CS5010 - Problem Set 05 - Test Results

pdp-group-dhd1991-PrathmeshJ

October 24, 2013

This test suite tests your implementation of Problem Set 05

1 File: trees.rkt

Tests your implementation of Draggable trees
Common Definitions

```
(define SQUARE-SIDE-LENGTH 20)

(define SQ-SIDE-HALF-LENGTH (/ SQUARE-SIDE-LENGTH 2))

(define CHILD-NODE-Y-POS (* SQUARE-SIDE-LENGTH 3))

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

(define WORLD-WITH-ONE-TREE-NODE
(world-after-key-event INITIAL-WORLD "t"))

(define TREE-NODE-POSN
(node-to-center (first (world-to-roots WORLD-WITH-ONE-TREE-NODE))))

(define get-min-x-pos
(lambda (w root-tree)
(foldr
 min
 (foldr
  max
  0
  (map
   (lambda (n) (posn-x (node-to-center n)))
   (node-to-sons (root-tree (world-to-roots w)))))
(map
 (lambda (n) (posn-x (node-to-center n)))
 (node-to-sons (root-tree (world-to-roots w)))))))
```

```
(define CX (posn-x TREE-NODE-POSN))

(define CY (posn-y TREE-NODE-POSN))

(define CX-AFTER-DRAG-200 200)

(define CY-AFTER-DRAG-200 200)

(define SELECTED-ROOT-NODE
(world-after-mouse-event
 WORLD-WITH-ONE-TREE-NODE
 CX
 CY
 "button-down"))

(define DRAGGED-ROOT-NODE
(world-after-mouse-event SELECTED-ROOT-NODE 200 200 "drag"))

(define ONE-ROOT-NODE-WORLD-AFTER-N-KEY
(world-after-key-event DRAGGED-ROOT-NODE "n"))

(define UNSELECTED-ROOT-NODE
(world-after-mouse-event
 ONE-ROOT-NODE-WORLD-AFTER-N-KEY
 200
 200
 "button-up"))

(define SELECTED-ROOT-NODE-2
(world-after-mouse-event
 ONE-ROOT-NODE-WORLD-AFTER-N-KEY
 200
 200
 "button-down"))

(define DRAGGED-PARENT-AND-CHILD
(world-after-mouse-event SELECTED-ROOT-NODE-2 100 100 "drag"))
```

```
(define CHILD-NODE-SELECTED
(world-after-mouse-event
 UNSELECTED-ROOT-NODE
 200
 (+ 200 (* SQUARE-SIDE-LENGTH 3))
 "button-down"))


(define DRAG-CHILD-NODE
(world-after-mouse-event CHILD-NODE-SELECTED 300 250 "drag"))


(define PLACE-CHILD-NODE-TO-RIGHT
(world-after-mouse-event DRAG-CHILD-NODE 300 250 "button-up"))


(define SELECT-ROOT-TO-CREATE-CHILD-2
(world-after-mouse-event
 PLACE-CHILD-NODE-TO-RIGHT
 200
 200
 "button-down"))


(define ROOT-WITH-2-CHILD-NODES
(world-after-key-event SELECT-ROOT-TO-CREATE-CHILD-2 "n"))


(define WORLD-WITH-SUB-TREE
(world-after-key-event CHILD-NODE-SELECTED "n"))


(define SELECT-SUBTREE
(world-after-mouse-event
 WORLD-WITH-SUB-TREE
 200
 (+ 200 (* SQUARE-SIDE-LENGTH 3))
 "button-down"))


(define DRAG-SUBTREE-TO-TOP-RIGHT
(world-after-mouse-event
 SELECT-SUBTREE
 300
 SQUARE-SIDE-LENGTH
 "drag"))
```

```
(define PLACE-SUBTREE-TO-TOP-RIGHT
(world-after-mouse-event
 DRAG-SUBTREE-TO-TOP-RIGHT
 300
 SQUARE-SIDE-LENGTH
 "button-up"))

(define SELECTED-SUBTREE-TO-DELETE
(world-after-mouse-event
 DRAG-SUBTREE-TO-TOP-RIGHT
 300
 SQUARE-SIDE-LENGTH
 "button-down"))

(define TREE-WITH-DELETED-SUBTREE
(world-after-key-event SELECTED-SUBTREE-TO-DELETE "d"))

(define SELETED-TREE-TO-DELETE
(world-after-mouse-event
 TREE-WITH-DELETED-SUBTREE
 200
 200
 "button-down"))

(define DELETED-TREE
(world-after-key-event SELETED-TREE-TO-DELETE "d"))
```

1.1 Test-Group: Basic functionality (3 Points)

3/3

Covers the basic requirement of the problem

1.1.1 Test (equality)

The canvas starts empty!
Input:

```
(world-to-roots INITIAL-WORLD)
```

Expected Output:

```
empty
```

Expected Output Value:

```
()
```

Correct

1.1.2 Test (equality)

Hitting "t" on initial world creates a new root node
Input:

```
(length (world-to-roots WORLD-WITH-ONE-TREE-NODE))
```

Expected Output:

```
1
```

Expected Output Value:

```
1
```

Correct

1.1.3 Test (equality, 1/2 partial points)

Hitting "t", the root node appears tangent to the top of the canvas
Input:

```
(posn-y
(node-to-center (first (world-to-roots WORLD-WITH-ONE-TREE-NODE))))
```

Expected Output:

```
(/ SQUARE-SIDE-LENGTH 2)
```

Expected Output Value:

```
10
```

Correct

1.1.4 Test (equality, 1/2 partial points)

Hitting "t" creates a root node without sons
Input:

```
(length
(node-to-sons (first (world-to-roots WORLD-WITH-ONE-TREE-NODE))))
```

Expected Output:

```
0
```

Expected Output Value:

```
0
```

Correct

1.1.5 Test (equality, 1/2 partial points)

Button-down anywhere outside the tree node should not select the tree node
Input:

```
(node-to-selected?
(first
 (world-to-roots
  (world-after-mouse-event
   WORLD-WITH-ONE-TREE-NODE
   CX
   50
   "button-down"))))
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.1.6 Test (equality, 1/2 partial points)

Button down anywhere inside root node should select the node
Input:

```
(node-to-selected? (first (world-to-roots SELECTED-ROOT-NODE)))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

1.1.7 Test (equality, 1 partial points)

Dragging the root node should move the node in the mouse's position
Input:

```
(node-to-center (first (world-to-roots DRAGGED-ROOT-NODE)))
```

Expected Output:

```
(make-posn CX-AFTER-DRAG-200 CY-AFTER-DRAG-200)
```

Expected Output Value:

```
#(struct:posn 200 200)
```

Correct

11/2/6

1.2 Test-Group: selecting, dragging and deleting node (6 Points)

Covers the tests on Mouse events and key events

1.2.1 Test (equality)

one child should be created for selected tree node
Input:

```
(length
(node-to-sons
 (first (world-to-roots ONE-ROOT-NODE-WORLD-AFTER-N-KEY))))
```

Expected Output:

```
1
```

Expected Output Value:

```
1
```

Correct

1.2.2 Test (equality, 1/2 partial points)

The first son of a node should appear 3 square-lengths down and directly beneath the parent node
Input:

```
(node-to-center
(first
 (node-to-sons
  (first (world-to-roots ONE-ROOT-NODE-WORLD-AFTER-N-KEY)))))
```

Expected Output:

```
(make-posn
CX-AFTER-DRAG-200
(+ CY-AFTER-DRAG-200 (* SQUARE-SIDE-LENGTH 3)))
```

Expected Output Value:

```
#(struct:posn 200 260)
```

Correct

1.2.3 Test (equality)

The second son of a node should appear 2 square-lengths left to leftmost node of root node
Input:

```
(node-to-center
(first
 (node-to-sons
  (first
   (world-to-roots
    (world-after-key-event ONE-ROOT-NODE-WORLD-AFTER-N-KEY "n"))))))
```

Expected Output:

```
(make-posn
(-
 (get-min-x-pos ONE-ROOT-NODE-WORLD-AFTER-N-KEY first)
 (* 2 SQUARE-SIDE-LENGTH))
(+ 200 (* SQUARE-SIDE-LENGTH 3)))
```

Expected Output Value:

```
#(struct:posn 160 260)
```

Correct

1.2.4 Test (equality, 1/2 partial points)

The child node should be selected
Input:

```
(node-to-selected?
(first (node-to-sons (first (world-to-roots CHILD-NODE-SELECTED)))))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

1.2.5 Test (equality)

Selecting child node should not affect the parent node
Input:

```
(node-to-selected? (first (world-to-roots CHILD-NODE-SELECTED)))
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.2.6 Test (equality, 1/2 partial points)

Selecting child node should not affect the parent node
Input:

```
(node-to-center (first (world-to-roots CHILD-NODE-SELECTED)))
```

Expected Output:

```
(node-to-center
(first (world-to-roots ONE-ROOT-NODE-WORLD-AFTER-N-KEY)))
```

Expected Output Value:

```
#(struct:posn 200 200)
```

Correct

1.2.7 Test (equality)

Selecting the parent node should not affect the child node
Input:

```
(node-to-selected?
(first (node-to-sons (first (world-to-roots SELECTED-ROOT-NODE-
2)))))
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.2.8 Test (equality, 1/2 partial points)

Dragging the parent node should update its center position
Input:

```
(node-to-center (first (world-to-roots DRAGGED-PARENT-AND-CHILD)))
```

Expected Output:

```
(make-posn 100 100)
```

Expected Output Value:

```
#(struct:posn 100 100)
```

Correct

1.2.9 Test (equality, 1/2 partial points)

Dragging parent node should also move the subtree
Input:

```
(node-to-center
(first
 (node-to-sons (first (world-to-roots DRAGGED-PARENT-AND-CHILD)))))
```

Expected Output:

```
(make-posn 100 (+ 100 (* 3 SQUARE-SIDE-LENGTH)))
```

Expected Output Value:

```
#(struct:posn 100 160)
```

Correct

1.2.10 Test (equality)

Selecting the child and pressing "n" should not affect its parent
Input:

```
(length (node-to-sons (first (world-to-roots WORLD-WITH-SUB-TREE))))
```

Expected Output:

```
1
```

Expected Output Value:

```
1
```

Correct

1.2.11 Test (equality, 1/2 partial points)

Selecting the child and pressing "n" should create new child
Input:

```
(length
(node-to-sons
 (first
  (node-to-sons (first (world-to-roots WORLD-WITH-SUB-TREE))))))
```

Expected Output:

```
1
```

Expected Output Value:

```
1
```

Correct

1.2.12 Test (equality, 1/2 partial points)

The first son of a node should appear 3 square-lengths down and directly beneath the node
Input:

```
(node-to-center
(first
 (node-to-sons
  (first
   (node-to-sons (first (world-to-roots WORLD-WITH-SUB-TREE)))))))
```

Expected Output:

```
(make-posn
(posn-x
 (node-to-center
  (first
   (node-to-sons (first (world-to-roots WORLD-WITH-SUB-TREE))))))
(+
 (posn-y
  (node-to-center
   (first
    (node-to-sons (first (world-to-roots WORLD-WITH-SUB-TREE))))))
(* SQUARE-SIDE-LENGTH 3)))
```

Expected Output Value:

```
#(struct:posn 200 320)
```

Correct

1.2.13 Test (equality)

Selecting a node should not affect its child and its parent
Input:

```
(or (node-to-selected?
(first
 (node-to-sons
  (first
   (node-to-sons (first (world-to-roots SELECT-SUBTREE)))))))
(node-to-selected? (first (world-to-roots SELECT-SUBTREE))))
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.2.14 Test (equality, 1/2 partial points)

Selecting a node should not affect its child and its parent
Input:

```
(node-to-selected?
(first (node-to-sons (first (world-to-roots SELECT-SUBTREE)))))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

1.2.15 Test (equality, 1/2 partial points)

The child node should have been dragged along with parent node
Input:

```
(node-to-center
(first
 (node-to-sons
  (first
   (node-to-sons
    (first (world-to-roots PLACE-SUBTREE-TO-TOP-RIGHT)))))))
```

Expected Output:

```
(make-posn
(posn-x
 (node-to-center
  (first
   (node-to-sons
    (first (world-to-roots PLACE-SUBTREE-TO-TOP-RIGHT))))))
(+
 (posn-y
  (node-to-center
   (first
    (node-to-sons
     (first (world-to-roots PLACE-SUBTREE-TO-TOP-RIGHT))))))
(* SQUARE-SIDE-LENGTH 3)))
```

Expected Output Value:

```
#(struct:posn 300 80)
```

Correct

1.2.16 Test (equality, 1/2 partial points)

Deleting a selected node should delete its subtree
Input:

```
(node-to-sons (first (world-to-roots TREE-WITH-DELETED-SUBTREE)))
```

Expected Output:

```
empty
```

Expected Output Value:

```
()
```

Correct

1.2.17 Test (equality, 1/2 partial points)

Deleting a selected tree node should delete entire tree
Input:

```
(world-to-roots DELETED-TREE)
```

Expected Output:

```
empty
```

Expected Output Value:

```
()
```

Correct

1.2.18 Test (equality, 1/2 partial points)

Second child node should have been 2 square lengths to the left most child node
Input:

```
(node-to-center
(first
 (node-to-sons (first (world-to-roots ROOT-WITH-2-CHILD-NODES)))))
```

Expected Output:

```
(make-posn
(-
 (posn-x
  (node-to-center
   (first
    (node-to-sons
     (first (world-to-roots SELECT-ROOT-TO-CREATE-CHILD-2))))))
(* 2 SQUARE-SIDE-LENGTH))
(+
 (posn-y
  (node-to-center
   (first (world-to-roots SELECT-ROOT-TO-CREATE-CHILD-2))))
(* 3 SQUARE-SIDE-LENGTH)))
```

Expected Output Value:

```
#(struct:posn 260 260)
```

Wrong Output:

```
#(struct:posn 200 260)
```

1.3 Test-Group: selecting, dragging, creating and deleting multiple nodes (6 Points)

6/6

Covers the tests on Mouse events and key events on multiple nodes
Common Definitions

```
(define WORLD-WITH-TWO-TREES
(world-after-key-event PLACE-SUBTREE-TO-TOP-RIGHT "t"))


(define WORLD-WITH-SECOND-ROOT-NODE-SELECTED
(world-after-mouse-event WORLD-WITH-TWO-TREES CX CY "button-down"))


(define SECOND-ROOT-NODE-WITH-CHILD
(world-after-key-event WORLD-WITH-SECOND-ROOT-NODE-SELECTED "n"))
```

```
(define UNSELECT-SECOND-TREE
(world-after-mouse-event
 SECOND-ROOT-NODE-WITH-CHILD
 CX
 CY
 "button-up"))

(define SELECT-SECOND-TREE-CHILD
(world-after-mouse-event
 UNSELECT-SECOND-TREE
 CX
 (+ CY 60)
 "button-down"))

(define SECOND-TREE-WITH-GRAND-CHILD
(world-after-key-event SELECT-SECOND-TREE-CHILD "n"))

(define UNSELECT-SECOND-TREE-WITH-GRAND-CHILD
(world-after-mouse-event
 SECOND-TREE-WITH-GRAND-CHILD
 CX
 CY
 "button-up"))

(define SELECT-SECOND-TREE
(world-after-mouse-event
 UNSELECT-SECOND-TREE-WITH-GRAND-CHILD
 CX
 CY
 "button-down"))

(define DRAG-SECOND-TREE
(world-after-mouse-event
 SELECT-SECOND-TREE
 (- CX 100)
 (+ CY 100)
 "drag"))

(define PLACE-TREE
(world-after-mouse-event
 DRAG-SECOND-TREE
 (- CX 100)
 (+ CY 100)
 "button-up"))
```

```
(define DRAG-SECOND-TREE-CHILD
(world-after-mouse-event
 SELECT-SECOND-TREE-CHILD
 300
 SQUARE-SIDE-LENGTH
 "drag"))

(define UNSELECT-SECOND-TREE-CHILD
(world-after-mouse-event
 DRAG-SECOND-TREE-CHILD
 300
 SQUARE-SIDE-LENGTH
 "button-up"))

(define SELECT-MULTIPLE-NODES
(world-after-mouse-event
 UNSELECT-SECOND-TREE-CHILD
 300
 SQUARE-SIDE-LENGTH
 "button-down"))

(define DRAG-MULTIPLE-NODES
(world-after-mouse-event
 SELECT-MULTIPLE-NODES
 SQUARE-SIDE-LENGTH
 SQUARE-SIDE-LENGTH
 "drag"))

(define CREATE-NODES-BEYOND-CANVAS
(world-after-key-event DRAG-MULTIPLE-NODES "n"))

(define DELETE-MULTIPLE-NODES
(world-after-key-event CREATE-NODES-BEYOND-CANVAS "d"))

(define CREATE-MULTIPLE-NODES
(lambda (w n)
(foldr
 (lambda (n w) (world-after-key-event w "n"))
 w
 (build-list n (lambda (x) x)))))
```

1.3.1 Test (equality, 1/2 partial points)

Two root nodes should be present in the world
Input:

```
(length (world-to-roots WORLD-WITH-TWO-TREES))
```

Expected Output:

```
2
```

Expected Output Value:

```
2
```

Correct

1.3.2 Test (equality, 1/2 partial points)

Selecting the root node should not affect the other tree, if the new root node doesn't overlaps the nodes in other tree
Input:

```
(and (node-to-selected?
(first (world-to-roots WORLD-WITH-SECOND-ROOT-NODE-SELECTED)))
(not
 (node-to-selected?
  (second
   (world-to-roots WORLD-WITH-SECOND-ROOT-NODE-SELECTED))))
(not
 (ormap
  node-to-selected?
  (node-to-sons
   (second
    (world-to-roots WORLD-WITH-SECOND-ROOT-NODE-SELECTED))))))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

1.3.3 Test (equality, 1/2 partial points)

Dragging a node over the other node should not affect the underlying node
Input:

```
(and (node-to-selected?
(first
 (node-to-sons
  (first (world-to-roots DRAG-SECOND-TREE-CHILD)))))
(not
 (node-to-selected?
  (second (world-to-roots DRAG-SECOND-TREE-CHILD))))
(not
 (ormap
  node-to-selected?
  (node-to-sons
   (second (world-to-roots DRAG-SECOND-TREE-CHILD))))))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

1.3.4 Test (equality, 1/2 partial points)

Overlapping nodes should be selected if anyone of them is selected
Input:

```
(and (node-to-selected?
(first
 (node-to-sons (first (world-to-roots SELECT-MULTIPLE-NODES)))))
(node-to-selected?
 (first
  (node-to-sons
   (second (world-to-roots SELECT-MULTIPLE-NODES))))))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

1.3.5 Test (equality, 1/2 partial points)

Overlapping nodes which were selected should be dragged together
Input:

```
(equal?
(node-to-center
 (first (node-to-sons (first (world-to-roots DRAG-MULTIPLE-NODES)))))
(node-to-center
 (first
  (node-to-sons (second (world-to-roots DRAG-MULTIPLE-NODES))))))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

1.3.6 Test (equality, 1/2 partial points)

Dragging the root node should also move its child nodes and their child nodes
Input:

```
(list
(node-to-center (first (world-to-roots PLACE-TREE)))
(node-to-center
 (first (node-to-sons (first (world-to-roots PLACE-TREE)))))
(node-to-center
 (first
  (node-to-sons
   (first (node-to-sons (first (world-to-roots PLACE-TREE))))))))
```

Expected Output:

```
(list
(make-posn (- CX 100) (+ CY 100))
(make-posn (- CX 100) (+ CY (* 3 SQUARE-SIDE-LENGTH) 100))
(make-posn (- CX 100) (+ CY (* 6 SQUARE-SIDE-LENGTH) 100)))
```

Expected Output Value:

```
(#(struct:posn 100 110) #(struct:posn 100 170) #(struct:posn 100 230))
```

Correct

1.3.7 Test (equality, 1/2 partial points)

Two child nodes should have been created for each selected Node on different trees
Input:

```
(list
(length
 (node-to-sons
  (first
   (node-to-sons
    (first (world-to-roots CREATE-NODES-BEYOND-CANVAS))))))
(length
 (node-to-sons
  (first
   (node-to-sons
    (second (world-to-roots CREATE-NODES-BEYOND-CANVAS)))))))
```

Expected Output:

```
(list 1 1)
```

Expected Output Value:

```
(1 1)
```

Correct

1.3.8 Test (equality, 1/2 partial points)

Selected nodes should be deleted completely
Input:

```
(list
(length
 (node-to-sons (first (world-to-roots DELETE-MULTIPLE-NODES))))
(length
 (node-to-sons
  (second (world-to-roots CREATE-NODES-BEYOND-CANVAS)))))
```

Expected Output:

```
(list 0 1)
```

Expected Output Value:

```
(0 1)
```

Correct

1.3.9 Test (or, 1/2 partial points)

Child node should not be created beyond left boundary. Accepting if center is in the canvas

Test (equality)

Child node should not be created beyond the left boundary.

Input:

```
(length
(node-to-sons
 (first
  (world-to-roots (CREATE-MULTIPLE-NODES SELECTED-ROOT-NODE-2 9)))))
```

Expected Output:

```
5
```

Expected Output Value:

```
5
```

Correct

Test (equality)

Child node center should not be created beyond the left boundary.

Input:

```
(length
(node-to-sons
 (first
  (world-to-roots (CREATE-MULTIPLE-NODES SELECTED-ROOT-NODE-2 9)))))
```

Expected Output:

```
6
```

Expected Output Value:

```
6
```

Wrong Output:

```
5
```

2 Results

Successes: 33
Wrong Outputs: 1
Errors: 0
Achieved Points: 29/2
Total Points (rounded): 14/15