NEW YORK CITY COLLEGE OF TECHNOLOGY THE CITY UNIVERSITY OF NEW YORK
Department of Computer Engineering Technology *300 Jay Street, Brooklyn, NY 11201-1909*

LAB REPORT

CET 3640 – OL30

(SOFTWARE FOR COMPUTER CONTROL)

LAB#7

JAVA PROGRAM EIGHT QUEENS

Name: Puja Roy

Date: 5/6/22

Due Date: 5/7/22

DESCRIPTION OF THE LAB:

In this lab, I wrote a java program in Eclipse that solves a puzzle for the Eighth Queens Problem. I created a class named NQueenProblem. After that, I wrote a function that prints the solution of the overall chess board results. The function checks if a queen can be placed on a chessboard (row or column) since it is called recursively when “col” queens are placed already in columns from 0 to column -1. The left side of the chess board displays if the queens are attacking. After that, I wrote several loops to verify the left side, upper diagonal, and lower diagonal attacks made by the queen of the board. The recursive function solves the Eight queen problem. As a result, the Eight Queen problem was solved by recursive backtracking because the function returns false if queens can’t be placed. On the other hand, if the queen is placed, then the function returns true and prints the placement of the queen in terms of binary numbers 1 and 0.

```java
//NAME: Puja Roy
//DATE: 5/6/22
public class NQueenProblem {
    final static int N = 8;

    // This is the function that prints the solution on the output screen
    void printSolution(int chessBoard[][]) {
    for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++)
    System.out.print(" " + chessBoard[i][j]
    + " ");
    System.out.println();
    }
}

    /* This function checks whether a queen can be placed on chessBoard[row][col].
    This function is called recursively when "col" queens are placed
    already in columns from 0 to column -1. The left side might show if the queens are attacking.
    */
    boolean isSecured(int chessBoard[][], int row, int col) {
    int i, j;

    /* A for loop that checks the left side of attacks */
    for (i = 0; i < col; i++)
    if (chessBoard[row][i] == 1)
    return false;

    /* A for loop that checks the upper diagonal on left side of the board*/
    for (j = col, i = row; i >= 0 && j >= 0; i--, j--)
    if (chessBoard[i][j] == 1)
    return false;
```

Console ×

<terminated> NQueenProblem [Java Application] C:\Users\pujar\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20

```
1  0  0  0  0  0  0  0
0  0  0  0  0  0  1  0
0  0  0  0  1  0  0  0
0  0  0  0  0  0  0  1
0  1  0  0  0  0  0  0
0  0  0  1  0  0  0  0
0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0
```

```
32
33      /* A for loop that checks the lower diagonal on left side of the board*/
34      for (j = col, i = row; j >= 0 && i < N; i++, j--) {
35      if (chessBoard[i][j] == 1)
36      return false;
37         }
38      return true;
39      }
40
41      /* A recursive function that solves the N Queen problem */
42      boolean NQueenProblemSolve(int chessBoard[][], int col) {
43      /* If all queens are placed on the board then return the output is true */
44      if (col >= N)
45      return true;
46
47      /* Places the queen in all rows in the designated column */
48      for (int i = 0; i < N; i++) {
49      //] Checks if the queen can be placed on chessBoard[i][col] */
50      if (isSecured(chessBoard, i, col)) {
51      /* Places the queen in chessBoard[i][col] */
52      chessBoard[i][col] = 1;
53
54      /* recurse to place rest of the queens */
55      if (NQueenProblemSolve(chessBoard, col + 1) == true)
56      return true;
57
58      /* If placing queen in chessBoard[i][col]
59      doesn't lead to a solution then
60      remove queen from chessBoard[i][col] */
61      chessBoard[i][col] = 0; // If there is no solution just BACKTRACK
62      }
```

Console ×

<terminated> NQueenProblem [Java Application] C:\Users\pujar\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x

```
1  0  0  0  0  0  0  0
0  0  0  0  0  0  1  0
0  0  0  0  1  0  0  0
0  0  0  0  0  0  0  1
0  1  0  0  0  0  0  0
0  0  0  1  0  0  0  0
0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0
```

```
63      }
64
65      /* Returns false if the queen cannot be placed in any of the row in the designated column */
66      return false;
67      }
68
69      /* 8 queen problem is solved using Backtracking recursively.
70      The NQueenProblemSolve function returns false if queens
71      cannot be placed, else,it returns true and prints placement of queens
72      and empty in the form of 1s and 0s. */
73      boolean solveNQ()
74      {
75      int chessBoard[][] = new int[N][N];
76
77      if (NQueenProblemSolve(chessBoard, 0) == false) {
78      System.out.print("Solution does not exist");
79      return false;
80      }
81
82      printSolution(chessBoard);
83      return true;
84      }
85
86      // Driver program to test above function
87      public static void main(String args[])
88      {
89      NQueenProblem Queen = new NQueenProblem();
90      Queen.solveNQ();
91      }
92      }
```

Console ×

<terminated> NQueenProblem [Java Application] C:\Users\pujar\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.

```
1  0  0  0  0  0  0  0
0  0  0  0  0  0  1  0
0  0  0  0  1  0  0  0
0  0  0  0  0  0  0  1
0  1  0  0  0  0  0  0
0  0  0  1  0  0  0  0
0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0
```