```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Text;
 4
 5  namespace NQueens
 6  {
 7      /// <summary>
 8      /// This object represents a chessboard for the purposes of this program.
          squares on the board are stored in a 2D Square array
 9      /// </summary>
10      class ChessBoard
11      {
12          public Square[][] board;
13          int numQueens = 0;
14          public int queenHits = 0;
15
16          public ChessBoard(int size)
17          {
18              //Create structure
19              board = new Square[size][];
20              for (int i = 0; i < size; i++)
21              {
22                  board[i] = new Square[size];
23              }
24
25              //Initilize
26              for (int i = 0; i < size; i++)
27              {
28                  for (int j = 0; j < size; j++)
29                  {
30                      board[i][j] = new Square();
31                  }
32              }
33
34          }
35
36          /// <summary>
37          /// Marks the square at the given coordinate as a queen,
38          /// Marks the hits on each of the squares she can attack,
39          /// Increments the number of queens
40          /// </summary>
41          /// <param name="c">Coordinate for queen to be placed</param>
42          public void AddQueen(Coord c)
43          {
44              board[c.row][c.col].isQueen = true;
45              numQueens++;
46              manipulateBoard(c);
47              CalcQueenHits();
48          }
```

```
49
50          /// <summary>
51          /// Marks the square at the given coordinate as not a queen,
52          /// Removes the former queens marks,
53          /// decriment the number of queens
54          /// </summary>
55          /// <param name="c">Coordinate for queen to be removed</param>
56          public void RemoveQueen(Coord c)
57          {
58              board[c.row][c.col].isQueen = false;
59              numQueens--;
60              manipulateBoard(c);
61              CalcQueenHits();
62          }
63
64          /// <summary>
65          /// Manipulates the board by adding or removing marks from squares
              based on whether the given location is a queen
66          /// </summary>
67          /// <param name="c">Coordinate of starting square</param>
68          private void manipulateBoard(Coord c)
69          {
70              //Determins whether to increment or decreminet marks based on if
                  coord is queen
71              int val = (board[c.row][c.col].isQueen) ? 1 : -1;
72
73              int length = board.Length;
74
75              for (int i = 1; i < length; i++)
76              {
77                  //Left
78                  if (c.col - i >= 0)
79                  {
80                      board[c.row][c.col - i].MarkHit(val);
81                  }
82
83                  //Right
84                  if (c.col + i < length)
85                  {
86                      board[c.row][c.col + i].MarkHit(val);
87                  }
88
89                  //Down
90                  if (c.row + i < length)
91                  {
92                      //Down
93                      board[c.row + i][c.col].MarkHit(val);
94
95                      //Down Right
```

```
96                      if (c.col + i < length)
97                      {
98                          board[c.row + i][c.col + i].MarkHit(val);
99                      }
100
101                     //Down Left
102                     if (c.col - i >= 0)
103                     {
104                         board[c.row + i][c.col - i].MarkHit(val);
105                     }
106                 }
107
108                 //Up
109                 if(c.row - i >= 0)
110                 {
111                     //Up
112                     board[c.row - i][c.col].MarkHit(val);
113
114                     //Up Left
115                     if (c.col - i >= 0)
116                     {
117                         board[c.row - i][c.col - i].MarkHit(val);
118                     }
119
120                     //Up Right
121                     if (c.col + i < length)
122                     {
123                         board[c.row - i][c.col + i].MarkHit(val);
124                     }
125                 }
126
127
128             }
129         }
130         /// <summary>
131         /// Iteratively sums the total number of hits of every queen on the
              board
132         ///
133         /// </summary>
134         private void CalcQueenHits()
135         {
136             queenHits = 0;
137             for (int k = 0; k < board.Length; k++)
138             {
139                 for (int j = 0; j < board.Length; j++)
140                 {
141                     if (board[k][j].isQueen)
142                     {
143                         queenHits += board[k][j].GetNumHits();
```

```
144                     }
145                 }
146             }
147         }
148
149         public int GetNumQueens() { return numQueens; }
150
151         /// <summary>
152         /// Allows the ChessBoard to be cloned. See documentation for ICloneable
153         /// </summary>
154         /// <returns>Clone of this object</returns>
155         public ChessBoard Clone()
156         {
157             int length = board.Length;
158             ChessBoard newChessboard = new ChessBoard(length);
159
160             for (int i = 0; i < length; i++)
161             {
162                 for (int j = 0; j < length; j++)
163                 {
164                     newChessboard.board[i][j] = board[i][j].Clone();
165                 }
166             }
167
168             newChessboard.numQueens = numQueens;
169             return newChessboard;
170         }
171
172
173     }
174 }
175
```