# CSE 101

C Project



## Project Title : IPL Cricket Tournament

## Scheduler

Section :

**Submitted by: Shafe Ahsan**

**Submitted to : Rahul Rajput Sir**

## --: Group Members :--

1. Name : Shafe Ahsan

Roll No : 39

**ACKNOWLEDGEMENT**

We would like to express our Special thanks of

Gratitude to our university as well as Faculty Rahul Rajput Sir who gave us this opportunity to work on the project to design a program for N-Queens. Which also helped us in learning and developing skills in java Programming Language.

*Finally, I would like to thank my friends who helped me a lot to understand and in finishing this topic within limited time.*

Thanks to All !

# INTRODUCTION

The N-Queens problem is a classic example in the field of computer science and mathematics, which involves placing N chess queens on an N×N chessboard so that no two queens threaten each other. This means that no two queens can share the same row, column, or diagonal. The problem was first posed in 1848 by chess player Max Bezzel, and it has since become a well-known combinatorial problem.

# PROJECT

The main objectives of this project are:

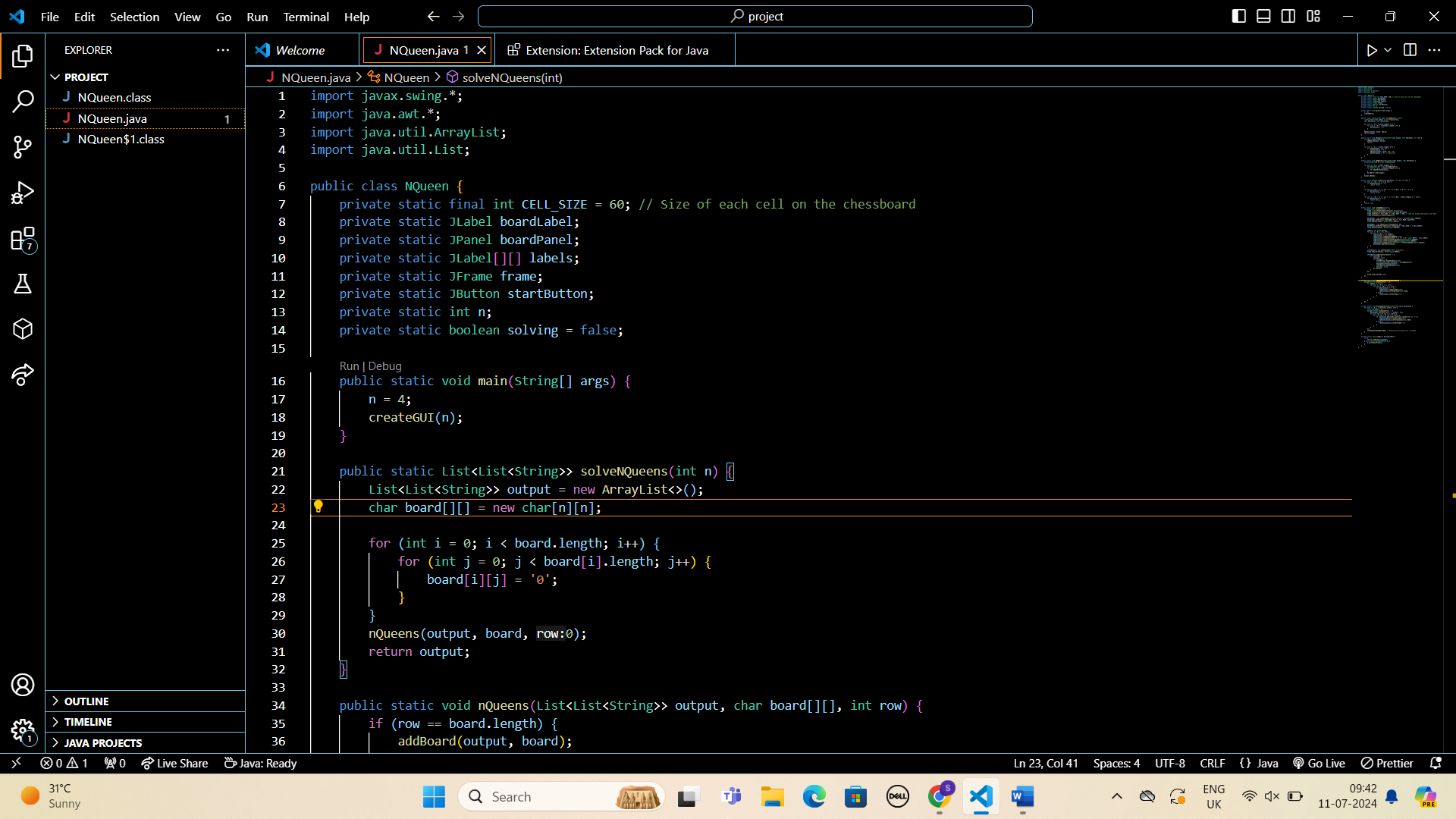
1. To develop a solution for the N-Queens problem using a backtracking algorithm.
2. To implement a graphical user interface (GUI) using Java Swing to visualize the solutions of the N-Queens problem.
3. To provide a step-by-step visualization of the backtracking algorithm as it places and removes queens on the board.

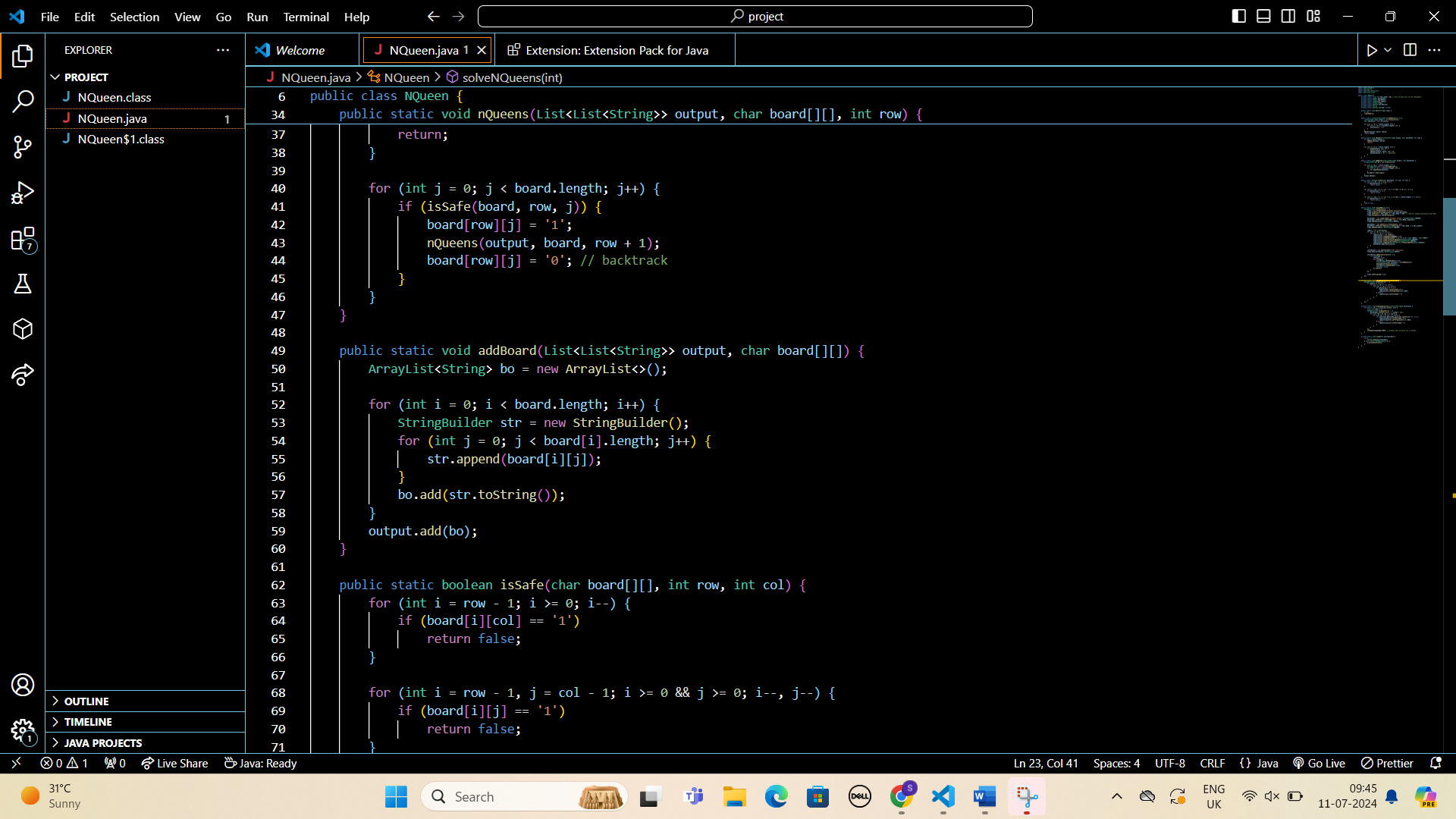
### Significance

Understanding the N-Queens problem is significant for several reasons:

1. **Algorithm Design**: The problem illustrates the concept of backtracking, which is a fundamental algorithm design technique used in various applications, such as solving puzzles, constraint satisfaction problems, and in AI for games.
2. **Optimization**: It helps in understanding optimization problems and how to approach them effectively.
3. **Visualization**: Creating a visual representation of the algorithm helps in better understanding and debugging the logic, making it an excellent educational tool for students and educators.

**CODE**

****

****

**CODE** in written form

import javax.swing.\*;

import java.awt.\*;

import java.util.ArrayList;

import java.util.List;

public class NQueen {

    private static final int CELL\_SIZE = 60; // Size of each cell on the chessboard

    private static JLabel boardLabel;

    private static JPanel boardPanel;

    private static JLabel[][] labels;

    private static JFrame frame;

    private static JButton startButton;

    private static int n;

    private static boolean solving = false;

    public static void main(String[] args) {

        n = 4;

        createGUI(n);

    }

    public static List<List<String>> solveNQueens(int n) {

        List<List<String>> output = new ArrayList<>();

        char board[][] = new char[n][n];

        for (int i = 0; i < board.length; i++) {

            for (int j = 0; j < board[i].length; j++) {

                board[i][j] = '0';

            }

        }

        nQueens(output, board, 0);

        return output;

    }

    public static void nQueens(List<List<String>> output, char board[][], int row) {

        if (row == board.length) {

            addBoard(output, board);

            return;

        }

        for (int j = 0; j < board.length; j++) {

            if (isSafe(board, row, j)) {

                board[row][j] = '1';

                nQueens(output, board, row + 1);

                board[row][j] = '0'; // backtrack

            }

        }

    }

    public static void addBoard(List<List<String>> output, char board[][]) {

        ArrayList<String> bo = new ArrayList<>();

        for (int i = 0; i < board.length; i++) {

            StringBuilder str = new StringBuilder();

            for (int j = 0; j < board[i].length; j++) {

                str.append(board[i][j]);

            }

            bo.add(str.toString());

        }

        output.add(bo);

    }

    public static boolean isSafe(char board[][], int row, int col) {

        for (int i = row - 1; i >= 0; i--) {

            if (board[i][col] == '1')

                return false;

        }

        for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--) {

            if (board[i][j] == '1')

                return false;

        }

        for (int i = row - 1, j = col + 1; i >= 0 && j < board.length; i--, j++) {

            if (board[i][j] == '1')

                return false;

        }

        return true;

    }

    public static void createGUI(int n) {

        SwingUtilities.invokeLater(() -> {

            frame = new JFrame("N-Queens Solutions");

            frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

            frame.setSize(n \* CELL\_SIZE, n \* CELL\_SIZE + 120); // +120 for window decorations and label

            frame.setLayout(new BorderLayout());

            boardLabel = new JLabel("N-Queens Solver", SwingConstants.CENTER);

            boardLabel.setFont(new Font("Serif", Font.BOLD, 20));

            frame.add(boardLabel, BorderLayout.NORTH);

            boardPanel = new JPanel(new GridLayout(n, n));

            boardPanel.setPreferredSize(new Dimension(n \* CELL\_SIZE, n \* CELL\_SIZE));

            frame.add(boardPanel, BorderLayout.CENTER);

            labels = new JLabel[n][n];

            for (int i = 0; i < n; i++) {

                for (int j = 0; j < n; j++) {

                    labels[i][j] = new JLabel();

                    labels[i][j].setOpaque(true);

                    labels[i][j].setBackground((i + j) % 2 == 0 ? Color.WHITE : Color.GRAY);

                    labels[i][j].setHorizontalAlignment(SwingConstants.CENTER);

                    labels[i][j].setVerticalAlignment(SwingConstants.CENTER);

                    labels[i][j].setBorder(BorderFactory.createLineBorder(Color.BLACK));

                    boardPanel.add(labels[i][j]);

                }

            }

            startButton = new JButton("Start Solving");

            frame.add(startButton, BorderLayout.SOUTH);

            startButton.addActionListener(e -> {

                if (!solving) {

                    solving = true;

                    new Thread(() -> {

                        startButton.setEnabled(false);

                        List<List<String>> solutions = solveNQueens(n);

                        displaySolutions(solutions);

                        startButton.setEnabled(true);

                        solving = false;

                    }).start();

                }

            });

            frame.setVisible(true);

        });

    }

    private static void updateBoardGUI(char[][] board) {

        SwingUtilities.invokeLater(() -> {

            if (labels != null) {

                for (int i = 0; i < n; i++) {

                    for (int j = 0; j < n; j++) {

                        if (board[i][j] == '1') {

                            labels[i][j].setText("Q");

                            labels[i][j].setForeground(Color.RED);

                        } else {

                            labels[i][j].setText("");

                        }

                    }

                }

            }

        });

    }

    private static void displaySolutions(List<List<String>> solutions) {

        for (int i = 0; i < solutions.size(); i++) {

            final int index = i;

            SwingUtilities.invokeLater(() -> {

                boardLabel.setText("Board " + (index + 1));

                for (int row = 0; row < n; row++) {

                    for (int col = 0; col < n; col++) {

                        if (solutions.get(index).get(row).charAt(col) == '1') {

                            labels[row][col].setText("Q");

                            labels[row][col].setForeground(Color.RED);

                        } else {

                            labels[row][col].setText("");

                        }

                    }

                }

            });

            sleep(2000); // Display each solution for 2 seconds

        }

    }

    private static void sleep(int milliseconds) {

        try {

            Thread.sleep(milliseconds);

        } catch (InterruptedException e) {

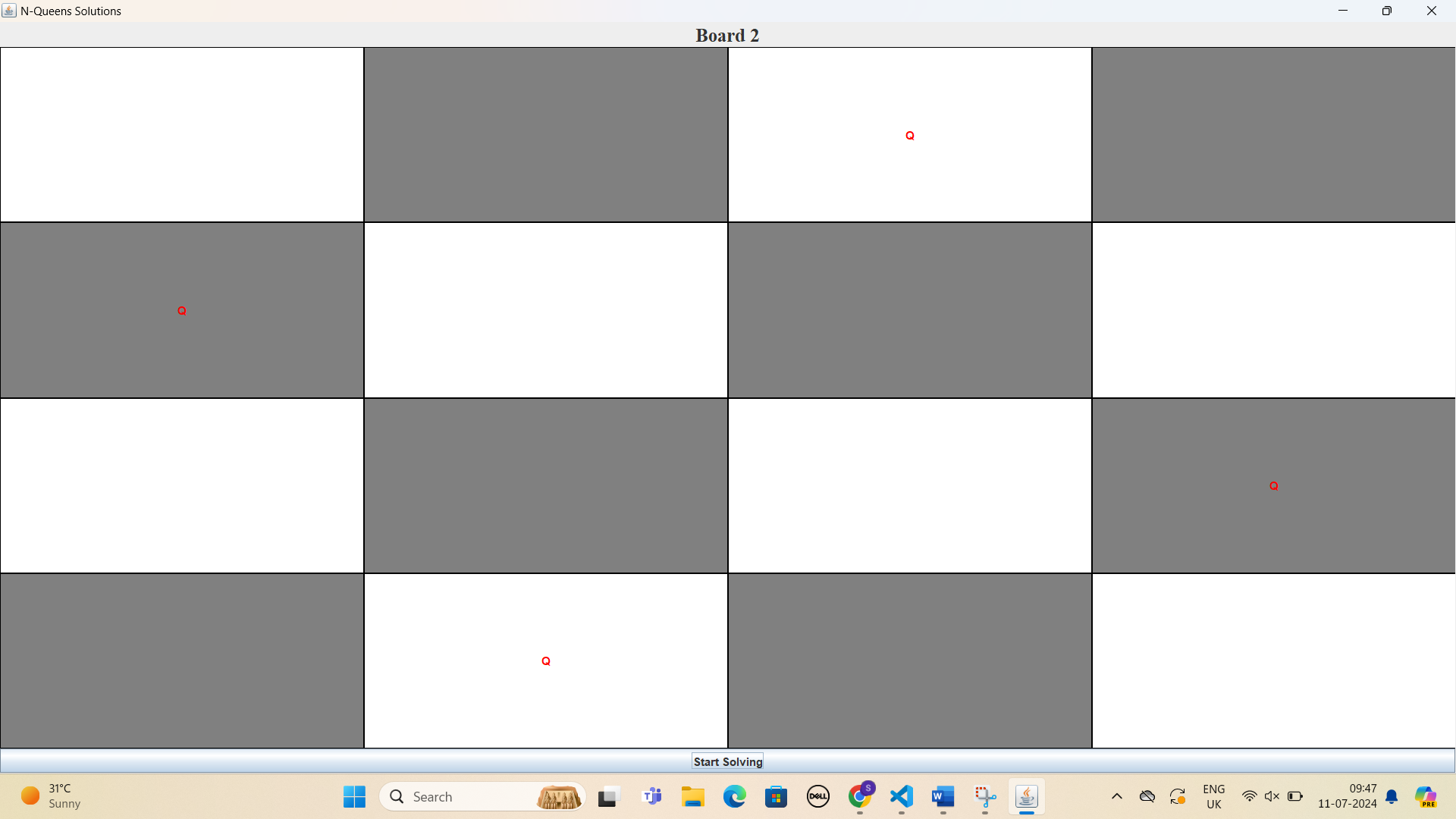
            e.printStackTrace();

        }

    }

}

**RESULT**

****

