CS5010 - Problem Set 03 - Test Results

pdp-prasadnm

October 6, 2014

This test suite tests your implementation of inventory question of Problem Set 03

1 File: inventory.rkt

This week, we codewalk problem 2
Common Definitions

```
(define inventory-1
(list
 (make-book
  15
  "How to Design Programs"
  "Felleisen et al."
  "MIT Press"
  59
  49
  100
  (make-reorder 2 5)
  1/12)
(make-book
 16
 "A Game of Thrones"
 "George R. R. Martin"
 "Bantam"
 12
 5
 15
 (make-empty-reorder 'any)
 1/20)))


(define inventory-2
(list
 (make-book
  15
```

```
  "How to Design Programs"
  "Felleisen et al."
  "MIT Press"
  49
  39
  2
  (make-reorder 0 50)
  1/12)
 (make-book
  16
  "A Game of Thrones"
  "George R. R. Martin"
  "Bantam"
  12
  5
  15
  (make-empty-reorder 'any)
  1/20)))

(define line-item-1 (make-line-item 15 3))

(define order-1 (list line-item-1))

(define order-2 (list (make-line-item 42 1)))
```

1.1 Test-Group: Required Functions (2 Points)

2/2

Basic tests for the required functions not tested below

1.1.1 Test (equality)

The total profit of an empty inventory should be 0
Input:

```
(inventory-potential-profit empty)
```

Expected Output:

```
0
```

Expected Output Value:

```
0
```

Correct

1.1.2 Test (equality)

The total profit of inventory-1 should be (100 * 10 + 15 * 7)
Input:

```
(inventory-potential-profit inventory-1)
```

Expected Output:

```
(+ (* 100 10) (* 15 7))
```

Expected Output Value:

```
1105
```

Correct

1.1.3 Test (equality)

The total volume of an empty inventory should be 0
Input:

```
(inventory-total-volume empty)
```

Expected Output:

```
0
```

Expected Output Value:

```
0
```

Correct

1.1.4 Test (equality, 0.2 partial points)

The total volume of inventory-1 should be (100 / 12 + 15 / 20)
Input:

```
(inventory-total-volume inventory-1)
```

Expected Output:

```
(+ (* 100 1/12) (* 15 1/20))
```

Expected Output Value:

```
109/12
```

Correct

1.1.5 Test (equality)

For an empty inventory, price-for-line-item should return false
Input:

```
(price-for-line-item empty line-item-1)
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.1.6 Test (equality, 0.2 partial points)

The price for line-item-1 in inventory-1 should be 3*59
Input:

```
(price-for-line-item inventory-1 line-item-1)
```

Expected Output:

```
(* 59 3)
```

Expected Output Value:

```
177
```

Correct

1.1.7 Test (equality)

An empty inventory can not fill a non-empty order
Input:

```
(fillable-now? order-1 empty)
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.1.8 Test (equality, 0.2 partial points)

inventory-1 should be able to fill order-1
Input:

```
(fillable-now? order-1 inventory-1)
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

1.1.9 Test (equality, 0.1 partial points)

An inventory cannot fill an order where it has not enough books on hand
Input:

```
(fillable-now? order-1 inventory-2)
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.1.10 Test (equality, 0.1 partial points)

An inventory cannot fill an order that contains books that are not in the inventory
Input:

```
(fillable-now? order-2 inventory-1)
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.1.11 Test (equality, 0.1 partial points)

The price of an order should be the sum of the prices of the line items
Input:

```
(price-for-order inventory-1 order-1)
```

Expected Output:

```
(* 3 59)
```

Expected Output Value:

```
177
```

Correct

1.1.12 Test (equality, 0.1 partial points)

The price of an order should be the sum of the prices of the line items
Input:

```
(price-for-order inventory-2 order-1)
```

Expected Output:

```
(* 3 49)
```

Expected Output Value:

```
147
```

Correct

1.2 Test-Group: days-til-fillable (2 Points)

2/2

More detailed tests for days-til-fillable

1.2.1 Test (equality)

An empty inventory can never fill a non-empty order
Input:

```
(days-til-fillable order-1 empty)
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.2.2 Test (equality, 0.5 partial points)

inventory-1 should be able to fill order-1 immediately
Input:

```
(days-til-fillable order-1 inventory-1)
```

Expected Output:

```
0
```

Expected Output Value:

```
0
```

Correct

1.2.3 Test (or, 0.5 partial points)

This is a tricky detail, so we also accept a slightly wrong interpretation
Test (equality)
inventory-2 should be able to fill order-1 tomorrow
Input:

```
(days-til-fillable order-1 inventory-2)
```

Expected Output:

```
1
```

Expected Output Value:

```
1
```

Wrong Output:

```
0
```

Test (equality)
It is also okay to say that inventory-2 should be able to fill order-1 today
Input:

```
(days-til-fillable order-1 inventory-2)
```

Expected Output:

```
0
```

Expected Output Value:

```
0
```

Correct

1.3 Test-Group: inventory-after-order (3 Points)

3/3

More detailed tests for inventory-after-order
Common Definitions

```
(define inventory-1-after-order-1
(list
 (make-book
  15
  "How to Design Programs"
  "Felleisen et al."
  "MIT Press"
  59
  49
  97
  (make-reorder 2 5)
  1/12)
(make-book
 16
 "A Game of Thrones"
 "George R. R. Martin"
 "Bantam"
 12
 5
 15
 (make-empty-reorder 'any)
 1/20)))


(define order-3 (list line-item-1 (make-line-item 16 5)))


(define inventory-1-after-order-3
(list
 (make-book
  15
  "How to Design Programs"
  "Felleisen et al."
  "MIT Press"
  59
  49
  97
  (make-reorder 2 5)
  1/12)
(make-book
 16
 "A Game of Thrones"
```

```
 "George R. R. Martin"
 "Bantam"
 12
 5
 10
 (make-empty-reorder 'any)
 1/20)))
```

1.3.1 Test (equality, 0.5 partial points)

An empty order should leave the inventory unchanged
Input:

```
(inventory-after-order inventory-1 empty)
```

Expected Output:

```
inventory-1
```

Expected Output Value:

```
(#(struct:book
15
"How to Design Programs"
"Felleisen et al."
"MIT Press"
59
49
100
#(struct:reorder-status 2 5 #t)
1/12)
#(struct:book
  16
  "A Game of Thrones"
  "George R. R. Martin"
  "Bantam"
  12
  5
  15
  #(struct:reorder-status 1 1 #f)
  1/20))
```

Correct

1.3.2 Test (equality, 0.5 partial points)

After processing order-1, inventory-1 should have three books (of ISBN 15) less in stock
Input:

```
(inventory-after-order inventory-1 order-1)
```

Expected Output:

```
inventory-1-after-order-1
```

Expected Output Value:

```
(#(struct:book
15
"How to Design Programs"
"Felleisen et al."
"MIT Press"
59
49
97
#(struct:reorder-status 2 5 #t)
1/12)
#(struct:book
  16
  "A Game of Thrones"
  "George R. R. Martin"
  "Bantam"
  12
  5
  15
  #(struct:reorder-status 1 1 #f)
  1/20))
```

Correct

1.3.3 Test (equality, 1 partial points)

After processing order-3, inventory-2 should have three books less of ISBN 15 and 5 books less of ISBN 16.
Input:

```
(inventory-after-order inventory-1 order-3)
```

Expected Output:

```
inventory-1-after-order-3
```

Expected Output Value:

```
(#(struct:book
15
"How to Design Programs"
```

```
"Felleisen et al."
"MIT Press"
59
49
97
#(struct:reorder-status 2 5 #t)
1/12)
#(struct:book
  16
  "A Game of Thrones"
  "George R. R. Martin"
  "Bantam"
  12
  5
  10
  #(struct:reorder-status 1 1 #f)
  1/20))
```

Correct

1.4 Test-Group: increase-prices (2 Points)

2/2

More detailed tests for increase-prices
Common Definitions

```
(define inventory-1-after-increase
(list
 (make-book
  15
  "How to Design Programs"
  "Felleisen et al."
  "MIT Press"
  59
  49
  100
  (make-reorder 2 5)
  1/12)
(make-book
 16
 "A Game of Thrones"
 "George R. R. Martin"
 "Bantam"
 15
 5
 15
 (make-empty-reorder 'any)
 1/20)))
```

```
(define inventory-3
(cons
 (make-book
  14
  "A Storm of Swords"
  "George R. R. Martin"
  "Bantam"
  20
  5
  3
  (make-empty-reorder 'test)
  1/20)
inventory-1))


(define inventory-3-after-increase
(cons
 (make-book
  14
  "A Storm of Swords"
  "George R. R. Martin"
  "Bantam"
  25
  5
  3
  (make-empty-reorder 'test)
  1/20)
inventory-1-after-increase))
```

1.4.1 Test (equality)

An empty inventory should not change when prices are increased
Input:

```
(increase-prices empty "MIT Press" 5)
```

Expected Output:

```
empty
```

Expected Output Value:

```
()
```

Correct

1.4.2 Test (equality, 0.5 partial points)

Only books of the given Publisher should have their prices increased
Input:

```
(increase-prices inventory-1 "Bantam" 25)
```

Expected Output:

```
inventory-1-after-increase
```

Expected Output Value:

```
(#(struct:book
15
"How to Design Programs"
"Felleisen et al."
"MIT Press"
59
49
100
#(struct:reorder-status 2 5 #t)
1/12)
#(struct:book
  16
  "A Game of Thrones"
  "George R. R. Martin"
  "Bantam"
  15
  5
  15
  #(struct:reorder-status 1 1 #f)
  1/20))
```

Correct

1.4.3 Test (equality, 0.5 partial points)

All books of the given Publisher should have their prices increased
Input:

```
(increase-prices inventory-3 "Bantam" 25)
```

Expected Output:

```
inventory-3-after-increase
```

Expected Output Value:

```
(#(struct:book
14
"A Storm of Swords"
"George R. R. Martin"
"Bantam"
25
5
3
#(struct:reorder-status 1 1 #f)
1/20)
#(struct:book
  15
  "How to Design Programs"
  "Felleisen et al."
  "MIT Press"
  59
  49
  100
  #(struct:reorder-status 2 5 #t)
  1/12)
#(struct:book
  16
  "A Game of Thrones"
  "George R. R. Martin"
  "Bantam"
  15
  5
  15
  #(struct:reorder-status 1 1 #f)
  1/20))
```

Correct

2 File: balls-in-box.rkt

Tests your implementation of Balls in Box
Common Definitions

```
(define CANVAS-WIDTH 400)


(define CANVAS-HEIGHT 300)


(define CANVAS-HALF-WIDTH (/ CANVAS-WIDTH 2))
```

```
(define CANVAS-HALF-HEIGHT (/ CANVAS-HEIGHT 2))


(define BALL-RADIUS 20)


(define world-after-kev
(lambda (world kev) (world-after-key-event world kev)))


(define world-after-mev
(lambda (world x y mev) (world-after-mouse-event world x y mev)))


(define INITIAL-WORLD (initial-world "TEST"))


(define ONE-BALL-WORLD (world-after-key-event (initial-world 1) "n"))


(define balls-after
(lambda (w)
(map
 (lambda (ball)
(list
 (ball-x-pos ball)
 (ball-y-pos ball)
 (ball-selected? ball)))
(world-balls w))))
```

2.1 Test-Group: Basic functionality (1 Points)

1/1

Covers the basic requirement of the problem

2.1.1 Test (equality, 1/2 partial points)

The initial world should not contain any balls
Input:

```
(world-balls INITIAL-WORLD)
```

Expected Output:

```
empty
```

Expected Output Value:

```
()
```

Correct

2.1.2 Test (equality, 1/2 partial points)

Pressing 'n' should create a ball
Input:

```
(length (world-balls ONE-BALL-WORLD))
```

Expected Output:

```
1
```

Expected Output Value:

```
1
```

Correct

2.1.3 Test (equality, 1/2 partial points)

A new ball should appear halfway between the left and right edges
Input:

```
(ball-x-pos (first (world-balls ONE-BALL-WORLD)))
```

Expected Output:

```
CANVAS-HALF-WIDTH
```

Expected Output Value:

```
200
```

Correct

2.1.4 Test (equality)

Any other key event than 'n' should not change the world
Input:

```
(world-after-key-event ONE-BALL-WORLD "w")
```

Expected Output:

```
ONE-BALL-WORLD
```

Expected Output Value:

```
#(struct:world (#(struct:ball 200 150 0 0 #f)) 1)
```

Correct

2.1.5 Test (equality, 1/2 partial points)

Additional balls should be visible in the world's ball-list
Input:

```
(length (world-balls (world-after-key-event ONE-BALL-WORLD "n")))
```

Expected Output:

```
2
```

Expected Output Value:

```
2
```

Correct

2.2 Test-Group: Mouse Events (1/2 Points)

1/2/1/2

The initial world should not change on a mouse event

2.2.1 Test (equality)

World changed on button-down
Input:

```
(world-after-mouse-event INITIAL-WORLD 150 100 "button-down")
```

Expected Output:

```
INITIAL-WORLD
```

Expected Output Value:

```
#(struct:world () 0)
```

Correct

2.2.2 Test (equality)

World changed on button-up
Input:

```
(world-after-mouse-event INITIAL-WORLD 120 200 "button-up")
```

Expected Output:

```
INITIAL-WORLD
```

Expected Output Value:

```
#(struct:world () 0)
```

Correct

2.2.3 Test (equality)

World changed on drag
Input:

```
(world-after-mouse-event INITIAL-WORLD 0 0 "drag")
```

Expected Output:

```
INITIAL-WORLD
```

Expected Output Value:

```
#(struct:world () 0)
```

Correct

2.2.4 Test (equality)

World changed on enter
Input:

```
(world-after-mouse-event INITIAL-WORLD 150 100 "enter")
```

Expected Output:

```
INITIAL-WORLD
```

Expected Output Value:

```
#(struct:world () 0)
```

Correct

2.2.5 Test (equality)

World changed on leave
Input:

```
(world-after-mouse-event INITIAL-WORLD 17 65 "leave")
```

Expected Output:

```
INITIAL-WORLD
```

Expected Output Value:

```
#(struct:world () 0)
```

Correct

2.3 Test-Group: Key Events (1/2 Points)

1/2/1/2

The initial world should not change on a key event other than "n"

2.3.1 Test (equality)

World changed on backspace
Input:

```
(world-after-key-event INITIAL-WORLD "\b")
```

Expected Output:

```
INITIAL-WORLD
```

Expected Output Value:

```
#(struct:world () 0)
```

Correct

2.3.2 Test (equality)

World changed on q
Input:

```
(world-after-key-event INITIAL-WORLD "q")
```

Expected Output:

```
INITIAL-WORLD
```

Expected Output Value:

```
#(struct:world () 0)
```

Correct

2.3.3 Test (equality)

World changed on space
Input:

```
(world-after-key-event INITIAL-WORLD " ")
```

Expected Output:

```
INITIAL-WORLD
```

Expected Output Value:

```
#(struct:world () 0)
```

Correct

2.3.4 Test (equality)

World changed on %
Input:

```
(world-after-key-event INITIAL-WORLD "%")
```

Expected Output:

```
INITIAL-WORLD
```

Expected Output Value:

```
#(struct:world () 0)
```

Correct

2.4 Test-Group: Key events (1 Points)

1/1

Pressing n should spawn a new ball at the center of the canvas
Common Definitions

```
(define one-world-balls (world-balls ONE-BALL-WORLD))


(define one-ball (first one-world-balls))
```

2.4.1 Test (equality)

There should be only one ball after n was pressed in the initial world
Input:

```
(length one-world-balls)
```

Expected Output:

```
1
```

Expected Output Value:

```
1
```

Correct

2.4.2 Test (equality)

A new ball should spawn in the center of the canvas
Input:

```
(list (ball-x-pos one-ball) (ball-y-pos one-ball))
```

Expected Output:

```
(list CANVAS-HALF-WIDTH CANVAS-HALF-HEIGHT)
```

Expected Output Value:

```
(200 150)
```

Correct

2.4.3 Test (equality, 1/2 partial points)

A new ball should not be selected
Input:

```
(ball-selected? one-ball)
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

2.4.4 Test (equality, 1/2 partial points)

If the world already contains some balls, it should still spawn new balls on KeyEvent n and ignore all other KeyEvents
Input:

```
(length
(world-balls
 (world-after-key-event
  (world-after-key-event
   (world-after-key-event
    (world-after-key-event
     (world-after-key-event
      (world-after-key-event INITIAL-WORLD "n")
      "n")
"%")
"n")
"left")
"n")))
```

Expected Output:

```
4
```

Expected Output Value:

```
4
```

Correct

2.5 Test-Group: Mouse Events (3 Points)

3/3

Tests balls behavior on mouse events
Common Definitions

```
(define CX-200 (+ CANVAS-HALF-WIDTH 5))

(define CY-150 (+ CANVAS-HALF-HEIGHT 5))

(define ONE-BALL-AFTER-BUTTON-DOWN
(world-after-mouse-event
 ONE-BALL-WORLD
 CX-200
 CY-150
 "button-down"))

(define ONE-BALL-AFTER-DRAG
(world-after-mouse-event ONE-BALL-AFTER-BUTTON-DOWN 300 50 "drag"))

(define ONE-BALL-AFTER-BUTTON-UP
(world-after-mouse-event ONE-BALL-AFTER-DRAG 300 50 "button-up"))

(define TWO-BALLS-WORLD
(world-after-key-event ONE-BALL-AFTER-BUTTON-UP "n"))

(define TWO-BALLS-AFTER-BUTTON-DOWN
(world-after-mouse-event
 TWO-BALLS-WORLD
 CX-200
 CY-150
 "button-down"))

(define TWO-BALLS-AFTER-DRAG
(world-after-mouse-event TWO-BALLS-AFTER-BUTTON-DOWN 50 200 "drag"))
```

```
(define TWO-BALLS-AFTER-BUTTON-UP
(world-after-mouse-event TWO-BALLS-AFTER-DRAG 50 200 "button-up"))

(define OVERLAP-TEST-BUTTON-DOWN
(world-after-mouse-event
 TWO-BALLS-AFTER-BUTTON-UP
 50
 200
 "button-down"))

(define OVERLAP-TEST-DRAG
(world-after-mouse-event OVERLAP-TEST-BUTTON-DOWN 300 50 "drag"))

(define multiple-balls
(balls-after
 (world-after-mouse-event
  (world-after-key-event
   (world-after-key-event
    (world-after-key-event
     (world-after-key-event
      (world-after-key-event
       (world-after-key-event INITIAL-WORLD "n")
       "n")
"%")
"n")
"left")
"n")
CANVAS-HALF-WIDTH
CANVAS-HALF-HEIGHT
"button-down")))

(define multiple-balls-selected?
(andmap (lambda (ball) (third ball)) multiple-balls))
```

2.5.1 Test (equality)

The ball should be selected but it's position shouldn't change if mouse is not in center!
Input:

```
(balls-after ONE-BALL-AFTER-BUTTON-DOWN)
```

Expected Output:

```
`(,`(,CANVAS-HALF-WIDTH ,CANVAS-HALF-HEIGHT ,true))
```

Expected Output Value:

```
((200 150 #t))
```

Correct

2.5.2 Test (equality, 1/2 partial points)

Mouse relative distance to ball's center should be maintained while dragging the ball
Input:

```
(balls-after ONE-BALL-AFTER-DRAG)
```

Expected Output:

```
`(,`(,(- 300 5) ,(- 50 5) ,true))
```

Expected Output Value:

```
((295 45 #t))
```

Correct

2.5.3 Test (equality, 1/2 partial points)

The ball should be placed in position and gets unselected
Input:

```
(balls-after ONE-BALL-AFTER-BUTTON-UP)
```

Expected Output:

```
`(,`(,(- 300 5) ,(- 50 5) ,false))
```

Expected Output Value:

```
((295 45 #f))
```

Correct

2.5.4 Test (or)

Test (equality)
The second ball should be selected but it's position shouldn't change if mouse is not in center! First ball should not be affected
Input:

```
(balls-after TWO-BALLS-AFTER-BUTTON-DOWN)
```

Expected Output:

```
`(,`(,CANVAS-HALF-WIDTH ,CANVAS-HALF-HEIGHT ,true)
,`(,(- 300 5) ,(- 50 5) ,false))
```

Expected Output Value:

```
((200 150 #t) (295 45 #f))
```

Correct
Test (equality)
The second ball should be selected but it's position shouldn't change if mouse is not in center! First ball should not be affected
Input:

```
(balls-after TWO-BALLS-AFTER-BUTTON-DOWN)
```

Expected Output:

```
`(,`(,(- 300 5) ,(- 50 5) ,false)
,`(,CANVAS-HALF-WIDTH ,CANVAS-HALF-HEIGHT ,true))
```

Expected Output Value:

```
((295 45 #f) (200 150 #t))
```

Wrong Output:

```
((200 150 #t) (295 45 #f))
```

2.5.5 Test (or, 1/2 partial points)

Test (equality)
The second ball should be selected and dragged along with the mouse! Mouse relative distance to ball's center should be maintained while dragging the ball
Input:

```
(balls-after TWO-BALLS-AFTER-DRAG)
```

Expected Output:

```
`(,`(,(- 50 5) ,(- 200 5) ,true) ,`(,(- 300 5) ,(- 50 5) ,false))
```

Expected Output Value:

```
((45 195 #t) (295 45 #f))
```

Correct
Test (equality)
The second ball should be selected and dragged along with the mouse! Mouse relative distance to ball's center should be maintained while dragging the ball
Input:

```
(balls-after TWO-BALLS-AFTER-DRAG)
```

Expected Output:

```
‘(,‘(,(- 300 5) ,(- 50 5) ,false) ,‘(,(- 50 5) ,(- 200 5) ,true))
```

Expected Output Value:

```
((295 45 #f) (45 195 #t))
```

Wrong Output:

```
((45 195 #t) (295 45 #f))
```

2.5.6 Test (or, 1/2 partial points)

Test (equality)
The second ball should be unselected and dropped in the position!
Input:

```
(balls-after TWO-BALLS-AFTER-BUTTON-UP)
```

Expected Output:

```
‘(,‘(,(- 50 5) ,(- 200 5) ,false) ,‘(,(- 300 5) ,(- 50 5) ,false))
```

Expected Output Value:

```
((45 195 #f) (295 45 #f))
```

Correct
Test (equality)
The second ball should be unselected and dropped in the position!
Input:

```
(balls-after TWO-BALLS-AFTER-BUTTON-UP)
```

Expected Output:

```
‘(,‘(,(- 300 5) ,(- 50 5) ,false) ,‘(,(- 50 5) ,(- 200 5) ,false))
```

Expected Output Value:

```
((295 45 #f) (45 195 #f))
```

Wrong Output:

```
((45 195 #f) (295 45 #f))
```

2.5.7 Test (or)

Test (equality)
The second ball should be selected!
Input:

```
(balls-after
(world-after-mouse-event
 TWO-BALLS-AFTER-BUTTON-UP
 50
 200
 "button-down"))
```

Expected Output:

```
‘(,‘(,(- 50 5) ,(- 200 5) ,true) ,‘(,(- 300 5) ,(- 50 5) ,false))
```

Expected Output Value:

```
((45 195 #t) (295 45 #f))
```

Correct
Test (equality)
The second ball should be selected!
Input:

```
(balls-after
(world-after-mouse-event
 TWO-BALLS-AFTER-BUTTON-UP
 50
 200
 "button-down"))
```

Expected Output:

```
‘(,‘(,(- 300 5) ,(- 50 5) ,false) ,‘(,(- 50 5) ,(- 200 5) ,true))
```

Expected Output Value:

```
((295 45 #f) (45 195 #t))
```

Wrong Output:

```
((45 195 #t) (295 45 #f))
```

2.5.8 Test (or, 1/2 partial points)

Test (equality)
Overlapping the balls should not affect each others state!
Input:

```
(balls-after OVERLAP-TEST-DRAG)
```

Expected Output:

```
‘(,‘(,(- 300 5) ,(- 50 5) ,true) ,‘(,(- 300 5) ,(- 50 5) ,false))
```

Expected Output Value:

```
((295 45 #t) (295 45 #f))
```

Correct
Test (equality)
Overlapping balls should not affect each others state!
Input:

```
(balls-after OVERLAP-TEST-DRAG)
```

Expected Output:

```
‘(,‘(,(- 300 5) ,(- 50 5) ,false) ,‘(,(- 300 5) ,(- 50 5) ,true))
```

Expected Output Value:

```
((295 45 #f) (295 45 #t))
```

Wrong Output:

```
((295 45 #t) (295 45 #f))
```

2.5.9 Test (or)

Test (equality)
Dragging the ball shouldn’t affect a new ball creation
Input:

```
(balls-after (world-after-key-event OVERLAP-TEST-DRAG "n"))
```

Expected Output:

```
‘(,‘(,(- 300 5) ,(- 50 5) ,false)
,‘(,(- 300 5) ,(- 50 5) ,true)
,‘(,CANVAS-HALF-WIDTH ,CANVAS-HALF-HEIGHT ,false))
```

Expected Output Value:

```
((295 45 #f) (295 45 #t) (200 150 #f))
```

Wrong Output:

```
((200 150 #f) (295 45 #t) (295 45 #f))
```

Test (equality)
Dragging the ball shouldn't affect a new ball creation
Input:

```
(balls-after (world-after-key-event OVERLAP-TEST-DRAG "n"))
```

Expected Output:

```
`(,`(,CANVAS-HALF-WIDTH ,CANVAS-HALF-HEIGHT ,false)
,`(,(- 300 5) ,(- 50 5) ,true)
,`(,(- 300 5) ,(- 50 5) ,false))
```

Expected Output Value:

```
((200 150 #f) (295 45 #t) (295 45 #f))
```

Correct

2.5.10 Test (equality, 1/2 partial points)

If there is a button-down event on multiple balls, every ball in the mouse position should be selected
Input:

```
multiple-balls-selected?
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

3 Results

Successes: 49
Wrong Outputs: 0
Errors: 0
Achieved Points: 15
Total Points (rounded): 15/15