

```

1 package Pieces;
2
3 import Game.Board;
4 import Game.Colour;
5 import Game.Player;
6
7 /**
8  * This class represents the Queen piece
9  *
10  * @author Param
11  */
12 public class Queen extends Piece {
13
14     public boolean canMove;
15
16     public Queen(Colour colour) {
17         super(PieceType.Queen, colour, 9);
18     }
19
20     @Override
21     public int threats(Board board, int row, int column) {
22         Piece[][] currentBoard = board.getBoard();
23         Piece toExamine;
24         int threatened = 0;
25         // check up
26         if (row > 0) {
27             for (int x = row - 1; x >= 0; x--) {
28                 toExamine = currentBoard[x][column];
29                 if (toExamine != null) {
30                     if (this.isOppositeColour(toExamine)) {
31                         threatened += toExamine.weight;
32                     }
33                     break;
34                 }
35             }
36         }
37         // check down
38         if (row < 7) {
39             for (int x = row + 1; x <= 7; x++) {
40                 toExamine = currentBoard[x][column];
41                 if (toExamine != null) {
42                     if (this.isOppositeColour(toExamine)) {
43                         threatened += toExamine.weight;
44                     }
45                     break;
46                 }
47             }
48         }
49         // check left
50         if (column > 0) {
51             for (int y = column - 1; y >= 0; y--) {
52                 toExamine = currentBoard[row][y];
53                 if (toExamine != null) {
54                     if (this.isOppositeColour(toExamine)) {
55                         threatened += toExamine.weight;
56                     }
57                     break;
58                 }
59             }
60         }
61         // check right
62         if (column < 7) {
63             for (int y = column + 1; y <= 7; y++) {
64                 toExamine = currentBoard[row][y];
65                 if (toExamine != null) {
66                     if (this.isOppositeColour(toExamine)) {
67                         threatened += toExamine.weight;
68                     }
69                     break;

```

```

70         }
71     }
72 }
73 int posX = row;
74 int posY = column;
75 // diagonal top-left
76 while (posx > 0 && posY > 0) {
77     toExamine = currentBoard[posx--][posy--];
78     if (toExamine != null) {
79         if (this.isOppositeColour(toExamine)) {
80             threatened += toExamine.weight;
81         }
82         break;
83     }
84 }
85 posX = row;
86 posY = column;
87 // diagonal top-right
88 while (posx > 0 && posY < 7) {
89     toExamine = currentBoard[posx--][posy++];
90     if (toExamine != null) {
91         if (this.isOppositeColour(toExamine)) {
92             threatened += toExamine.weight;
93         }
94         break;
95     }
96 }
97 posX = row;
98 posY = column;
99 // diagonal bottom-left
100 while (posx < 7 && posY > 0) {
101     toExamine = currentBoard[posx++][posy--];
102     if (toExamine != null) {
103         if (this.isOppositeColour(toExamine)) {
104             threatened += toExamine.weight;
105         }
106         break;
107     }
108 }
109 posX = row;
110 posY = column;
111 // diagonal bottom-right
112 while (posx < 7 && posY < 7) {
113     toExamine = currentBoard[posx++][posy++];
114     if (toExamine != null) {
115         if (this.isOppositeColour(toExamine)) {
116             threatened += toExamine.weight;
117         }
118         break;
119     }
120 }
121 return threatened;
122 }
123
124 @Override
125 public int[][] attacks(Board board, int row, int column) {
126     Piece[][] currentBoard = board.getBoard();
127     Piece toExamine;
128     int[][] attacked = new int[8][8];
129     if (row > 0) {
130         // check up
131         for (int x = row - 1; x > 0; x--) {
132             toExamine = currentBoard[x][column];
133             attacked[x][column]++;
134             if (toExamine != null) {
135                 attacked[x][column]--;
136                 break;
137             }
138         }

```

```

139     }
140     if (row < 7) {
141         // check down
142         for (int x = row + 1; x < 8; x++) {
143             toExamine = currentBoard[x][column];
144             attacked[x][column]++;
145             if (toExamine != null) {
146                 attacked[x][column]--;
147                 break;
148             }
149         }
150     }
151     if (column > 0) {
152         // check left
153         for (int y = column - 1; y > 0; y--) {
154             toExamine = currentBoard[row][y];
155             attacked[row][y]++;
156             if (toExamine != null) {
157                 attacked[row][y]--;
158                 break;
159             }
160         }
161     }
162     if (column < 7) {
163         // check right
164         for (int y = column + 1; y < 8; y++) {
165             toExamine = currentBoard[row][y];
166             attacked[row][y]++;
167             if (toExamine != null) {
168                 attacked[row][y]--;
169                 break;
170             }
171         }
172     }
173     int posx = row;
174     int posy = column;
175     // diagonal top-left
176     while (posx > 0 && posy > 0) {
177         toExamine = currentBoard[posx--][posy--];
178         attacked[posx][posy]++;
179         if (toExamine != null) {
180             attacked[posx][posy]--;
181             break;
182         }
183     }
184     posx = row;
185     posy = column;
186     // diagonal top-right
187     while (posx > 0 && posy < 7) {
188         toExamine = currentBoard[posx--][posy++];
189         attacked[posx][posy]++;
190         if (toExamine != null) {
191             attacked[posx][posy]--;
192             break;
193         }
194     }
195     posx = row;
196     posy = column;
197     // diagonal bottom-left
198     while (posx < 7 && posy > 0) {
199         toExamine = currentBoard[posx++][posy--];
200         attacked[posx][posy]++;
201         if (toExamine != null) {
202             attacked[posx][posy]--;
203             break;
204         }
205     }
206     posx = row;
207     posy = column;

```

```

208         // diagonal bottom-right
209         while (posx < 7 && posy < 7) {
210             toExamine = currentBoard[posx++][posy++];
211             attacked[posx][posy]++;
212             if (toExamine != null) {
213                 attacked[posx][posy]--;
214                 break;
215             }
216         }
217         return attacked;
218     }
219
220     @Override
221     public boolean[][] validMoves(Player opponent, Board board, int row, int column) {
222         Piece[][] currentBoard = board.getBoard();
223         Piece toExamine;
224         // reset to false and check
225         canMove = false;
226         boolean[][] validPositions = new boolean[8][8];
227         if (row > 0) {
228             // check up
229             for (int x = row - 1; x >= 0; x--) {
230                 toExamine = currentBoard[x][column];
231                 validPositions[x][column] = true;
232                 if (toExamine != null) {
233                     if (this.isOppositeColour(toExamine)) {
234                         validPositions[x][column] = false;
235                         break;
236                     }
237                     canMove = true;
238                     break;
239                 }
240                 canMove = true;
241             }
242         }
243         if (row < 7) {
244             // check down
245             for (int x = row + 1; x <= 7; x++) {
246                 toExamine = currentBoard[x][column];
247                 validPositions[x][column] = true;
248                 if (toExamine != null) {
249                     if (this.isOppositeColour(toExamine)) {
250                         validPositions[x][column] = false;
251                         break;
252                     }
253                     canMove = true;
254                     break;
255                 }
256                 canMove = true;
257             }
258         }
259         if (column > 0) {
260             // check left
261             for (int y = column - 1; y >= 0; y--) {
262                 toExamine = currentBoard[row][y];
263                 validPositions[row][y] = true;
264                 if (toExamine != null) {
265                     if (this.isOppositeColour(toExamine)) {
266                         validPositions[row][y] = false;
267                         break;
268                     }
269                     canMove = true;
270                     break;
271                 }
272                 canMove = true;
273             }
274         }
275         if (column < 7) {
276             // check right

```

```

277     for (int y = column + 1; y <= 7; y++) {
278         toExamine = currentBoard[row][y];
279         validPositions[row][y] = true;
280         if (toExamine != null) {
281             if (this.isOppositeColour(toExamine)) {
282                 validPositions[row][y] = false;
283                 break;
284             }
285             canMove = true;
286             break;
287         }
288         canMove = true;
289     }
290 }
291 int posX = row;
292 int posY = column;
293 // diagonal top-left
294 while (posx > 0 && posY > 0) {
295     toExamine = currentBoard[posx--][posy--];
296     validPositions[posx][posy] = true;
297     if (toExamine != null) {
298         if (!this.isOppositeColour(toExamine)) {
299             validPositions[posx][posy] = false;
300             break;
301         }
302         canMove = true;
303         break;
304     }
305     canMove = true;
306 }
307 posX = row;
308 posY = column;
309 // diagonal top-right
310 while (posx > 0 && posY < 7) {
311     toExamine = currentBoard[posx--][posy++];
312     validPositions[posx][posy] = true;
313     if (toExamine != null) {
314         if (!this.isOppositeColour(toExamine)) {
315             validPositions[posx][posy] = false;
316             break;
317         }
318         canMove = true;
319         break;
320     }
321     canMove = true;
322 }
323 posX = row;
324 posY = column;
325 // diagonal bottom-left
326 while (posx < 7 && posY > 0) {
327     toExamine = currentBoard[posx++][posy--];
328     validPositions[posx][posy] = true;
329     if (toExamine != null) {
330         if (!this.isOppositeColour(toExamine)) {
331             validPositions[posx][posy] = false;
332             break;
333         }
334         canMove = true;
335         break;
336     }
337     canMove = true;
338 }
339 posX = row;
340 posY = column;
341 // diagonal bottom-right
342 while (posx < 7 && posY < 7) {
343     toExamine = currentBoard[posx++][posy++];
344     validPositions[posx][posy] = true;
345     if (toExamine != null) {

```

```
346         if (!this.isOppositeColour(toExamine)) {
347             validPositions[posx][posy] = false;
348             break;
349         }
350         canMove = true;
351         break;
352     }
353     canMove = true;
354 }
355 return validPositions;
356 }
357
358 @Override
359 public boolean validSpecial() {
360     return false;
361 }
362
363 @Override
364 public void modifySpecial() {
365     // nothing
366 }
367
368 @Override
369 public String printToBoard() {
370     return this.colour == Colour.White ? "\u2655" : "\u265B";
371 }
372
373 @Override
374 public String printToLog() {
375     return "Q";
376 }
377
378 @Override
379 public boolean getCanMove() {
380     return canMove;
381 }
382 }
383
```