

Submitted by Submitted To

Shubham Kumar Mr. Rahul Singh

Reg No : 12223527

**ACKNOWLEDGEMENT**

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

**Introduction**

**The N-Queen problem is a classic computer science and artificial intelligence problem that involves placing N chess queens on an N×N chessboard such that no two queens threaten each other. This project aims to solve the N-Queen problem using a recursive backtracking algorithm and visualize the solutions using a graphical user interface (GUI).**

**Objectives**

* **Algorithm Implementation: Implement a backtracking algorithm to solve the N-Queen problem for a given N.**
* **Solution Representation: Store and display all valid solutions.**
* **Graphical User Interface: Create a GUI to visually represent the solutions.**

**Methodology**

**3.1 Algorithm**

**The N-Queen problem is solved using a recursive backtracking approach. The algorithm attempts to place a queen in each row, moving from the top row to the bottom row. If placing a queen in a particular position does not lead to a solution, the algorithm backtracks and tries the next possible position.**

**Pseudocode for the Backtracking Algorithm:**

1. **Start in the leftmost column**
2. **If all queens are placed, return true**
3. **Try all rows in the current column**
   * **If a queen can be placed safely in the current row, mark this cell and move to the next column recursively.**
   * **If placing the queen in the current row leads to a solution, return true.**
   * **If placing the queen in the current row does not lead to a solution, backtrack and unmark this cell.**
4. **If no row in the current column leads to a solution, return false.**

**3.2 GUI Implementation**

**The solutions are visualized using the Java Swing framework. The GUI includes:**

* **A chessboard where queens are placed.**
* **A button to navigate through different solutions.**
* **Labels to display the current solution number**

**Implementation**

**The implementation consists of several key components, each responsible for different aspects of the solution.**

**1 NQueenStaticBoard Class**

**This is the main class that ties together the algorithm and the GUI components. It includes methods for solving the problem, printing the solutions, and displaying the solutions in a GUI.**

**2 SolveNQueens Method**

**This method initializes the board and calls the recursive function to solve the problem.**

**3 nQueens Method**

**This method is the core of the backtracking algorithm.**

**4 isSafe Method**

**This method checks if a queen can be placed at a given position.**

**5 GUI Methods**

**These methods handle the GUI display of the solutions.**

**displaySolutions sets up the GUI:**

**Results**

**The N-Queen problem solver successfully computes all possible solutions for a given N and displays them both in the console and through a graphical user interface. The project demonstrates the effectiveness of the backtracking algorithm and the usability of Java Swing for creating interactive applications.**

**Conclusion**

**This project highlights the combination of algorithmic problem solving and GUI development. By implementing the N-Queen problem solver, the project not only provides a solution to a classical problem but also enhances user interaction through visual representation.**

**Future enhancements could include optimizing the algorithm for larger values of N and adding more interactive features to the GUI, such as saving solutions or exploring individual placements.**

**Code**

import javax.swing.*\**;

import java.awt.*\**;

import java.util.ArrayList;

import java.util.List;

public class NQueenStaticBoard

{

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

    public static void main(String[] args) {

        int n = 4;

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

        printChessBoard(chessboards);

        displaySolutions(chessboards, 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 printChessBoard(List<List<String>> board) {

        for (int i = 0; i < board.get(0).size() + 4; i++) {

            System.out.print("-");

        }

        System.out.println();

        int j = 1;

        for (List<String> bd : board) {

            System.out.println("Board: " + j++);

            for (String ln : bd) {

                System.out.println("| " + ln + " |");

            }

            for (int i = 0; i < board.get(0).size() + 4; i++) {

                System.out.print("-");

            }

            System.out.println();

        }

        System.out.println("Possible Arrangements : " + board.size());

    }

    public static void displaySolutions(List<List<String>> solutions, int n) {

        SwingUtilities.invokeLater(() -> {

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

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

            frame.setSize(n \* CELL\_SIZE, n \* CELL\_SIZE + 80); *// +80 for window decorations and label*

            frame.setLayout(new BorderLayout());

            JLabel boardLabel = new JLabel("Board 1", SwingConstants.CENTER);

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

            frame.add(boardLabel, BorderLayout.NORTH);

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

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

            frame.add(boardPanel, BorderLayout.CENTER);

            JLabel[][] 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.BLACK);

                    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]);

                }

            }

            JButton nextButton = new JButton("Next Solution");

            frame.add(nextButton, BorderLayout.SOUTH);

            final int[] solutionIndex = {0};

            nextButton.addActionListener(e -> {

                if (solutions.size() > 0) {

                    solutionIndex[0] = (solutionIndex[0] + 1) % solutions.size();

                    displaySolution(labels, solutions.get(solutionIndex[0]), n);

                    boardLabel.setText("Board " + (solutionIndex[0] + 1));

                }

            });

            if (solutions.size() > 0) {

                displaySolution(labels, solutions.get(0), n);

            }

            frame.setVisible(true);

        });

    }

    private static void displaySolution(JLabel[][] labels, List<String> solution, int n) {

        for (int i = 0; i < n; i++) {

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

                if (solution.get(i).charAt(j) == '1') {

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

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

                } else {

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

                }

            }

        }

    }

}

****