

# CS5010 - Problem Set 10 - Test Results

pdp-pair-zhikai-smithabn

December 3, 2014

This test suite tests your implementation of Problem Set 10

## 1 File: buddies.rkt

Tests your solution to the buddies problem

Common Definitions

```
(define observe-toy
  (lambda (t)
    (list
      (send t toy-x)
      (send t toy-y)
      (send t toy-color)
      (send t toy-selected?))))

(define observe-toys-in-world
  (lambda (w) (map observe-toy (send w get-toys))))

(define CX 400)

(define CY 500)

(define HALF-SQUARE 15)

(define RADIUS 10)

(define check-color
  (lambda (color)
    (lambda (c)
      (or (and (string? c) (string-ci=? c color))
          (and (symbol? c) (string-ci=? (symbol->string c) color))))))
```

## 1.1 Test-Group: Example 1 (5 Points)

### 1.1.1 Test (state, 5 partial points)

Example 1

Common Definitions

```
(define WORLD (b:make-world))
```

### 1.1.2 Test (action/test)

the target is initially black

Action:

```
(void)
```

Test:

```
((check-color "black") (send WORLD target-color))
```

Correct

### 1.1.3 Test (action/test, 1/2 partial points)

after button down the target is orange

Action:

```
(send WORLD on-mouse (/ CX 2) (/ CY 2) "button-down")
```

Test:

```
((check-color "orange") (send WORLD target-color))
```

Correct

### 1.1.4 Test (action/test)

the target should be selected

Action:

```
(void)
```

Test:

```
(equal? (send WORLD target-selected?) #t)
```

Correct

### 1.1.5 Test (action/test, 1/2 partial points)

the square should be created correctly

Action:

```
(begin
  (send WORLD on-key "s")
  (send WORLD on-mouse (+ (/ CX 2) 40) (/ CY 2) "drag")
  (send WORLD on-mouse (+ (/ CX 2) 40) (/ CY 2) "button-up"))
```

Test:

```
(let* ((square (first (send WORLD get-toys)))
  (square-x (send square toy-x))
  (square-y (send square toy-y))
  (square-color (send square toy-color)))
  (and (equal? (list (/ CX 2) (/ CY 2)) (list square-x square-y))
    ((check-color "green") square-color)))
```

Correct

### 1.1.6 Test (action/test, 1/2 partial points)

the square should become selected

Action:

```
(send WORLD on-mouse (/ CX 2) (/ CY 2) "button-down")
```

Test:

```
(let* ((square (first (send WORLD get-toys)))
  (square-color (send square toy-color))
  (square-selected? (send square toy-selected?)))
  (and (equal? square-selected? #t)
    ((check-color "red") square-color)))
```

Correct

### 1.1.7 Test (action/test, 1/2 partial points)

there should be 2 squares

Action:

```
(begin (send WORLD on-key "s"))
```

Test:

```
(set=?
  (observe-toys-in-world WORLD)
  (list
    (list (/ CX 2) (/ CY 2) "red" #t)
    (list (+ (/ CX 2) 40) (/ CY 2) "green" #f)))
```

Correct

### 1.1.8 Test (action/test, 1 partial points)

the 2 square should now be buddies

Action:

```
(send WORLD on-mouse (+ (/ CX 2) 25) (/ CY 2) "drag")
```

Test:

```
(set=?  
(observe-toys-in-world WORLD)  
(list  
  (list (+ (/ CX 2) 25) (/ CY 2) "red" #t)  
  (list (+ (/ CX 2) 40) (/ CY 2) "red" #f)))
```

Wrong State:

```
#f
```

### 1.1.9 Test (action/test, 1 partial points)

the 2 square should now be moving together

Action:

```
(send WORLD on-mouse (+ (/ CX 2) 25 -100) (+ 50 (/ CY 2)) "drag")
```

Test:

```
(set=?  
(observe-toys-in-world WORLD)  
(list  
  (list (+ (/ CX 2) 25 -100) (+ 50 (/ CY 2)) "red" #t)  
  (list (+ (/ CX 2) 40 -100) (+ 50 (/ CY 2)) "red" #f)))
```

Wrong State:

```
#f
```

## 1.2 Test-Group: Example 2 (5 Points)

### 1.2.1 Test (state, 5 partial points)

EXAMPLE 2

Common Definitions

```
(define WORLD (b:make-world))
```

### 1.2.2 Test (action/test, 1/2 partial points)

move the single square

Action:

```
(begin
  (send WORLD on-key "s")
  (send WORLD on-mouse (/ CX 2) (/ CY 2) "button-down")
  (send WORLD on-mouse (/ CX 2) (/ CY 2) "button-up")
  (send WORLD on-mouse (/ CX 2) (- (/ CY 2) 15) "button-down")
  (send WORLD on-mouse (/ CX 2) (- (/ CY 2) 15 10) "drag")
  (send WORLD on-mouse (/ CX 2) (- (/ CY 2) 15 10) "button-up"))
```

Test:

```
(set=?
  (observe-toys-in-world WORLD)
  (list (list (/ CX 2) (- (/ CY 2) 10) "green" #f)))
```

Correct

### 1.2.3 Test (action/test, 1/2 partial points)

create a second square overlapping the first

Action:

```
(send WORLD on-key "s")
```

Test:

```
(set=?
  (observe-toys-in-world WORLD)
  (list
    (list (/ CX 2) (- (/ CY 2) 10) "green" #f)
    (list (/ CX 2) (/ CY 2) "green" #f)))
```

Correct

### 1.2.4 Test (action/test, 1/2 partial points)

no drag event so only one rectangle is selected, no buddies

Action:

```
(send WORLD on-mouse (/ CX 2) (- (/ CY 2) 10 15) "button-down")
```

Test:

```
(set=?
  (observe-toys-in-world WORLD)
  (list
    (list (/ CX 2) (- (/ CY 2) 10) "red" #t)
    (list (/ CX 2) (/ CY 2) "green" #f)))
```

Correct

### 1.2.5 Test (action/test, 1/2 partial points)

drag event should make them buddies

Action:

```
(send WORLD on-mouse (/ CX 2) (- (/ CY 2) 10 15) "drag")
```

Test:

```
(set=?  
(observe-toys-in-world WORLD)  
(list  
  (list (/ CX 2) (- (/ CY 2) 10) "red" #t)  
  (list (/ CX 2) (/ CY 2) "red" #f)))
```

Wrong State:

```
#f
```

### 1.2.6 Test (action/test, 1/2 partial points)

both should be unselected

Action:

```
(send WORLD on-mouse (/ CX 2) (- (/ CY 2) 10 15) "button-up")
```

Test:

```
(set=?  
(observe-toys-in-world WORLD)  
(list  
  (list (/ CX 2) (- (/ CY 2) 10) "green" #f)  
  (list (/ CX 2) (/ CY 2) "green" #f)))
```

Correct

### 1.2.7 Test (action/test, 1/2 partial points)

selecting the other toy should make both red

Action:

```
(send WORLD on-mouse (/ CX 2) (+ (/ CY 2) 15) "button-down")
```

Test:

```
(set=?  
(observe-toys-in-world WORLD)  
(list  
  (list (/ CX 2) (- (/ CY 2) 10) "red" #f)  
  (list (/ CX 2) (/ CY 2) "red" #t)))
```

Wrong State:

```
#f
```

### 1.2.8 Test (action/test, 1/2 partial points)

both should move together on a drag

Action:

```
(send WORLD on-mouse 300 75 "drag")
```

Test:

```
(set=?  
(observe-toys-in-world WORLD)  
(list (list 300 60 "red" #t) (list 300 50 "red" #f)))
```

Wrong State:

```
#f
```

### 1.2.9 Test (action/test, 1/2 partial points)

both should become unselected

Action:

```
(send WORLD on-mouse 300 75 "button-up")
```

Test:

```
(set=?  
(observe-toys-in-world WORLD)  
(list (list 300 60 "green" #f) (list 300 50 "green" #f)))
```

Correct

## 2 File: toys.rkt

Tests your solution to the toys problem

Common Definitions

```
(define observe-world-state  
  (lambda (w)  
    (list  
      (send w target-x)  
      (send w target-y)  
      (send w target-selected?))))  
  
(define observe-world-toys  
  (lambda (w) (map observe-toy (send w get-toys)))))
```

```

(define observe-toy
  (lambda (t)
    (list (send t toy-x) (send t toy-y) (send t toy-color))))

(define check-color
  (lambda (color)
    (lambda (c)
      (or (and (string? c) (string-ci=? c color))
          (and (symbol? c) (string-ci=? (symbol->string c) color))))))

(define get-toy-colors
  (lambda (w)
    (map (lambda (t) (send t toy-color)) (send w get-toys))))

(define to-colorstring
  (lambda (s)
    (cond
      ((string? s) (string-downcase s))
      ((symbol? s) (string-downcase (symbol->string s)))
      (else (error "invalid colorstring")))))

(define color-set=?
  (lambda (colors)
    (lambda (c) (set=? colors (map to-colorstring c)))))

```

## 2.1 Test-Group: test features of circle toys (1 Points)

1/1

Common Definitions

```

(define X 117)

(define Y 257)

```

### 2.1.1 Test (state, 2 partial points)

test circle

Common Definitions

```

(define CIRCLE (t:make-circle-toy X Y))

```



### 2.1.2 Test (action/test, 1/4 partial points)

A circle should start green

Action:

```
(void)
```

Test:

```
((check-color "green") (send CIRCLE toy-color))
```

Correct

### 2.1.3 Test (action/test, 1/2 partial points)

A circle should be red after 5 ticks

Action:

```
(send* CIRCLE (on-tick) (on-tick) (on-tick) (on-tick) (on-tick))
```

Test:

```
((check-color "red") (send CIRCLE toy-color))
```

Correct

### 2.1.4 Test (action/test, 1/4 partial points)

A circle should be green again after 10 total ticks

Action:

```
(send* CIRCLE (on-tick) (on-tick) (on-tick) (on-tick) (on-tick))
```

Test:

```
((check-color "green") (send CIRCLE toy-color))
```

Correct

## 2.2 Test-Group: test features of world (4 Points)

Common Definitions

```
(define SPEED 13)
```

```
(define CENTER-X 200)
```

```
(define CENTER-Y 250)
```

```

(define HALF-SQUARE 20)

(define MAX-X 400)

(define TANGENT-RIGHT (- MAX-X HALF-SQUARE))

(define DX 5)

(define DY 5)

(define N-Y 300)

(define N-X (- MAX-X HALF-SQUARE (* 2 DX)))

(define NEW-X (+ N-X DX))

(define NEW-Y (+ N-Y DY))

```

### 2.2.1 Test (state, 4 partial points)

Test the target and square toys  
Common Definitions

```

(define WORLD (t:make-world SPEED))

```

### 2.2.2 Test (action/test, 1/2 partial points)

Button down inside target selects it  
Action:

```

(send WORLD on-mouse (+ CENTER-X DX) (+ CENTER-Y DY) "button-down")

```

Test:

```

(equal? (send WORLD target-selected?) #t)

```

Correct

### 2.2.3 Test (action/test, 1/2 partial points)

The target should be dragged to the new location

Action:

```
(send WORLD on-mouse NEW-X NEW-Y "drag")
```

Test:

```
(equal?  
 (list (send WORLD target-x) (send WORLD target-y))  
 (list N-X N-Y))
```

Correct

### 2.2.4 Test (action/test, 1 partial points)

A square toy is created at the correct position

Action:

```
(begin  
 (send WORLD on-mouse NEW-X NEW-Y "button-up")  
 (send WORLD on-key "s"))
```

Test:

```
(let* ((square-toy (first (send WORLD get-toys)))  
      (square-x (send square-toy toy-x))  
      (square-y (send square-toy toy-y)))  
 (equal? (list N-X N-Y) (list square-x square-y)))
```

Correct

### 2.2.5 Test (action/test, 1 partial points)

The square toy bounces correctly

Action:

```
(send WORLD on-tick)
```

Test:

```
(let* ((square-toy (first (send WORLD get-toys)))  
      (square-x (send square-toy toy-x)))  
 (equal? square-x TANGENT-RIGHT))
```

Correct

### 2.2.6 Test (state, 1 partial points)

Test the circle toys in the world

Common Definitions

```
(define WORLD (t:make-world SPEED))
```

### 2.2.7 Test (action/test, 1/2 partial points)

2 circles should have different colors at tick 5

Action:

```
(begin
  (send WORLD on-key "c")
  (send WORLD on-tick)
  (send WORLD on-key "c")
  (send* WORLD (on-tick) (on-tick) (on-tick) (on-tick)))
```

Test:

```
(let ((toy-colors (get-toy-colors WORLD)))
  ((color-set=? (list "red" "green")) toy-colors))
```

Correct

### 2.2.8 Test (action/test, 1/2 partial points)

2 circles should both be red on tick 6

Action:

```
(send WORLD on-tick)
```

Test:

```
(let ((toy-colors (get-toy-colors WORLD)))
  ((color-set=? (list "red" "red")) toy-colors))
```

Correct

## 3 Results

Successes: 20

Wrong Outputs: 5

Errors: 0

Achieved Points: 19/2

Total Points (rounded): 10/15