

```

1 package Project3;
2
3 import org.junit.Test;
4
5 import static org.junit.Assert.*;
6
7 /*****
8  * CIS 163 Section 01
9  * Project 3: Chess Game
10 * ChessTesting Class
11 *
12 * This class contains tests for all methods in ChessModel.
13 *
14 * @author George Fayette
15 * @version 1/14/2019
16 *****/
17 public class ChessTesting {
18
19     @Test
20     public void testBlackPawnMove() {
21         ChessModel game = new ChessModel();
22         game.clearBoard();
23         game.setPiece(1, 0, new Pawn(Player.BLACK));
24         game.tryMove(new Move(1, 0, 3, 0));
25         assertTrue(game.pieceAt(3, 0).type().equals("Pawn"));
26         assertTrue(game.pieceAt(3, 0).player() == Player.BLACK);
27     }
28
29     @Test
30     public void testWhitePawnMove() {
31         ChessModel game = new ChessModel();
32         game.clearBoard();
33         game.setPiece(6, 0, new Pawn(Player.WHITE));
34         game.tryMove(new Move(6, 0, 4, 0));
35         assertTrue(game.pieceAt(4, 0).type().equals("Pawn"));
36         assertTrue(game.pieceAt(4, 0).player() == Player.WHITE);
37     }
38
39     @Test
40     public void testBlackKingMove() {
41         ChessModel game = new ChessModel();
42         game.clearBoard();
43         game.setPiece(0, 4, new King(Player.BLACK));
44         game.tryMove(new Move(0, 4, 0, 3));
45         assertTrue(game.pieceAt(0, 3).type().equals("King"));
46         assertTrue(game.pieceAt(0, 3).player() == Player.BLACK);
47     }
48
49     @Test
50     public void testWhiteKingMove() {
51         ChessModel game = new ChessModel();
52         game.clearBoard();
53         game.setPiece(7, 4, new King(Player.WHITE));
54         game.tryMove(new Move(7, 4, 7, 5));
55         assertTrue(game.pieceAt(7, 5).type().equals("King"));
56         assertTrue(game.pieceAt(7, 5).player() == Player.WHITE);
57     }
58
59     @Test
60     public void testBlackQueenMove() {

```

```

61     ChessModel game = new ChessModel();
62     game.clearBoard();
63     game.setPiece(0, 3, new Queen(Player.BLACK));
64     game.tryMove(new Move(0, 3, 3, 0));
65     assertTrue(game.pieceAt(3, 0).type().equals("Queen"));
66     assertTrue(game.pieceAt(3, 0).player() == Player.BLACK);
67 }
68
69 @Test
70 public void testWhiteQueenMove() {
71     ChessModel game = new ChessModel();
72     game.clearBoard();
73     game.setPiece(0, 3, new Queen(Player.WHITE));
74     game.tryMove(new Move(0, 3, 3, 0));
75     assertTrue(game.pieceAt(3, 0).type().equals("Queen"));
76     assertTrue(game.pieceAt(3, 0).player() == Player.WHITE);
77 }
78
79 @Test
80 public void testBlackBishopMove() {
81     ChessModel game = new ChessModel();
82     game.clearBoard();
83     game.setPiece(0, 2, new Bishop(Player.BLACK));
84     game.tryMove(new Move(0, 2, 2, 0));
85     assertTrue(game.pieceAt(2, 0).type().equals("Bishop"));
86     assertTrue(game.pieceAt(2, 0).player() == Player.BLACK);
87 }
88
89 @Test
90 public void testWhiteBishopMove() {
91     ChessModel game = new ChessModel();
92     game.clearBoard();
93     game.setPiece(0, 2, new Bishop(Player.WHITE));
94     game.tryMove(new Move(0, 2, 2, 0));
95     assertTrue(game.pieceAt(2, 0).type().equals("Bishop"));
96     assertTrue(game.pieceAt(2, 0).player() == Player.WHITE);
97 }
98
99
100 @Test
101 public void testBlackKnightMove() {
102     ChessModel game = new ChessModel();
103     game.clearBoard();
104     game.setPiece(0, 1, new Knight(Player.BLACK));
105     game.tryMove(new Move(0, 1, 2, 0));
106     assertTrue(game.pieceAt(2, 0).type().equals("Knight"));
107     assertTrue(game.pieceAt(2, 0).player() == Player.BLACK);
108 }
109
110 @Test
111 public void testWhiteKnightMove() {
112     ChessModel game = new ChessModel();
113     game.clearBoard();
114     game.setPiece(0, 1, new Knight(Player.WHITE));
115     game.tryMove(new Move(0, 1, 2, 0));
116     assertTrue(game.pieceAt(2, 0).type().equals("Knight"));
117     assertTrue(game.pieceAt(2, 0).player() == Player.WHITE);
118 }
119
120 @Test

```

```

121     public void testBlackRookMove() {
122         ChessModel game = new ChessModel();
123         game.clearBoard();
124         game.setPiece(0, 0, new Rook(Player.BLACK));
125         game.tryMove(new Move(0, 0, 0, 7));
126         assertTrue(game.pieceAt(0, 7).type().equals("Rook"));
127         assertTrue(game.pieceAt(0, 7).player() == Player.BLACK);
128     }
129
130     @Test
131     public void testWhiteRookMove() {
132         ChessModel game = new ChessModel();
133         game.clearBoard();
134         game.setPiece(0, 0, new Rook(Player.WHITE));
135         game.tryMove(new Move(0, 0, 0, 7));
136         assertTrue(game.pieceAt(0, 7).type().equals("Rook"));
137         assertTrue(game.pieceAt(0, 7).player() == Player.WHITE);
138     }
139
140
141     @Test
142     public void testCastling() {
143         ChessModel game = new ChessModel();
144         game.clearBoard();
145         game.setPiece(0, 0, new Rook(Player.BLACK));
146         game.setPiece(0, 4, new King(Player.BLACK));
147         game.tryMove(new Move(0, 4, 0, 2));
148         assertTrue(game.pieceAt(0, 3).type().equals("Rook"));
149         assertTrue(game.pieceAt(0, 3).player() == Player.BLACK);
150         assertTrue(game.pieceAt(0, 2).type().equals("King"));
151         assertTrue(game.pieceAt(0, 2).player() == Player.BLACK);
152     }
153
154     @Test
155     public void testEnPassant() {
156         ChessModel game = new ChessModel();
157         game.clearBoard();
158         game.setPiece(1, 0, new Pawn(Player.BLACK));
159         game.setPiece(4, 1, new Pawn(Player.WHITE));
160         game.tryMove(new Move(4, 1, 3, 1));
161         game.tryMove(new Move(1, 0, 3, 0));
162         game.tryMove(new Move(3, 1, 2, 0));
163         assertTrue(game.pieceAt(2, 0).type().equals("Pawn"));
164         assertTrue(game.pieceAt(2, 0).player() == Player.WHITE);
165         assertTrue(game.pieceAt(3, 0) == null);
166     }
167
168
169     @Test
170     public void testUndo() {
171         ChessModel game = new ChessModel();
172         game.clearBoard();
173         game.setPiece(1, 0, new Pawn(Player.BLACK));
174         game.tryMove(new Move(1, 0, 3, 0));
175         game.undo();
176         assertTrue(game.pieceAt(1, 0).type().equals("Pawn"));
177         assertTrue(game.pieceAt(1, 0).player() == Player.BLACK);
178     }
179 }
180

```

```

181
182     @Test
183     public void testUpgrade() {
184         ChessModel game = new ChessModel();
185         game.clearBoard();
186         game.setPiece(5, 0, new Pawn(Player.WHITE));
187         game.tryMove(new Move(5, 0, 3, 0));
188         game.upgradePawn("Queen");
189         assertTrue(game.pieceAt(3, 0).type().equals("Queen"));
190         assertTrue(game.pieceAt(3, 0).player() == Player.WHITE);
191
192     }
193
194     @Test
195     public void testUpdateStatus() {
196         ChessModel game = new ChessModel();
197         game.clearBoard();
198         game.setPiece(5, 0, new Pawn(Player.WHITE));
199         game.setPiece(1, 0, new Pawn(Player.BLACK));
200         game.tryMove(new Move(5, 0, 3, 0));
201         game.updateStatus();
202         assertTrue(game.GUICode() == GUICodes.NO_MESSAGE);
203
204     }
205
206     @Test
207     public void testAI() {
208         ChessModel game = new ChessModel();
209         game.clearBoard();
210         game.setPiece(7, 0, new Rook(Player.WHITE));
211         game.setPiece(1, 0, new Pawn(Player.BLACK));
212         game.tryMove(new Move(7, 0, 6, 0));
213         game.AI();
214         assertTrue(game.pieceAt(2, 0).type().equals("Pawn"));
215         assertTrue(game.pieceAt(2, 0).player() == Player.BLACK);
216     }
217
218     @Test
219     public void testCheckMate() {
220         ChessModel game = new ChessModel();
221         game.clearBoard();
222         game.setPiece(7, 0, new Rook(Player.WHITE));
223         game.setPiece(6, 1, new Rook(Player.WHITE));
224         game.setPiece(0, 0, new King(Player.BLACK));
225         game.tryMove(new Move(7, 0, 6, 0));
226         assertTrue(game.isCheckmate());
227     }
228
229     @Test
230     public void testDraw() {
231         ChessModel game = new ChessModel();
232         game.clearBoard();
233         game.setPiece(1, 7, new Rook(Player.WHITE));
234         game.setPiece(7, 1, new Rook(Player.WHITE));
235         game.setPiece(0, 0, new King(Player.BLACK));
236         game.tryMove(new Move(7, 1, 6, 1));
237         assertTrue(game.isDraw());
238     }
239
240     @Test

```

```
241     public void testMove() {
242         Move move = new Move();
243         move = new Move(7, 1, 6, 1);
244         assertTrue(move.toString()
245             .equals("Move [fromRow=7, fromColumn=1, toRow=6, toColumn=1]"))
246     }
247
248
249 }
250
```