```
'''
Created on Mar 1, 2015

@author: mroch
'''
import unittest

import boardlibrary

# Unit tests for verifying functionality of checkerboard class
# Students do not need this to complete the assignment, but if you
# want to learn about unit testing, this may be helpful although
# you'll need to couple it with a unit test tutorial as this is not
# designed to be a ground-up tutorial.

class testBoard(unittest.TestCase):
    def setUp(self):
        pass

    def tupleize_list(self, l):
        # tupleize_list - list
        # Convert list of lists to tuple so that we can
        # use it in set operations (tuples are hashable,
        # lists are not).
        return tuple([tuple(item) for item in l])

    def test_prisitine(self):
        "Check moves on initial checkerboard"

        # Initial board
        b = boardlibrary.boards["Pristine"]

        # Red moves?
        actions = b.get_actions('r')

        if False:
            # Show board moves
            # Not a real unit test
            for a in actions:
                newb = b.move(a)

        # Convert to tuple for set operations
        actions = self.tupleize_list(actions)
        redexpected = set(
                (((5, 0), (4, 1)),
                 ((5, 2), (4, 1)),
                 ((5, 2), (4, 3)),
                 ((5, 4), (4, 3)),
                 ((5, 4), (4, 5)),
                 ((5, 6), (4, 5)),
                 ((5, 6), (4, 7))))
        self.assertEqual(set(actions), redexpected, "Bad red move")

        # Black moves?
        actions = b.get_actions('b')
        # Convert to tuple for set operations
        actions = self.tupleize_list(actions)
        blackexpected = set(
                (((2, 1), (3, 0)),
```

```
60                 ((2, 1), (3, 2)),
61                 ((2, 3), (3, 2)),
62                 ((2, 3), (3, 4)),
63                 ((2, 5), (3, 4)),
64                 ((2, 5), (3, 6)),
65                 ((2, 7), (3, 6))))
66         self.assertEqual(set(actions), blackexpected, "Bad black move")
67
68     def test_simplecapture(self):
69         "Single capture - no multiple hops"
70
71         # See boardlibrary for details
72         b = boardlibrary.boards["SingleHopsRed"]
73         actions = b.get_actions('r')
74         actions = set(self.tupleize_list(actions))
75         redexpected = set(
76             self.tupleize_list([
77                     [(4, 7), (2, 5, (3, 6))],
78                     [(5, 2), (3, 4, (4, 3))],
79                     [(5, 4), (3, 2, (4, 3))]
80                     ]))
81         self.assertEqual(actions, redexpected)
82
83         # Set up black captures
84         # See boardlbirary for details
85         b = boardlibrary.boards["SingleHopsBlack"]
86         actions = b.get_actions('b')
87         actions = set(self.tupleize_list(actions))
88         blackexpected = set(self.tupleize_list([
89                 [(2, 7), (4, 5, (3, 6))],
90                 [(4, 3), (6, 1, (5, 2))]
91                 ]))
92         self.assertEqual(actions, blackexpected)
93
94     def test_multihopcapture(self):
95         "Can we predict multiple hops"
96
97         # See boardlibrary for details
98         b = boardlibrary.boards['multihop']
99
100        actions = b.get_actions('r')
101        actions = set(self.tupleize_list(actions))
102        redexpected = set(self.tupleize_list([
103                    [(5, 6), (3, 4, (4, 5)), (1, 6, (2, 5))]
104                    ]))
105        self.assertEqual(actions, redexpected)
106
107        actions = b.get_actions('b')
108
109        if True:
110            # Show board moves
111            # Not a real unit test
112            for a in actions:
113                newb = b.move(a, verbose=True)
114
115        actions = set(self.tupleize_list(actions))
116        blackexpected = set(self.tupleize_list([
117                [(0, 1), (2, 3, (1, 2))],
118                [(1, 0), (3, 2, (2, 1)), (5, 4, (4, 3)), (7, 2, (6, 3))],
```

```
119                 [(1, 0), (3, 2, (2, 1)), (5, 4, (4, 3)), (7, 6, (6, 5))]
120                 ]))
121         self.assertEqual(actions, blackexpected)
122 
123     def test_kingstour(self):
124         """test_kingstour - Verify kings tour
125         Verify that we can accurately find a king's tour and that a pawn
126         that is kinged cannot continue on to the King's tour
127         """
128 
129         # Black can be crowned after double jump move
130         # See boardlibrary for details
131         b = boardlibrary.boards['KingBlack']
132         # Need to make sure that we can go backwards after being kinged
133         # and that we don't retake any pieces that were already taken
134         actions = b.get_actions('b')
135         actions = set(self.tupleize_list(actions))
136         blackexpected = set(self.tupleize_list([
137                 [(3, 4), (5, 2, (4, 3)), (7, 4, (6, 3))],
138                 [(3, 4), (5, 6, (4, 5)), (7, 4, (6, 5))]
139             ]))
140         self.assertEqual(actions, blackexpected)
141 
142         # Black king can tour
143         # See boardlibrary for details
144         b = boardlibrary.boards['BlackKingTour']
145         actions = b.get_actions('b')
146         actions = set(self.tupleize_list(actions))
147         blackexpected = set(self.tupleize_list([
148                 [(3, 4), (5, 6, (4, 5)), (7, 4, (6, 5)),
149                  (5, 2, (6, 3)), (3, 4, (4, 3))],
150                 [(3, 4), (5, 2, (4, 3)), (7, 4, (6, 3)),
151                  (5, 6, (6, 5)), (3, 4, (4, 5))]
152             ]))
153         self.assertEqual(actions, blackexpected)
154 
155         # Red king can tour
156         # See boardlibrary for details
157         b = boardlibrary.boards['RedKingTour']
158         actions = b.get_actions('r')
159         actions = set(self.tupleize_list(actions))
160         redexpected = set(self.tupleize_list([
161                 [(3, 4), (5, 6, (4, 5)), (7, 4, (6, 5)),
162                  (5, 2, (6, 3)), (3, 4, (4, 3))],
163                 [(3, 4), (5, 2, (4, 3)), (7, 4, (6, 3)),
164                  (5, 6, (6, 5)), (3, 4, (4, 5))]
165             ]))
166         self.assertEqual(actions, redexpected)
167 
168 # Run test cases if invoked as main module
169 if __name__ == "__main__":
170     b = boardlibrary.boards["Pristine"]
171     for player in ['r', 'b']:
172         for r in range(8):
173             print("player %s row %d distance %d\n"%(player, r, b.disttoking(player, r)))
174 
175     # Execute the test suite shwoing results for each test
176     suite = unittest.TestLoader().loadTestsFromTestCase(testBoard)
177     unittest.TextTestRunner(verbosity=2).run(suite)
```

178