

```

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

```

```

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

```

```

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

```

```
208     }
209
210     @Override
211     public void modifySpecial() {
212         hasMoved = true;
213     }
214
215     @Override
216     public String printToBoard() {
217         return this.colour == Colour.White ? "\u2656" : "\u265C";
218     }
219
220     @Override
221     public String printToLog() {
222         return "R";
223     }
224
225     @Override
226     public boolean getCanMove() {
227         return canMove;
228     }
229
230 }
231
```