (or (equal? lst rmap) (equal? (reverse lst) rmap))
          #t
          rmap))))

(define map-rect-components
  (lambda (r)
    (list (send r get-x) (send r get-y) (send r is-selected?))))

(define check-rectangle
  (lambda (x y sel?)
    (lambda (w)
      (and (check-rectangle-x x w)
            (check-rectangle-y y w)
            (check-rectangle-selected? sel? w)))))

(define check-world
  (lambda (x y sel? w)
    (and (check-world-x x w)
          (check-world-y y w)
          (check-world-selected? sel? w))))

(define rectangle-equal?
  (lambda (r1 r2)
    (and (equal? (send r1 get-x) (send r2 get-x))
          (equal? (send r1 get-y) (send r2 get-y))
          (equal? (send r1 is-selected?) (send r2 is-selected?)))))

(define world-equal?
  (lambda (w1 w2)
    (and (equal? (send w1 get-x) (send w2 get-x))
          (equal? (send w1 get-y) (send w2 get-y))
          (equal? (send w1 get-selected?) (send w2 get-selected?))
          (andmap
            rectangle-equal?
            (send w1 get-shapes)
            (send w2 get-shapes)))))

```

```

(define simulate-until-at-wall
  (lambda (right? speed w)
    (let ((cur-x (send (first (send w get-shapes)) get-x)))
      (cond
        ((or (> cur-x RBORDER) (< cur-x LBORDER))
         (error "Moved past the edge"))
        ((and right? (or (= cur-x RBORDER) (= cur-x (- RBORDER 1)))) w)
        ((and (not right?) (= cur-x LBORDER)) w)
        (else
         (let* ((a-tick (send w on-tick))
                (next-x (send (first (send a-tick get-shapes)) get-x)))
           (if (or (and right? (> next-x cur-x))
                   (and (not right?) (< next-x cur-x)))
               (if (or (equal? (abs (- next-x cur-x)) speed)
                       (= next-x RBORDER)
                       (= next-x (- RBORDER 1))
                       (= next-x LBORDER))
                   (simulate-until-at-wall right? speed a-tick)
                   (error "Does not move at full speed when it should"))
               (error "Does not move towards correct wall"))))))))

```

## 1.1 Test-Group: Basic (non-)reactions (1 Points)

1/1

Common Definitions

```

(define INITIAL-WORLD (make-world-1 SPEED))

```

### 1.1.1 Test (equality)

On tick has no effect

Input:

```

(world-equal? INITIAL-WORLD (send INITIAL-WORLD on-tick))

```

Expected Output:

```

#t

```

Expected Output Value:

```

#t

```

Correct

### 1.1.2 Test (equality)

On tick has no effect

Input:

```
(world-equal?  
  INITIAL-WORLD  
  (send INITIAL-WORLD on-mouse 10 10 "button-down"))
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.1.3 Test (equality)

On tick has no effect

Input:

```
(world-equal?  
  INITIAL-WORLD  
  (send INITIAL-WORLD on-mouse CX CY "drag"))
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.1.4 Test (equality)

On tick has no effect

Input:

```
(world-equal? INITIAL-WORLD (send INITIAL-WORLD on-key "d"))
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

## 1.2 Test-Group: Dragging the target (3 Points)

Common Definitions

```
(define INITIAL-WORLD (make-world-1 SPEED))

(define SELECTED-WORLD
  (send INITIAL-WORLD on-mouse (+ CX 3) (- CY 4) "button-down"))

(define DRAGGED-WORLD (send SELECTED-WORLD on-mouse 50 150 "drag"))

(define DRAGGED-WORLD-2 (send DRAGGED-WORLD on-mouse 300 25 "drag"))

(define RELEASED-WORLD
  (send DRAGGED-WORLD-2 on-mouse 300 25 "button-up"))
```

### 1.2.1 Test (equality, 1 partial points)

Select the target

Input:

```
(check-world CX CY true SELECTED-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.2.2 Test (equality, 1/2 partial points)

Drag the target

Input:

```
(check-world 47 154 true DRAGGED-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.2.3 Test (equality, 1/2 partial points)

Further Drag the target

Input:

```
(check-world 297 29 true DRAGGED-WORLD-2)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.2.4 Test (equality, 1 partial points)

Release the target

Input:

```
(check-world 297 29 false RELEASED-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

## 1.3 Test-Group: Basic ball behavior (2 Points)

Common Definitions

```
(define INITIAL-WORLD (make-world-1 SPEED))
```

```
(define BALL-WORLD (send INITIAL-WORLD on-key "r"))
```

```
(define BALL-WORLD-1 (send BALL-WORLD on-tick))
```

```
(define BALL-WORLD-2 (send BALL-WORLD-1 on-tick))
```

### 1.3.1 Test (equality, 1 partial points)

On pressing n

Input:

```
(check-rectangles '((,CX ,CY ,false)) BALL-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.3.2 Test (equality)

After one tick

Input:

```
(check-rectangles '((, (+ CX SPEED) ,CY ,false)) BALL-WORLD-1)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.3.3 Test (equality, 1 partial points)

After two ticks

Input:

```
(check-rectangles '((, (+ CX SPEED SPEED) ,CY ,false)) BALL-WORLD-2)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

## 1.4 Test-Group: Bouncing (3 Points)

Common Definitions

```
(define INITIAL-WORLD (make-world-1 SPEED))

(define BALL-WORLD (send INITIAL-WORLD on-key "r"))

(define BALL-RBORDER (simulate-until-at-wall true SPEED BALL-WORLD))

(define BALL-RBORDER-1 (send BALL-RBORDER on-tick))

(define BALL-LBORDER
  (simulate-until-at-wall false SPEED BALL-RBORDER-1))
```

### 1.4.1 Test (or)

Right bounce

**Test (equality)**

Ball bounce from right

Input:

```
(check-rectangles '(((- RBORDER 1) ,CY ,false)) BALL-RBORDER)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Wrong Output:

```
((385 250 #f))
```

**Test (equality)**

Ball bounce from right

Input:

```
(check-rectangles '((,RBORDER ,CY ,false)) BALL-RBORDER)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct



### 1.4.2 Test (equality, 1 partial points)

Ball bounce from left

Input:

```
(check-rectangles '((,LBORDER ,CY ,false)) BALL-LBORDER)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

## 1.5 Test-Group: Dragging rectangle (4 Points)

Common Definitions

```
(define INITIAL-WORLD (make-world-1 SPEED))

(define BALL-WORLD (send INITIAL-WORLD on-key "r"))

(define SELECTED-BALL-WORLD
  (send BALL-WORLD on-mouse (+ CX 12) (- CY 5) "button-down"))

(define DRAG-RECT (send SELECTED-BALL-WORLD on-mouse 50 150 "drag"))

(define DRAG-RECT-2 (send DRAG-RECT on-tick))

(define 2-RECTS (send DRAG-RECT-2 on-key "r"))

(define 2-RECTS-DRAG (send 2-RECTS on-mouse 300 25 "drag"))

(define 2-RECTS-RELEASE
  (send 2-RECTS-DRAG on-mouse 300 25 "button-up"))
```

### 1.5.1 Test (equality, 1/2 partial points)

Create the rectangle

Input:

```
(check-rectangles '((,CX ,CY ,false)) BALL-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.5.2 Test (equality, 1/2 partial points)

Select the rectangle

Input:

```
(check-rectangles '((,CX ,CY ,true)) SELECTED-BALL-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.5.3 Test (equality, 1/2 partial points)

Drag the rectangle

Input:

```
(check-rectangles '((,38 ,155 ,true)) DRAG-RECT)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

#### 1.5.4 Test (equality, 1/2 partial points)

Tick after dragging should not affect the rectangle

Input:

```
(check-rectangles '((,38 ,155 ,true)) DRAG-RECT-2)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

#### 1.5.5 Test (equality, 1/2 partial points)

Create a new rectangle

Input:

```
(check-rectangles '((,CX ,CY ,false) (38 155 ,true)) 2-RECTS)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

#### 1.5.6 Test (equality, 1/2 partial points)

Drag the selected rectangle

Input:

```
(check-rectangles '((,CX ,CY ,false) (288 30 ,true)) 2-RECTS-DRAG)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.5.7 Test (equality, 1/2 partial points)

Release the selected rectangle

Input:

```
(check-rectangles '((,CX ,CY ,false) (288 30 ,false)) 2-RECTS-RELEASE)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.5.8 Test (equality, 1/2 partial points)

Target should not have moved

Input:

```
(check-world CX CY false 2-RECTS-RELEASE)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

## 1.6 Test-Group: Special Cases (1 Points)

Common Definitions

```
(define INITIAL-WORLD (make-world-1 SPEED))

(define BALL-WORLD (send INITIAL-WORLD on-key "r"))

(define SELECT-TARGET
  (send BALL-WORLD on-mouse (+ CX 12) (- CY 5) "button-down"))

(define DRAG-TARGET (send SELECT-TARGET on-mouse 2 2 "drag"))
```

```

(define DRAG-TARGET-2
  (send DRAG-TARGET on-mouse (+ CX 12) (- CY 5) "drag"))

(define RELEASE-TARGET
  (send DRAG-TARGET-2 on-mouse (+ CX 12) (- CY 5) "button-up"))

(define S-TARGET-2 (send INITIAL-WORLD on-mouse CX CY "button-down"))

(define DRAG-2 (send S-TARGET-2 on-mouse 50 50 "drag"))

(define DRAG-2-N (send DRAG-2 on-key "r"))

```

### 1.6.1 Test (equality, 1/2 partial points)

Even when we would drag the rectangle outside of the canvas, normal behavior should resume when we get back inside of the canvas

Input:

```
(check-rectangles '((,CX ,CY ,false)) RELEASE-TARGET)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Wrong Output:

```
((15 10 #f))
```

### 1.6.2 Test (equality, 1/2 partial points)

The rectangle should be created at the center of the target, even when it is dragged

Input:

```
(check-rectangles '((50 50 ,false)) DRAG-2-N)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

## 1.7 Test-Group: Simultaneous drag (1 Points)

Common Definitions

```
(define INITIAL-WORLD (make-world-1 SPEED))

(define BALL-WORLD (send INITIAL-WORLD on-key "r"))

(define SELECT-BOTH (send BALL-WORLD on-mouse CX CY "button-down"))

(define DRAG-BOTH (send SELECT-BOTH on-mouse 150 150 "drag"))

(define RELEASE-BOTH (send DRAG-BOTH on-mouse 150 150 "button-up"))
```

### 1.7.1 Test (equality)

The target should be selected

Input:

```
(check-world CX CY true SELECT-BOTH)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.7.2 Test (equality)

The rectangle should be selected

Input:

```
(check-rectangles '((,CX ,CY ,true)) SELECT-BOTH)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.7.3 Test (equality)

The target is dragged

Input:

```
(check-world 150 150 true DRAG-BOTH)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.7.4 Test (equality)

The rectangle is dragged

Input:

```
(check-rectangles '((,150 ,150 ,true)) DRAG-BOTH)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.7.5 Test (equality)

The target should be un-selected

Input:

```
(check-world 150 150 false RELEASE-BOTH)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

### 1.7.6 Test (equality)

The rectangle should be un-selected

Input:

```
(check-rectangles '((,150 ,150 ,false)) RELEASE-BOTH)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

## 2 Results

Successes: 28

Wrong Outputs: 1

Errors: 0

Achieved Points: 29/2

Total Points (rounded): 14/15