CS5010 - Problem Set 04 - Test Results

pdp-prasadnm

October 13, 2014

This test suite tests your implementation of inventory question of Problem Set 04

1 File: inventory.rkt

This week, we codewalk problem 1
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 1 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 (1 Points)

1/1

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
```

Correct
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
```

Wrong Output:

```
1
```

1.3 Test-Group: inventory-after-order (2 Points)

2/2

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 (1 Points)

1/1

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

1.5 Test-Group: inventory-after-deliveries (2 Points)

2/2

More detailed tests for inventory-after-deliveries
Common Definitions

```
(define inventory-1-after-update
(list
 (make-book
  15
  "How to Design Programs"
  "Felleisen et al."
  "MIT Press"
  59
  49
  100
  (make-reorder 1 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-after-update
(list
 (make-book
  15
  "How to Design Programs"
  "Felleisen et al."
  "MIT Press"
  49
  39
  52
  (make-empty-reorder 'any)
  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-4
(list
 (make-book
  53
  "Book1"
  "Author1"
  "Publisher1"
  100
  50
  14
  (make-reorder 1 20)
  1/4)
(make-book
 54
 "Book2"
 "Author2"
 "Publisher2"
 50
 40
 7
 (make-reorder 1 3)
 1/3)
(make-book
 55
 "Book3"
 "Author3"
 "Publisher3"
 20
 5
 59
 (make-reorder 2 50)
 1/2)))

(define inventory-4-after-1-day
(list
 (make-book
  53
  "Book1"
  "Author1"
  "Publisher1"
  100
  50
  34
  (make-empty-reorder 'any)
  1/4)
(make-book
 54
 "Book2"
 "Author2"
 "Publisher2"
 50
 40
 10
 (make-empty-reorder 'any)
 1/3)
(make-book
 55
 "Book3"
 "Author3"
 "Publisher3"
 20
 5
 59
 (make-reorder 1 50)
 1/2)))


(define inventory-4-after-2-days
(list
 (make-book
  53
  "Book1"
  "Author1"
  "Publisher1"
  100
  50
  34
  (make-empty-reorder 'any)
  1/4)
(make-book
 54
 "Book2"
 "Author2"
 "Publisher2"
 50
 40
 10
 (make-empty-reorder 'any)
 1/3)
(make-book
 55
 "Book3"
 "Author3"
 "Publisher3"
 20
 5
 109
 (make-empty-reorder 'any)
 1/2)))
```

1.5.1 Test (equality)

inventory-after-deliveries should not change an empty inventory
Input:

```
(inventory-after-deliveries empty)
```

Expected Output:

```
empty
```

Expected Output Value:

```
()
```

Correct

1.5.2 Test (equality, 0.5 partial points)

inventory-after-deliveries should deduct one from each expected shipment time and leave books without outstanding reorders unchanged
Input:

```
(inventory-after-deliveries inventory-1)
```

Expected Output:

```
inventory-1-after-update
```

Expected Output Value:

```
(#(struct:book
15
"How to Design Programs"
"Felleisen et al."
"MIT Press"
59
49
100
#(struct:reorder-status 1 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

1.5.3 Test (equality, 0.5 partial points)

inventory-after-deliveries should add incoming shipments to the stock and leave books without outstanding reorders unchanged
Input:

```
(inventory-after-deliveries inventory-2)
```

Expected Output:

```
inventory-2-after-update
```

Expected Output Value:

```
(#(struct:book
15
"How to Design Programs"
"Felleisen et al."
"MIT Press"
49
39
52
#(struct:reorder-status 1 1 #f)
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

1.5.4 Test (equality, 0.5 partial points)

inventory-after-deliveries should update shipments and stock according to their interpretation
Input:

```
(inventory-after-deliveries inventory-4)
```

Expected Output:

```
inventory-4-after-1-day
```

Expected Output Value:

```
(#(struct:book
53
"Book1"
"Author1"
"Publisher1"
100
50
34
#(struct:reorder-status 1 1 #f)
1/4)
#(struct:book
  54
  "Book2"
  "Author2"
  "Publisher2"
  50
  40
  10
  #(struct:reorder-status 1 1 #f)
  1/3)
#(struct:book
  55
  "Book3"
  "Author3"
  "Publisher3"
  20
  5
  59
  #(struct:reorder-status 1 50 #t)
  1/2))
```

Correct

1.5.5 Test (equality, 0.5 partial points)

inventory-after-deliveries should update shipments and stock according to their interpretation
Input:

```
(inventory-after-deliveries (inventory-after-deliveries inventory-
4))
```

Expected Output:

```
inventory-4-after-2-days
```

Expected Output Value:

```
(#(struct:book
53
"Book1"
"Author1"
"Publisher1"
100
50
34
#(struct:reorder-status 1 1 #f)
1/4)
#(struct:book
  54
  "Book2"
  "Author2"
  "Publisher2"
  50
  40
  10
  #(struct:reorder-status 1 1 #f)
  1/3)
#(struct:book
  55
  "Book3"
  "Author3"
  "Publisher3"
  20
  5
  109
  #(struct:reorder-status 1 1 #f)
  1/2))
```

Correct

2 File: balls-in-box.rkt

Tests your implementation of Balls in Box
Common Definitions

```
(define BALL-SPEED 7)


(define CANVAS-WIDTH 400)


(define BALL-RADIUS 20)
```

```
(define BALL-SIZE (* BALL-RADIUS 2))

(define LBORDER BALL-RADIUS)

(define RBORDER (- CANVAS-WIDTH BALL-RADIUS))

(define CANVAS-HEIGHT 300)

(define XCENTER (/ CANVAS-WIDTH 2))

(define YCENTER (/ CANVAS-HEIGHT 2))

(define FULL-ROUND (/ (- RBORDER LBORDER) BALL-SPEED))

(define HALF-ROUND (/ FULL-ROUND 2))

(define QUARTER-ROUND (/ HALF-ROUND 2))

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

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

(define INITIAL-WORLD (initial-world BALL-SPEED))

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

(define ball-after
(lambda (ball)
(list (ball-x-pos ball) (ball-y-pos ball) (ball-selected? ball))))

(define balls-after
(lambda (w)
(map (lambda (ball) (ball-after ball)) (world-balls w))))

(define balls-set-after (λ (w) (apply set (balls-after w))))

(define canonicalize-world balls-set-after)
```

```
(define CANONICAL-INITIAL-WORLD (canonicalize-world INITIAL-WORLD))


(define CANONICAL-ONE-BALL-WORLD (canonicalize-world ONE-BALL-WORLD))


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


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


(define get-ball (lambda (balls) (first balls)))


(define simulate-until-right-wall
(lambda (speed w)
(let ((cur-x (ball-x-pos (get-ball (world-balls w)))))
(cond
 ((> cur-x RBORDER) (error "Moved past the right edge"))
 ((or (= cur-x RBORDER) (= cur-x (- RBORDER 1))) w)
 (else
  (let* ((world-after-t (world-after-tick w))
         (next-x
          (ball-x-pos (get-ball (world-balls world-after-t)))))
(if (> next-x cur-x)
(if (or (equal? (abs (- next-x cur-x)) speed)
        (= next-x RBORDER)
        (= next-x (- RBORDER 1)))
(begin (simulate-until-right-wall speed world-after-t))
(error
 "Does not move at full speed to right when it should"))
(error "Does not move towards right wall"))))))))


(define simulate-until-left-wall
(lambda (speed w)
(let ((cur-x (ball-x-pos (get-ball (world-balls w)))))
(cond
 ((< cur-x LBORDER) (error "Moved past the left edge"))
 ((or (= cur-x LBORDER) (= cur-x (+ LBORDER 1))) w)
 (else
  (let* ((world-after-t (world-after-tick w))
         (next-x
          (ball-x-pos (get-ball (world-balls world-after-t)))))
(if (< next-x cur-x)
(if (or (equal? (abs (- next-x cur-x)) speed)
        (= next-x LBORDER)
        (= next-x (+ LBORDER 1)))
(begin (simulate-until-left-wall speed world-after-t))
(error
 "Does not move at full speed to left when it should"))
(error "Does not move towards left wall"))))))))
```

2.1 Test-Group: Basic functionality (1 Points)

1/1

Covers the basic requirement of the problem

2.1.1 Test (equality)

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)

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, 0.5 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:

```
(set=?
(canonical-world-after-kev ONE-BALL-WORLD "w")
CANONICAL-ONE-BALL-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.1.5 Test (equality, 0.5 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

0.5/0.5

2.2 Test-Group: Mouse Events (0.5 Points)

The initial world should not change on a mouse event

2.2.1 Test (equality)

World changed on button-down
Input:

```
(set=?
(canonical-world-after-mev INITIAL-WORLD 150 100 "button-down")
CANONICAL-INITIAL-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.2.2 Test (equality)

World changed on button-up
Input:

```
(set=?
(canonical-world-after-mev INITIAL-WORLD 120 200 "button-up")
CANONICAL-INITIAL-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.2.3 Test (equality)

World changed on drag
Input:

```
(set=?
(canonical-world-after-mev INITIAL-WORLD 0 0 "drag")
CANONICAL-INITIAL-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.2.4 Test (equality)

World changed on enter
Input:

```
(set=?
(canonical-world-after-mev INITIAL-WORLD 150 100 "enter")
CANONICAL-INITIAL-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.2.5 Test (equality)

World changed on leave
Input:

```
(set=?
(canonical-world-after-mev INITIAL-WORLD 17 65 "leave")
CANONICAL-INITIAL-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.3 Test-Group: Key Events (0.5 Points)

0.5/0.5

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

2.3.1 Test (equality)

World changed on backspace
Input:

```
(set=?
(canonical-world-after-kev INITIAL-WORLD "\b")
CANONICAL-INITIAL-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.3.2 Test (equality)

World changed on q
Input:

```
(set=?
(canonical-world-after-kev INITIAL-WORLD "q")
CANONICAL-INITIAL-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.3.3 Test (equality)

World changed on %
Input:

```
(set=?
(canonical-world-after-kev INITIAL-WORLD "%")
CANONICAL-INITIAL-WORLD)
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.4 Test-Group: Key events (1 Points)

1/1

Pressing n should spawn a new ball at the center of the canvas and space pauses the simulation.
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)

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, 0.5 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

2.4.5 Test (equality, 0.5 partial points)

Tick should not affect balls once the world is paused
Input:

```
(set=?
(balls-set-after
 (world-after-tick (world-after-key-event ONE-BALL-WORLD " ")))
(balls-set-after (world-after-key-event ONE-BALL-WORLD " ")))
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

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 BOUNDING-BOX-X (+ XCENTER (sub1 BALL-RADIUS)))


(define BOUNDING-BOX-Y (+ YCENTER (sub1 BALL-RADIUS)))


(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 BOUNDING-BOX-TEST-BUTTON-DOWN
(world-after-mouse-event
 ONE-BALL-WORLD
 BOUNDING-BOX-X
 BOUNDING-BOX-Y
 "button-down"))

(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, 0.1 partial points)

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, 0.3 partial points)

Mouse is in ball's (square-shaped) bounding box but not in ball; ball should be unselected.
Input:

```
(balls-after BOUNDING-BOX-TEST-BUTTON-DOWN)
```

Expected Output:

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

Expected Output Value:

```
((200 150 #f))
```

Correct

2.5.3 Test (equality, 0.3 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

2.5.4 Test (equality, 0.3 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

2.5.5 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:

```
(set=?
(balls-set-after TWO-BALLS-AFTER-BUTTON-DOWN)
(set
 ‘(,CANVAS-HALF-WIDTH ,CANVAS-HALF-HEIGHT ,true)
 ‘(,(- 300 5) ,(- 50 5) ,false)))
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.5.6 Test (equality, 0.5 partial points)

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:

```
(set=?
(balls-set-after TWO-BALLS-AFTER-DRAG)
(set `(,(- 50 5) ,(- 200 5) ,true) `(,(- 300 5) ,(- 50 5) ,false)))
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.5.7 Test (equality, 0.5 partial points)

The second ball should be unselected and dropped in the position!
Input:

```
(set=?
(balls-set-after TWO-BALLS-AFTER-BUTTON-UP)
(set `(,(- 50 5) ,(- 200 5) ,false) `(,(- 300 5) ,(- 50 5) ,false)))
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.5.8 Test (equality)

The second ball should be selected!
Input:

```
(set=?
(balls-set-after
 (world-after-mouse-event
  TWO-BALLS-AFTER-BUTTON-UP
  50
  200
  "button-down"))
(set `(,(- 50 5) ,(- 200 5) ,true) `(,(- 300 5) ,(- 50 5) ,false)))
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.5.9 Test (equality, 0.5 partial points)

Overlapping the balls should not affect each others state!
Input:

```
(set=?
(balls-set-after OVERLAP-TEST-DRAG)
(set `(,(- 300 5) ,(- 50 5) ,true) `(,(- 300 5) ,(- 50 5) ,false)))
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.5.10 Test (equality)

Dragging the ball shouldn't affect a new ball creation
Input:

```
(set=?
(balls-set-after (world-after-key-event OVERLAP-TEST-DRAG "n"))
(set
 `(,(- 300 5) ,(- 50 5) ,false)
 `(,(- 300 5) ,(- 50 5) ,true)
 `(,CANVAS-HALF-WIDTH ,CANVAS-HALF-HEIGHT ,false)))
```

Expected Output:

```
#t
```

Expected Output Value:

```
#t
```

Correct

2.5.11 Test (equality, 0.5 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

1/1

2.6 Test-Group: Tests to check the movements of ball (1 Points)

Common Definitions

```
(define WORLD-WITH-BALL-AT-RIGHT-EDGE
(simulate-until-right-wall BALL-SPEED ONE-BALL-WORLD))


(define WORLD-WITH-BALL-AT-LEFT-EDGE
(simulate-until-left-wall BALL-SPEED WORLD-WITH-BALL-AT-RIGHT-EDGE))
```

2.6.1 Test (equality, 1 partial points)

Ball should move to the right until it bounces off the right edge
Input:

```
(ball-after (get-ball (world-balls WORLD-WITH-BALL-AT-RIGHT-EDGE)))
```

Expected Output:

```
`(,RBORDER ,CANVAS-HALF-HEIGHT ,false)
```

Expected Output Value:

```
(380 150 #f)
```

Correct

2.6.2 Test (equality, 1 partial points)

Ball should move to the left until it bounces off the left edge
Input:

```
(ball-after (get-ball (world-balls WORLD-WITH-BALL-AT-LEFT-EDGE)))
```

Expected Output:

```
`(,LBORDER ,CANVAS-HALF-HEIGHT ,false)
```

Expected Output Value:

```
(20 150 #f)
```

Correct

3 Results

Successes: 57
Wrong Outputs: 0
Errors: 0
Achieved Points: 15.0
Total Points (rounded): 15.0/15.0