

```

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

```

```

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

```

```

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

```

```
208     public void modifySpecial() {
209         // nothing
210     }
211
212     @Override
213     public String printToBoard() {
214         return this.colour == Colour.White ? "\u2657" : "\u265D";
215     }
216
217     @Override
218     public String printToLog() {
219         return "B";
220     }
221
222     @Override
223     public boolean getCanMove() {
224         return canMove;
225     }
226
227 }
228
```