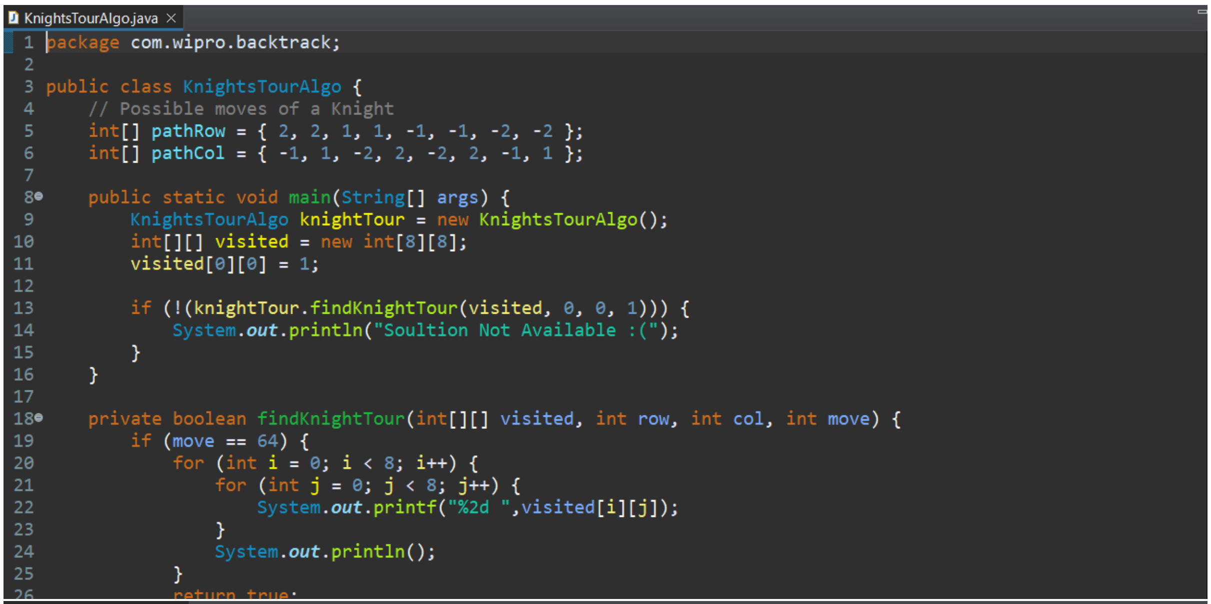
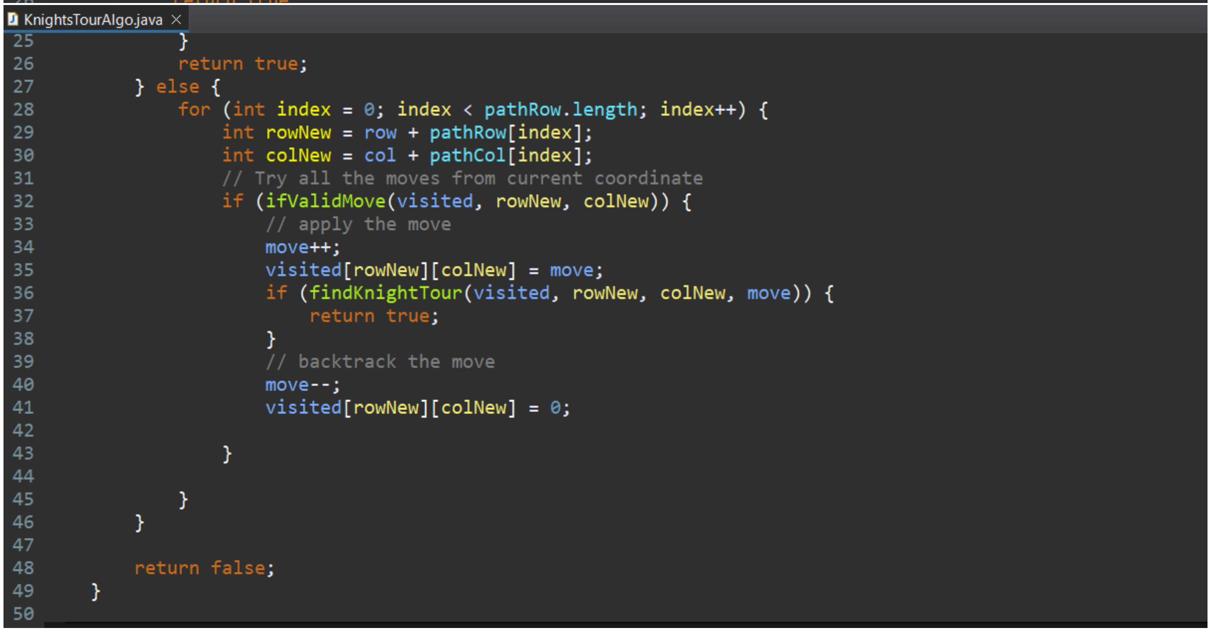
Task 1: The Knight's Tour Problem

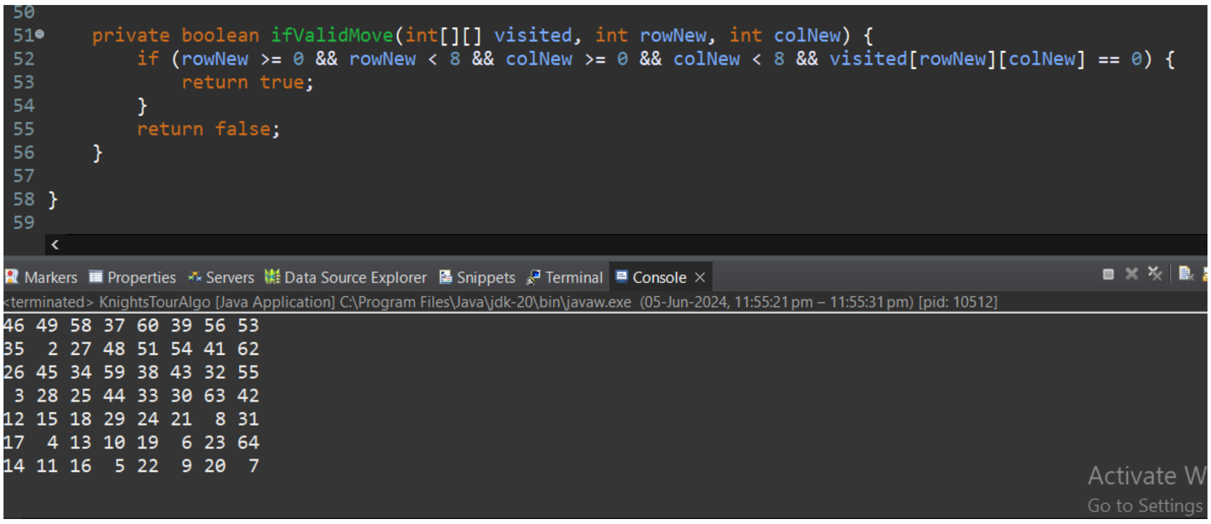
Create a function bool Solve Knights Tour(int[] board, int moveX, int moveY, int moveCount, int xMove, int yMove) that attempts to solve the Knight's Tour problem using backtracking. The function should relum true if a solution exists and false otherwise. The board represents the chessboard, moveX and movey are the the current coordinates of the knight, moveCount is the current move count, and xMove[], yMove[] are the possible next moves for the knight. Fill the chessboard such that the knight visits every square exactly once. Keep the chessboard size to 8x8.



```java
package com.wipro.backtrack;

public class KnightsTourAlgo {
    // Possible moves of a Knight
    int[] pathRow = { 2, 2, 1, 1, -1, -1, -2, -2 };
    int[] pathCol = { -1, 1, -2, 2, -2, 2, -1, 1 };

    public static void main(String[] args) {
        KnightsTourAlgo knightTour = new KnightsTourAlgo();
        int[][] visited = new int[8][8];
        visited[0][0] = 1;

        if (!(knightTour.findKnightTour(visited, 0, 0, 1))) {
            System.out.println("Soultion Not Available :(");
        }
    }

    private boolean findKnightTour(int[][] visited, int row, int col, int move) {
        if (move == 64) {
            for (int i = 0; i < 8; i++) {
                for (int j = 0; j < 8; j++) {
                    System.out.printf("%2d ",visited[i][j]);
                }
                System.out.println();
            }
```



```java
            }
            return true;
        } else {
            for (int index = 0; index < pathRow.length; index++) {
                int rowNew = row + pathRow[index];
                int colNew = col + pathCol[index];
                // Try all the moves from current coordinate
                if (ifValidMove(visited, rowNew, colNew)) {
                    // apply the move
                    move++;
                    visited[rowNew][colNew] = move;
                    if (findKnightTour(visited, rowNew, colNew, move)) {
                        return true;
                    }
                    // backtrack the move
                    move--;
                    visited[rowNew][colNew] = 0;

                }

            }
        }

        return false;
    }

```

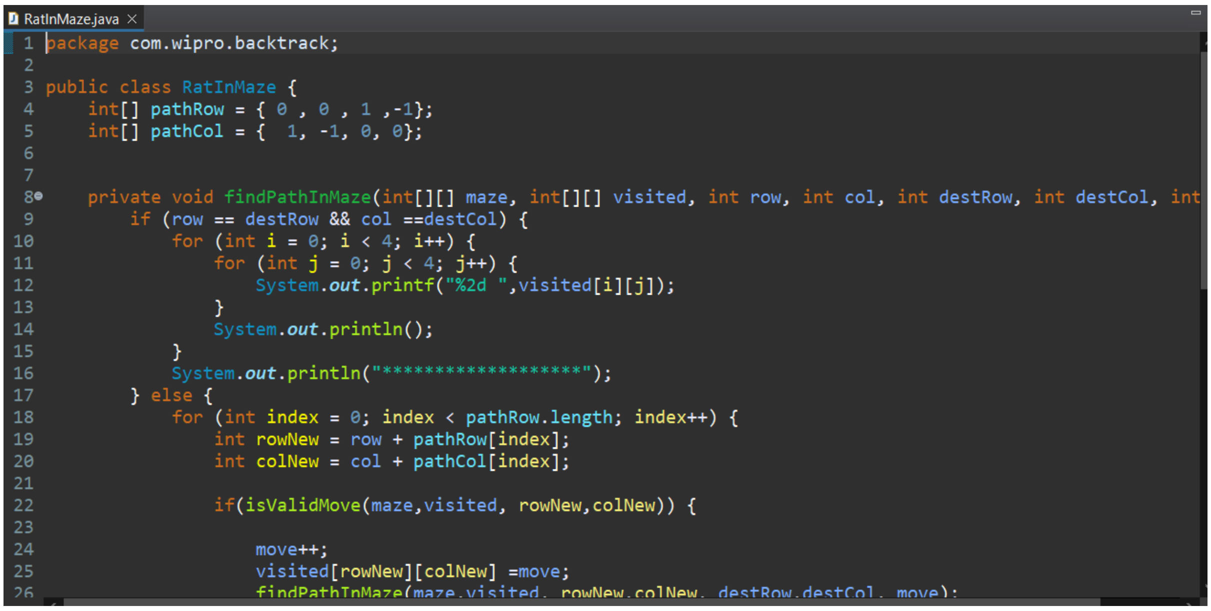


```java

    private boolean ifValidMove(int[][] visited, int rowNew, int colNew) {
        if (rowNew >= 0 && rowNew < 8 && colNew >= 0 && colNew < 8 && visited[rowNew][colNew] == 0) {
            return true;
        }
        return false;
    }

}

```

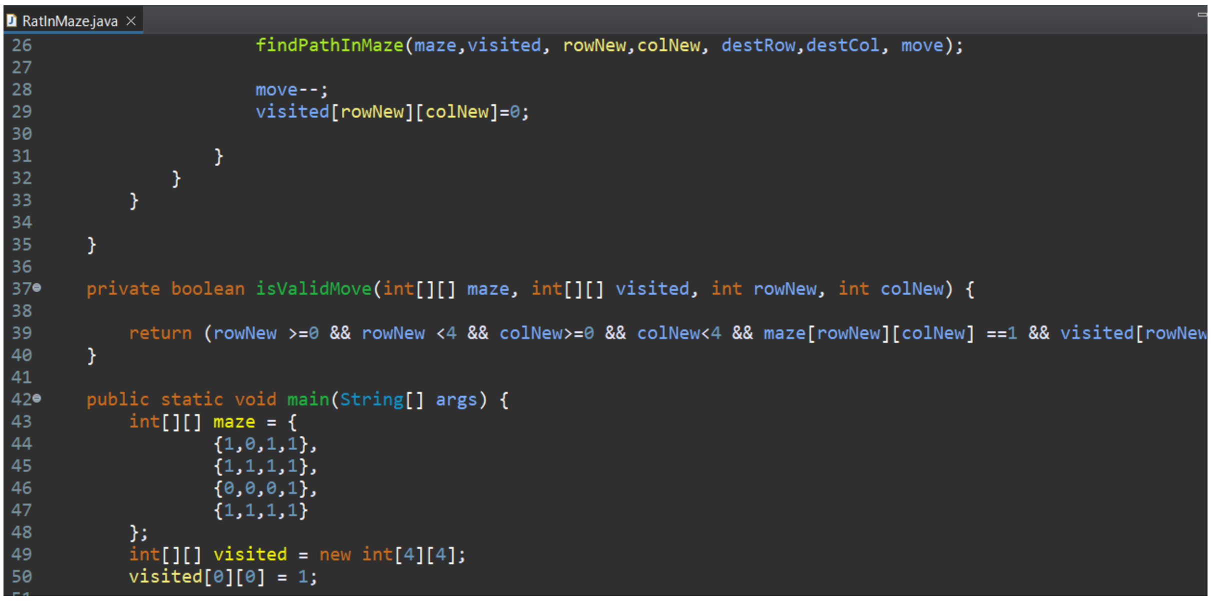
```
<terminated> KnightsTourAlgo [Java Application] C:\Program Files\Java\jdk-20\bin\javaw.exe  (05-Jun-2024, 11:55:21 pm – 11:55:31 pm) [pid: 10512]
46 49 58 37 60 39 56 53
35  2 27 48 51 54 41 62
26 45 34 59 38 43 32 55
 3 28 25 44 33 30 63 42
12 15 18 29 24 21  8 31
17  4 13 10 19  6 23 64
14 11 16  5 22  9 20  7
```

Task 2: Rat in a Maze

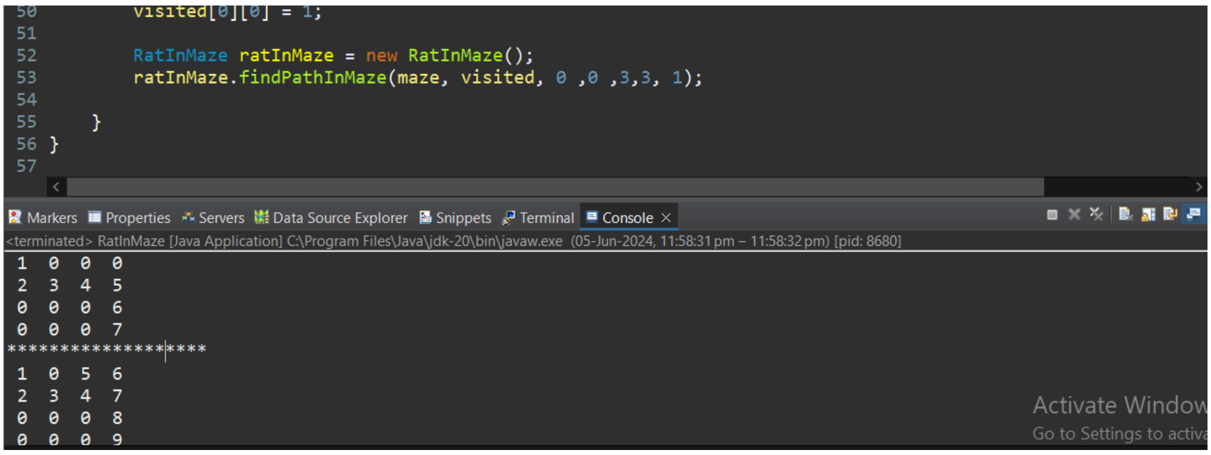
Implement a function bool SolveMaze(int[,] maze) that uses backtracking to find a path from the top left comer to the bottom right corner of a maze. The maze is represented by a 2D array where 1s are paths and Os are walls. Find a raf's path through the maze. The maze size is 6x6.



```java
package com.wipro.backtrack;

public class RatInMaze {
    int[] pathRow = { 0 , 0 , 1 ,-1};
    int[] pathCol = {  1, -1, 0, 0};


    private void findPathInMaze(int[][] maze, int[][] visited, int row, int col, int destRow, int destCol, int
        if (row == destRow && col ==destCol) {
            for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) {
                    System.out.printf("%2d ",visited[i][j]);
                }
                System.out.println();
            }
            System.out.println("******************");
        } else {
            for (int index = 0; index < pathRow.length; index++) {
                int rowNew = row + pathRow[index];
                int colNew = col + pathCol[index];

                if(isValidMove(maze,visited, rowNew,colNew)) {

                    move++;
                    visited[rowNew][colNew] =move;
                    findPathInMaze(maze,visited, rowNew,colNew, destRow,destCol, move);
```



```java
                    findPathInMaze(maze,visited, rowNew,colNew, destRow,destCol, move);

                    move--;
                    visited[rowNew][colNew]=0;

                }
            }
        }

    }

    private boolean isValidMove(int[][] maze, int[][] visited, int rowNew, int colNew) {

        return (rowNew >=0 && rowNew <4 && colNew>=0 && colNew<4 && maze[rowNew][colNew] ==1 && visited[rowNew
    }

    public static void main(String[] args) {
        int[][] maze = {
                {1,0,1,1},
                {1,1,1,1},
                {0,0,0,1},
                {1,1,1,1}
        };
        int[][] visited = new int[4][4];
        visited[0][0] = 1;
```



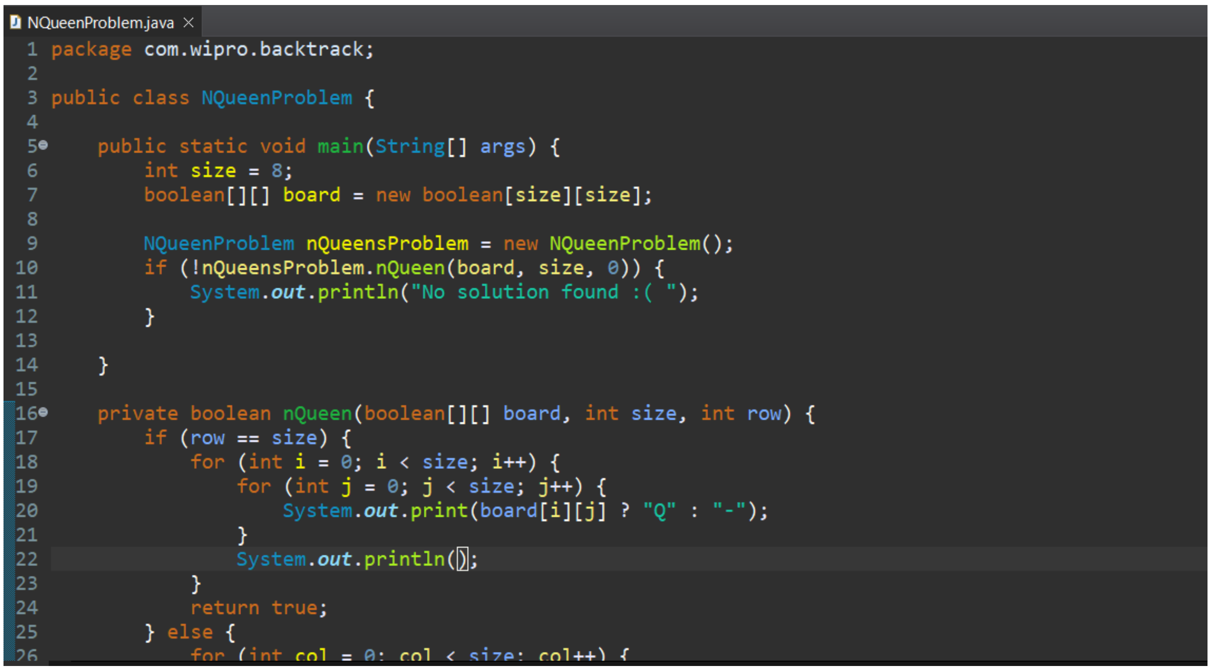
visited[0][0] = 1;

```java

        RatInMaze ratInMaze = new RatInMaze();
        ratInMaze.findPathInMaze(maze, visited, 0 ,0 ,3,3, 1);

    }
}
```

```
<terminated> RatInMaze [Java Application] C:\Program Files\Java\jdk-20\bin\javaw.exe  (05-Jun-2024, 11:58:31 pm – 11:58:32 pm) [pid: 8680]
 1  0  0  0
 2  3  4  5
 0  0  0  6
 0  0  0  7
*******************
 1  0  5  6
 2  3  4  7
 0  0  0  8
 0  0  0  9
```

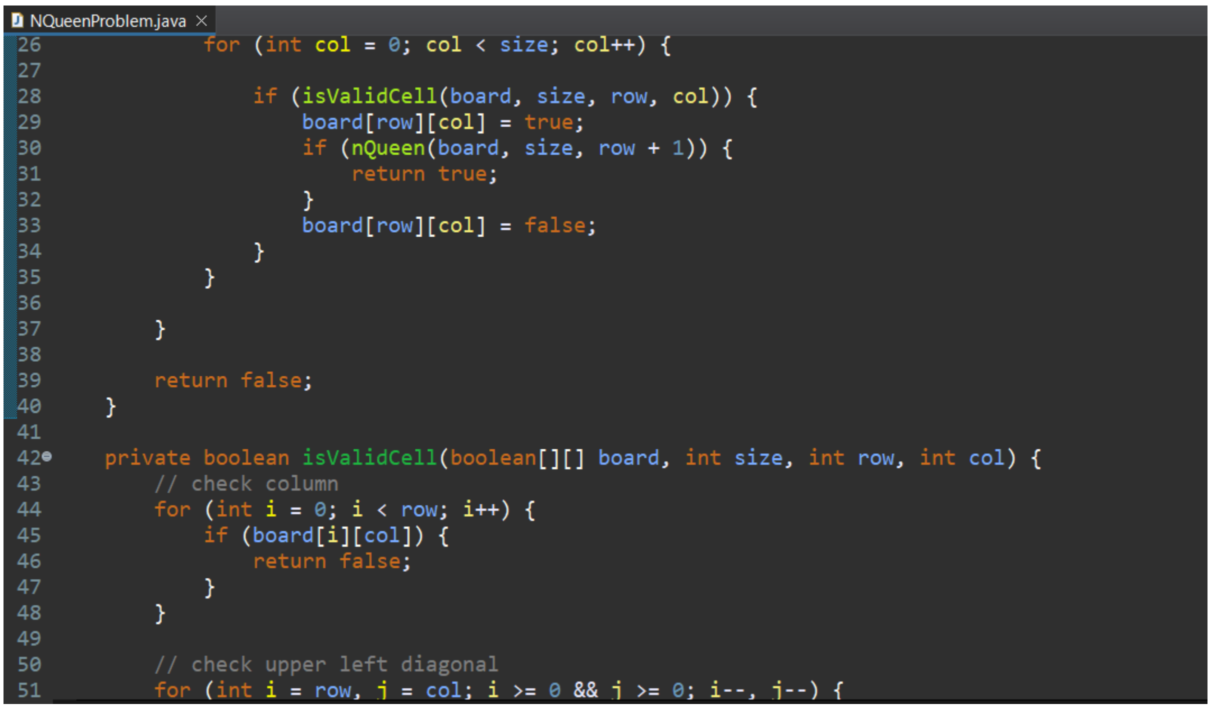
Task 3: N Queen Problem

Write a function bool SolveNQueen(int[,] board, int col) in Cil that places N queens on an Nx N chessboard so that no two queens attack each other using backtracking. Place N queens on the board such that no two queens can attack each other. Use a standard 8x8 chessboard



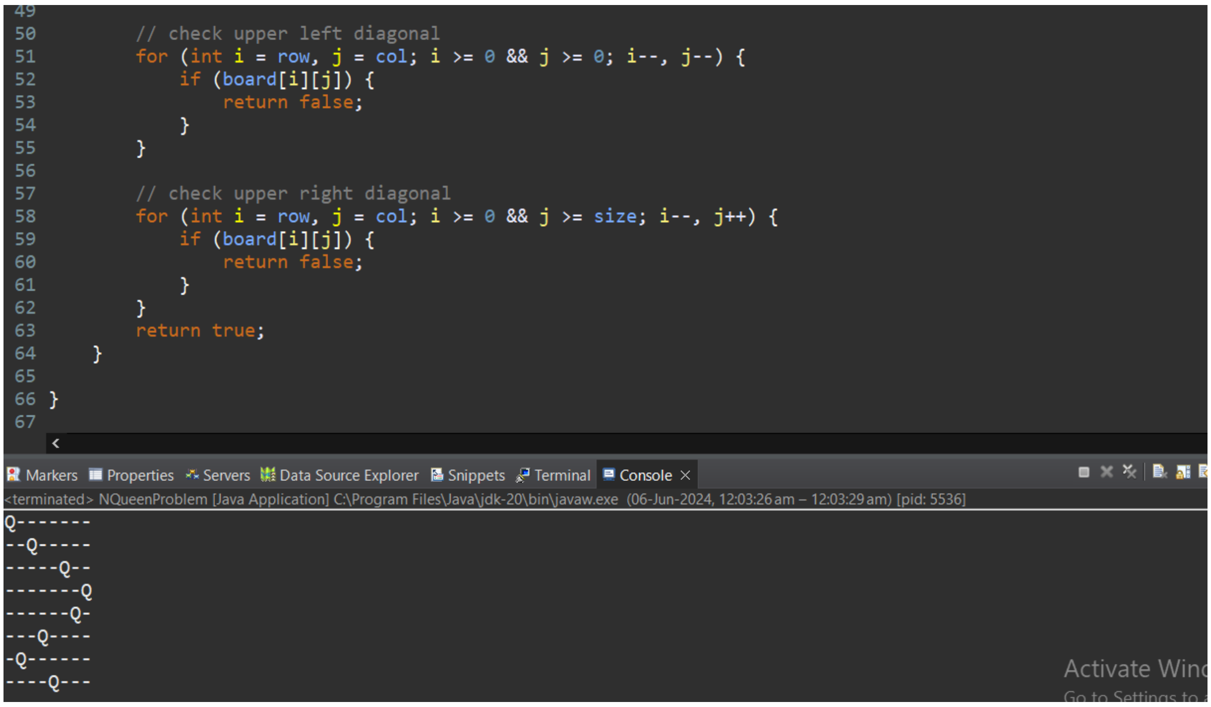
NQueenProblem.java

```java
package com.wipro.backtrack;

public class NQueenProblem {

    public static void main(String[] args) {
        int size = 8;
        boolean[][] board = new boolean[size][size];

        NQueenProblem nQueensProblem = new NQueenProblem();
        if (!nQueensProblem.nQueen(board, size, 0)) {
            System.out.println("No solution found :( ");
        }

    }

    private boolean nQueen(boolean[][] board, int size, int row) {
        if (row == size) {
            for (int i = 0; i < size; i++) {
                for (int j = 0; j < size; j++) {
                    System.out.print(board[i][j] ? "Q" : "-");
                }
                System.out.println();
            }
            return true;
        } else {
```



NQueenProblem.java

```java
            for (int col = 0; col < size; col++) {

                if (isValidCell(board, size, row, col)) {
                    board[row][col] = true;
                    if (nQueen(board, size, row + 1)) {
                        return true;
                    }
                    board[row][col] = false;
                }
            }

        }

        return false;
    }

    private boolean isValidCell(boolean[][] board, int size, int row, int col) {
        // check column
        for (int i = 0; i < row; i++) {
            if (board[i][col]) {
                return false;
            }
        }

        // check upper left diagonal
        for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
```



```java
        // check upper left diagonal
        for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
            if (board[i][j]) {
                return false;
            }
        }

        // check upper right diagonal
        for (int i = row, j = col; i >= 0 && j >= size; i--, j++) {
            if (board[i][j]) {
                return false;
            }
        }
        return true;
    }

}
```

```
<terminated> NQueenProblem [Java Application] C:\Program Files\Java\jdk-20\bin\javaw.exe  (06-Jun-2024, 12:03:26 am – 12:03:29 am) [pid: 5536]
Q-------
--Q-----
-----Q--
-------Q
------Q-
---Q----
-Q------
----Q---
```