CS5010 - Problem Set 09 - Test Results

pdp-pair-zhikai-smithabn

November 18, 2014

This test suite tests your implementation of Problem Set 09

1 File: toys.rkt

1.1 Test-Group: classes implement interfaces (2 Points)

2/2

1.1.1 Test (equality, 1/3 partial points)

The World% class should implement the World<%> interface
Input:

```
(implementation? World% World<%>)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

1.1.2 Test (equality, 1/3 partial points)

The SquareToy% class should implement the Toy<%> interface
Input:

```
(implementation? SquareToy% Toy<%>)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

1.1.3 Test (equality, 1/3 partial points)

The CircleToy% class should implement the Toy<%> interface
Input:

```
(implementation? CircleToy% Toy<%>)
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

1.2 Test-Group: circle-toy tests (2 Points)

2/2

Common Definitions

```
(define after-n-ticks
(lambda (o n)
(if (= n 0) o (after-n-ticks (send o on-tick) (sub1 n)))))


(define X 117)


(define Y 257)


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


(define CIRCLE-AFTER-TICK (after-n-ticks CIRCLE 1))


(define CIRCLE-AFTER-4-TICKS (after-n-ticks CIRCLE 4))


(define CIRCLE-AFTER-5-TICKS (after-n-ticks CIRCLE 5))


(define CIRCLE-AFTER-10-TICKS (after-n-ticks CIRCLE-AFTER-5-TICKS 5))


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

1.2.1 Test (equality)

The circle should be created at the correct x-coordinate
Input:

```
(send CIRCLE toy-x)
```

Expected Output:

```
X
```

Expected Output Value:

```
117
```

Correct

1.2.2 Test (equality)

The circle should be created at the correct y-coordinate
Input:

```
(send CIRCLE toy-y)
```

Expected Output:

```
Y
```

Expected Output Value:

```
257
```

Correct

1.2.3 Test (equality)

the circle should not move after a tick
Input:

```
(send CIRCLE-AFTER-TICK toy-x)
```

Expected Output:

```
X
```

Expected Output Value:

```
117
```

Correct

1.2.4 Test (equality)

the circle should not move after a tick
Input:

```
(send CIRCLE-AFTER-TICK toy-y)
```

Expected Output:

```
Y
```

Expected Output Value:

```
257
```

Correct

1.2.5 Test (predicate)

circle toys should initially be green
Input:

```
(send CIRCLE toy-color)
```

Output should match:

```
(check-color "green")
```

Correct

1.2.6 Test (predicate, 1/4 partial points)

Color should not change after a single tick
Input:

```
(send CIRCLE-AFTER-TICK toy-color)
```

Output should match:

```
(check-color "green")
```

Correct

1.2.7 Test (predicate, 1/4 partial points)

after 4 ticks the circle should be green
Input:

```
(send CIRCLE-AFTER-4-TICKS toy-color)
```

Output should match:

```
(check-color "green")
```

Correct

1.2.8 Test (predicate, 1/2 partial points)

after 5 ticks the circle should be red
Input:

```
(send CIRCLE-AFTER-5-TICKS toy-color)
```

Output should match:

```
(check-color "red")
```

Correct

1.2.9 Test (predicate, 1/2 partial points)

After 10 ticks the circle should be green again
Input:

```
(send CIRCLE-AFTER-10-TICKS toy-color)
```

Output should match:

```
(check-color "green")
```

Correct

1.3 Test-Group: square-toy-tests (3 Points)

3/3

Common Definitions

```
(define after-n-ticks
(lambda (o n)
(if (= n 0) o (after-n-ticks (send o on-tick) (sub1 n)))))


(define MAX-X 400)


(define MIN-X 0)


(define HALF-SQUARE 20)


(define SPEED 13)


(define DX 10)


(define X 200)
```

```
(define X-NEAR-WALL (- MAX-X HALF-SQUARE DX))

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

(define LEFT-AFTER-TICK (- TANGENT SPEED))

(define Y 250)

(define CENTER-SQUARE (make-square-toy X Y SPEED))

(define CENTER-SQUARE-AFTER-TICK (after-n-ticks CENTER-SQUARE 1))

(define SQUARE-NEAR-WALL (make-square-toy X-NEAR-WALL Y SPEED))

(define SQUARE-NEAR-WALL-AFTER-TICK
(after-n-ticks SQUARE-NEAR-WALL 1))

(define SQUARE-NEAR-LEFT-WALL
(after-n-ticks SQUARE-NEAR-WALL-AFTER-TICK 27))

(define SQUARE-BOUNCE-LEFT (send SQUARE-NEAR-LEFT-WALL on-tick))

(define SQUARE-MOVING-LEFT
(after-n-ticks SQUARE-NEAR-WALL-AFTER-TICK 1))

(define SQUARE-MOVING-LEFT-AFTER-TICK
(send SQUARE-MOVING-LEFT on-tick))

(define valid-color? (lambda (s) (or (string? s) (symbol? s))))
```

1.3.1 Test (equality)

the square should be created at the correct x coordinate
Input:

```
(send CENTER-SQUARE toy-x)
```

Expected Output:

```
X
```

Expected Output Value:

```
200
```

Correct

1.3.2 Test (equality)

the square should be created at the correct y coordinate
Input:

```
(send CENTER-SQUARE toy-y)
```

Expected Output:

```
Y
```

Expected Output Value:

```
250
```

Correct

1.3.3 Test (equality)

y coordinate should not change on tick
Input:

```
(send CENTER-SQUARE-AFTER-TICK toy-y)
```

Expected Output:

```
Y
```

Expected Output Value:

```
250
```

Correct

1.3.4 Test (equality, 1/2 partial points)

The square should move by SPEED in the x direction
Input:

```
(- (send CENTER-SQUARE-AFTER-TICK toy-x) (send CENTER-SQUARE toy-
x))
```

Expected Output:

```
SPEED
```

Expected Output Value:

```
13
```

Correct

1.3.5 Test (equality, 1/2 partial points)

If the square were to move past the wall it should be tangent on the next tick
Input:

```
(send SQUARE-NEAR-WALL-AFTER-TICK toy-x)
```

Expected Output:

```
TANGENT
```

Expected Output Value:

```
380
```

Correct

1.3.6 Test (equality, 1/2 partial points)

after bouncing off the wall the toy should move left by the correct speed
Input:

```
(send SQUARE-MOVING-LEFT toy-x)
```

Expected Output:

```
LEFT-AFTER-TICK
```

Expected Output Value:

```
367
```

Correct

1.3.7 Test (equality, 1/2 partial points)

the square should move at the correct speed even when moving left
Input:

```
(-
(send SQUARE-MOVING-LEFT-AFTER-TICK toy-x)
(send SQUARE-MOVING-LEFT toy-x))
```

Expected Output:

```
(* -1 SPEED)
```

Expected Output Value:

```
-13
```

Correct

1.3.8 Test (equality, 1/2 partial points)

the square should bounce and become tangent to the left wall
Input:

```
(send SQUARE-BOUNCE-LEFT toy-x)
```

Expected Output:

```
HALF-SQUARE
```

Expected Output Value:

```
20
```

Correct

1.3.9 Test (predicate)

The square should have a color
Input:

```
(send CENTER-SQUARE toy-color)
```

Output should match:

```
valid-color?
```

Correct

1.4 Test-Group: world object tests (8 Points)

8/8

Common Definitions

```
(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 after-n-ticks
(lambda (o n)
(if (= n 0) o (after-n-ticks (send o on-tick) (sub1 n)))))

(define send-keys
(lambda (o keys)
(if (empty? keys)
o
(send-keys (send o on-key (first keys)) (rest keys)))))
```

```
(define CENTER-X 200)

(define CENTER-Y 250)

(define HALF-SQUARE 20)

(define MAX-X 400)

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

(define S-X 10)

(define S-Y 10)

(define DX 5)

(define DY 5)

(define SPEED 13)

(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))

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

(define LOTS-OF-KEYS (list "s" "c" "s" "c" "c" "s" "s"))

(define WORLD-WITH-LOTS-OF-TOYS
(send-keys INITIAL-WORLD LOTS-OF-KEYS))

(define WORLD-AFTER-BUTTON-OUTSIDE-TARGET
(send INITIAL-WORLD on-mouse S-X S-Y "button-down"))
```

```
(define WORLD-WITH-BUTTON-DOWN-IN-TARGET
(send INITIAL-WORLD on-mouse
(+ CENTER-X DX)
(+ CENTER-Y DY)
"button-down"))


(define WORLD-AFTER-DRAG
(send WORLD-WITH-BUTTON-DOWN-IN-TARGET on-mouse NEW-X NEW-Y "drag"))


(define WORLD-AFTER-BUTTON-UP
(send WORLD-AFTER-DRAG on-mouse NEW-X NEW-Y "button-up"))


(define WORLD-WITH-SQUARE-TOY (send WORLD-AFTER-DRAG on-key "s"))


(define get-square-toy (lambda (w) (first (send w get-toys))))


(define WORLD-WITH-SQUARE-AFTER-TICK
(send WORLD-WITH-SQUARE-TOY on-tick))


(define WORLD-WITH-SQUARE-NEAR-LEFT-WALL
(after-n-ticks WORLD-WITH-SQUARE-AFTER-TICK 27))


(define WORLD-WITH-SQUARE-BOUNCE-LEFT
(send WORLD-WITH-SQUARE-NEAR-LEFT-WALL on-tick))


(define WORLD-WITH-CIRCLE (send INITIAL-WORLD on-key "c"))


(define WORLD-WITH-CIRCLE-AFTER-TICK (send WORLD-WITH-CIRCLE on-
tick))


(define get-circle-toy (lambda (w) (first (send w get-toys))))


(define WORLD-WITH-2-CIRCLES
(send WORLD-WITH-CIRCLE-AFTER-TICK on-key "c"))


(define WORLD-WITH-DIFFERENT-COLORED-CIRCLES
(after-n-ticks WORLD-WITH-2-CIRCLES 4))
```

```
(define WORLD-WITH-2-SAME-COLORED-CIRCLES
(send WORLD-WITH-DIFFERENT-COLORED-CIRCLES on-tick))

(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)))))
```

1.4.1 Test (equality)

the target should initially be in the center of the canvas
Input:

```
(send INITIAL-WORLD target-x)
```

Expected Output:

```
CENTER-X
```

Expected Output Value:

```
200
```

Correct

1.4.2 Test (equality)

the target should initially be in the center of the canvas
Input:

```
(send INITIAL-WORLD target-y)
```

Expected Output:

```
CENTER-Y
```

Expected Output Value:

```
250
```

Correct

1.4.3 Test (equality)

initially the target is not selected
Input:

```
(send INITIAL-WORLD target-selected?)
```

Expected Output:

```
#f
```

Expected Output Value:

```
#f
```

Correct

1.4.4 Test (equality, 1 partial points)

each s/c key event should add a toy to the world
Input:

```
(length (send WORLD-WITH-LOTS-OF-TOYS get-toys))
```

Expected Output:

```
(length LOTS-OF-KEYS)
```

Expected Output Value:

```
7
```

Correct

1.4.5 Test (equality, 1/2 partial points)

button down outside the target should not select it
Input:

```
(send WORLD-AFTER-BUTTON-OUTSIDE-TARGET target-selected?)
```

Expected Output:

```
#f
```

Expected Output Value:

```
#f
```

Correct

1.4.6 Test (equality, 1/2 partial points)

the target should be selected after a button down inside of it
Input:

```
(send WORLD-WITH-BUTTON-DOWN-IN-TARGET target-selected?)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

1.4.7 Test (equality, 1/2 partial points)

the target should be smoothly dragged to the new position
Input:

```
(send WORLD-AFTER-DRAG target-x)
```

Expected Output:

```
N-X
```

Expected Output Value:

```
370
```

Correct

1.4.8 Test (equality, 1/2 partial points)

the target should be smoothly dragged to the new location
Input:

```
(send WORLD-AFTER-DRAG target-y)
```

Expected Output:

```
N-Y
```

Expected Output Value:

```
300
```

Correct

1.4.9 Test (equality)

the created square should have the correct x coordinate
Input:

```
(send (get-square-toy WORLD-WITH-SQUARE-TOY) toy-x)
```

Expected Output:

```
N-X
```

Expected Output Value:

```
370
```

Correct

1.4.10 Test (equality)

the created square should have the correct y-coordinate
Input:

```
(send (get-square-toy WORLD-WITH-SQUARE-TOY) toy-y)
```

Expected Output:

```
N-Y
```

Expected Output Value:

```
300
```

Correct

1.4.11 Test (equality, 1 partial points)

the square should bounce off the right edge
Input:

```
(send (get-square-toy WORLD-WITH-SQUARE-AFTER-TICK) toy-x)
```

Expected Output:

```
TANGENT-RIGHT
```

Expected Output Value:

```
380
```

Correct

1.4.12 Test (equality, 1/2 partial points)

the square should correctly bounce off the left edge
Input:

```
(send (get-square-toy WORLD-WITH-SQUARE-BOUNCE-LEFT) toy-x)
```

Expected Output:

```
HALF-SQUARE
```

Expected Output Value:

```
20
```

Correct

1.4.13 Test (equality, 1/2 partial points)

the target should become unselected on button up
Input:

```
(send WORLD-AFTER-BUTTON-UP target-selected?)
```

Expected Output:

```
#f
```

Expected Output Value:

```
#f
```

Correct

1.4.14 Test (predicate)

the created circle should initially be green
Input:

```
(send (get-circle-toy WORLD-WITH-CIRCLE) toy-color)
```

Output should match:

```
(check-color "green")
```

Correct

1.4.15 Test (equality)

the circle should be created at the right x coordinate
Input:

```
(send (get-circle-toy WORLD-WITH-CIRCLE) toy-x)
```

Expected Output:

```
CENTER-X
```

Expected Output Value:

```
200
```

Correct

1.4.16 Test (equality)

the circle should be created at the correct y coordinate
Input:

```
(send (get-circle-toy WORLD-WITH-CIRCLE) toy-y)
```

Expected Output:

```
CENTER-Y
```

Expected Output Value:

```
250
```

Correct

1.4.17 Test (predicate, 1 partial points)

there should be 2 different colored circles
Input:

```
(get-toy-colors WORLD-WITH-DIFFERENT-COLORED-CIRCLES)
```

Output should match:

```
(color-set=? (list "red" "green"))
```

Correct

1.4.18 Test (equality, 1 partial points)

both circles should now be red
Input:

```
(get-toy-colors WORLD-WITH-2-SAME-COLORED-CIRCLES)
```

Expected Output:

```
(list "red" "red")
```

Expected Output Value:

```
("red" "red")
```

Correct

2 Results

Successes: 39
Wrong Outputs: 0
Errors: 0
Achieved Points: 15
Total Points (rounded): 15/15