```
 1  using System.Collections.Generic;
 2
 3
 4  namespace Assign_1
 5  {
 6      /** Matthew Alunni
 7      *  5865647
 8      *  COSC 3P71
 9      *  Assignment 1 **/
10
11
12
13      /** this class is for information about queens **/
14      public class Position
15      {
16          // the cost of finding a specific colution
17          public int Cost { get; set; }
18
19          public int Line { get; set; }
20          public int Row { get; set; }
21          public Position Parent { get; set; }
22
23          public Position(int Line, int Row, Position Parent)
24          {
25              this.Line = Line;
26              this.Row = Row;
27              this.Parent = Parent;
28          }
29
30          /** this method finds a solution by checking if a nearby queen has threats, then if a
31           * solution is reached, it adds it to a list of solutions**/
32          public void FindSolution(List<Solution> solutions,  int numberOfQueens,  int cost)
33          {
34              System.Diagnostics.Debug.WriteLine(cost);
35
36              Cost ++ ;
37
38              if (Line == numberOfQueens) // last line (=number of queens) reached: solution
39              {
40                  if (solutions == null)
41                  {
42                      solutions = new List<Solution>();
43                  }
44
45                  var solution = new Solution
46                  {
47                      Position = this
48                  };
49
```

```
50                  // calculate heuristic cost
51                  // the solution cost is the sum os the cost of each position found
52                  var pos = this;
53                  while (pos.Parent != null)
54                  {
55                      solution.Cost += pos.Cost;
56                      pos = pos.Parent;
57                  }
58
59                  solutions.Add(solution);
60                  return;
61              }
62              else
63              {
64                  for (var r = 0; r < numberOfQueens; r++) // try all rows in next line
65                  {
66                      // check threats for all queens in previous lines
67                      var queenAbove = this;
68                      while (!HasVerticalThreat(queenAbove, r)                    // vertical threat?
69                             && !HasDiagonalLeftThreat(queenAbove, r)             // diagonal threat left?
70                             && !HasDiagonalRightThreat(queenAbove, r))           // diagonal threat right?
71                      {
72                          queenAbove = queenAbove.Parent;
                            // repeat check for all queens in previous lines
73                      }
74
75                      if (queenAbove.Line == 0)                                   // back to first queen - no threat found
76                      {
77                          new Position(Line + 1, r, this).FindSolution(solutions, numberOfQueens, Cost); // put queen on next line
78                      }
79                  }
80              }
81          }
82
83          /** this method checks if the queen at position has a vertical threat**/
84          public bool HasVerticalThreat(Position queen, int row)
85          {
86
87              if (queen.Row >= 0 && row != queen.Row) // First row is Ok and different row is Ok
88              {
89
90                  return false;
```

```
91              }
92              else
93              {
94                  return true;
95              }
96          }
97
98
99          /** this method checks if the queen at position has a diagonal threat**/
100         public bool HasDiagonalLeftThreat(Position queen, int row)
101         {
102
103             if (row - queen.Row != Line + 1 - queen.Line)
104             {
105
106                 return false;
107             }
108             else
109             {
110                 return true;
111             }
112         }
113
114
115         /** this method checks if the queen at position has a diagonal threat**/
116         public bool HasDiagonalRightThreat(Position queen, int row)
117         {
118
119             if (queen.Row - row != Line + 1 - queen.Line)
120             {
121
122                 return false;
123             }
124             else
125             {
126                 Cost++;
127                 return true;
128             }
129         }
130     }
131 }
132
```