CS5010 - Problem Set 08 - Test Results

pdp-group-dantuluru-rosefox911

March 25, 2014

This test suite tests your implementation of Problem Set 08

1 File: robot.rkt

Tests your implementation of robot

1.1 Test-Group: Simple cases (3 Points)

3/3

1.1.1 Test (equality, 1/2 partial points)

The robot is already at the target
Input:

```
(path '(1 1) '(1 1) empty)
```

Expected Output:

```
empty
```

Expected Output Value:

```
()
```

Correct

1.1.2 Test (predicate, 1/2 partial points)

Simple move towards south west
Input:

```
(path '(1 1) '(10 10) empty)
```

Output should match:

```
(check-moves '(1 1) '(10 10))
```

Correct

1.1.3 Test (predicate, 1/2 partial points)

Simple move towards north east
Input:

```
(path '(10 10) '(1 1) empty)
```

Output should match:

```
(check-moves '(10 10) '(1 1))
```

Correct

1.1.4 Test (equality, 1/2 partial points)

Obstacle on target
Input:

```
(path '(1 1) '(10 10) '((10 10)))
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.1.5 Test (equality, 1/2 partial points)

Cannot move from start
Input:

```
(path '(1 1) '(10 10) '((2 1) (1 2)))
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

1.2 Test-Group: Simple movement where path exists given normal obstacles (1 Points)

1/1

Common Definitions

```
(define VERTICAL-WALL '((2 1) (2 2) (2 3)))


(define HORIZONTAL-WALL '((1 3) (2 3) (3 3) (4 3) (5 3) (6 3) (7 3)))
```

1.2.1 Test (predicate)

Move east with straight wall in between
Input:

```
(path '(1 1) '(4 1) VERTICAL-WALL)
```

Output should match:

```
(check-moves '(1 1) '(4 1))
```

Correct

1.2.2 Test (predicate)

Move south west with horizontal wall below
Input:

```
(path '(1 1) '(5 5) HORIZONTAL-WALL)
```

Output should match:

```
(check-moves '(1 1) '(5 5))
```

Correct

1.3 Test-Group: Movement through a maze (2 Points)

2/2

Common Definitions

```
(define MAZE
'((3 1)
  (3 2)
  (3 3)
  (3 4)
  (5 4)
  (5 3)
  (5 2)
  (5 5)
```

```
  (5 6)
  (5 7)
  (6 7)
  (7 7)
  (7 6)
  (7 5)
  (7 4)
  (7 3)
  (7 2)
  (7 1)))

(define CROSS-WALL
'((4 2)
  (4 3)
  (4 4)
  (4 5)
  (4 6)
  (4 7)
  (2 4)
  (3 4)
  (5 4)
  (6 4)
  (7 4)
  (8 4)))
```

1.3.1 Test (predicate, 1 partial points)

Move through a maze
Input:

```
(path '(1 1) '(6 4) MAZE)
```

Output should match:

```
(check-moves '(1 1) '(6 4))
```

Correct

1.3.2 Test (predicate, 1 partial points)

Move around cross to opposite side
Input:

```
(path '(3 5) '(5 3) CROSS-WALL)
```

Output should match:

```
(check-moves '(3 5) '(5 3))
```

Correct

1.4 Test-Group: Unreachable due to blocked off obstacles (1 Points)

1/1

Common Definitions

```
(define BOX
'((4 4)
  (5 4)
  (6 4)
  (7 4)
  (8 4)
  (8 5)
  (8 6)
  (8 7)
  (8 8)
  (8 9)
  (7 9)
  (6 9)
  (5 9)
  (4 9)
  (4 8)
  (4 7)
  (4 6)
  (4 5)))
```

1.4.1 Test (equality, 1 partial points)

No path to a target which is blocked off
Input:

```
(path '(1 1) '(6 6) BOX)
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

2 File: obstacles.rkt

Tests your implementation of board-to-obstacles

8/8

2.1 Test-Group: (8 Points)

2.1.1 Test (equality, 1/2 partial points)

Normal obstacle
Input:

```
(obstacle? '((1 2) (1 3) (2 3) (2 4)))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

2.1.2 Test (equality, 1/2 partial points)

Larger obstacle
Input:

```
(obstacle? '((1 1) (1 2) (1 3) (2 3) (3 3) (3 2) (3 1) (2 1)))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

2.1.3 Test (equality, 1/2 partial points)

Single block is an obstacle.
Input:

```
(obstacle? '((1 1)))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

2.1.4 Test (equality, 1/2 partial points)

Two single block obstacles sharing a corner should not be an obstacle
Input:

```
(obstacle? '((2 3) (3 2)))
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

2.1.5 Test (equality, 1 partial points)

Two obstacles sharing a corner should not be an obstacle
Input:

```
(obstacle? '((1 3) (2 3) (3 2) (4 2)))
```

Expected Output:

```
false
```

Expected Output Value:

```
#f
```

Correct

2.1.6 Test (equality, 1 partial points)

Single obstacle
Input:

```
(set-equal? (board-to-obstacles '((1 1))) '(((1 1))))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

2.1.7 Test (equality, 1 partial points)

One obstacle
Input:

```
(set-equal? (board-to-obstacles '((2 1) (1 2))) '(((1 2)) ((2 1))))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

2.1.8 Test (equality, 1 partial points)

Contains multiple obstacles.
Input:

```
(set-equal?
(board-to-obstacles '((1 2) (1 3) (2 3) (3 2) (3 4) (4 1) (4 4)))
'(((1 2) (1 3) (2 3)) ((3 2)) ((4 1)) ((3 4) (4 4))))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

2.1.9 Test (equality, 1 partial points)

One big obstacle
Input:

```
(set-equal?
(board-to-obstacles
 '((1 1) (1 2) (1 3) (2 3) (3 3) (3 2) (3 1) (2 1)))
'(((1 1) (1 2) (1 3) (2 3) (3 3) (3 2) (3 1) (2 1))))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

2.1.10 Test (equality, 1/2 partial points)

Big obstacle
Input:

```
(set-equal?
(board-to-obstacles
 '((1 1) (2 2) (1 3) (2 4) (1 5) (2 6) (1 7) (3 3) (4 4) (3 5)))
'(((1 1))
  ((2 2))
  ((1 3))
  ((2 4))
  ((1 5))
  ((2 6))
  ((1 7))
  ((3 3))
  ((4 4))
  ((3 5))))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

2.1.11 Test (equality, 1/2 partial points)

Obstacle at a distance
Input:

```
(set-equal?
(board-to-obstacles '((1 1000) (2 1000) (1000 1000)))
'(((1 1000) (2 1000)) ((1000 1000))))
```

Expected Output:

```
true
```

Expected Output Value:

```
#t
```

Correct

3 Results

Successes: 21
Wrong Outputs: 0
Errors: 0
Achieved Points: 15
Total Points (rounded): 15/15