**Project: N\_Queen\_Visualizer**



Submitted by Submitted To

Kashish Pratap Rahul Sir

Reg No : 12223339

**ACKNOWLEDGEMENT**

**I would like to extend my heartfelt gratitude to Rahul sir for providing me with the wonderful opportunity to work on the NQueensVisualizer project. This project has not only enhanced my technical skills but also instilled in me a deeper appreciation for the art of software development. I am truly grateful for Rahulsir's guidance and encouragement, which have played a pivotal role in shaping my learning experience.**

***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 NQueenVisualizer project represents a compelling exploration into the sophisticated realms of problem-solving and visual representation. Developed using Java and Swing, it dynamically showcases solutions to the classic N-Queens problem in a meticulously sequential manner. Each chessboard layout is intricately designed, featuring queens symbolized by distinctive black markers set against a carefully curated colour palette, enhancing both clarity and aesthetic appeal. Beyond its practical utility for algorithmic understanding, this project serves as a captivating visual experience, blending elegant design with intuitive user interaction. NQueenVisualizer not only underscores the fusion of creativity and logical reasoning but also exemplifies the beauty of computational thinking and the artistry of graphical representation in contemporary software development. Its interactive nature invites users to explore and appreciate the complexities of combinatorial problem-solving, making it a valuable educational tool and an engaging showcase of technical innovation.

**Code**

**import javax.swing.\*;**

**import java.awt.\*;**

**import java.util.ArrayList;**

**import java.util.List;**

**public class NQueensVisualizer extends JPanel {**

**private int[] queens;**

**private int n;**

**private static final int CELL\_SIZE = 80; // Increased cell size**

**private static final int DELAY = 390; // Delay in milliseconds**

**private List<int[]> solutions = new ArrayList<>();**

**public NQueensVisualizer(int n) {**

**this.n = n;**

**queens = new int[n];**

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

**queens[i] = -1;**

**}**

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

**}**

**public void findAllSolutions() {**

**solve(0);**

**}**

**private boolean solve(int row) {**

**if (row == n) {**

**int[] solution = queens.clone();**

**solutions.add(solution);**

**repaintAndSleep();**

**showSolutionFoundPopup();**

**return true;**

**}**

**boolean found = false;**

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

**if (isSafe(row, col)) {**

**queens[row] = col;**

**repaintAndSleep();**

**if (solve(row + 1)) {**

**found = true;**

**}**

**queens[row] = -1; // Reset the row when backtracking**

**repaintAndSleep();**

**}**

**}**

**return found;**

**}**

**private boolean isSafe(int row, int col) {**

**for (int i = 0; i < row; i++) {**

**if (queens[i] == col || Math.abs(queens[i] - col) == Math.abs(i - row)) {**

**return false;**

**}**

**}**

**return true;**

**}**

**private void repaintAndSleep() {**

**repaint();**

**try {**

**Thread.sleep(DELAY);**

**} catch (InterruptedException e) {**

**Thread.currentThread().interrupt(); // Restore interrupted status**

**e.printStackTrace();**

**}**

**}**

**private void showSolutionFoundPopup() {**

**JOptionPane.showMessageDialog(this, "Solution found!");**

**}**

**private void showTotalSolutions() {**

**JOptionPane.showMessageDialog(this, "Total solutions: " + solutions.size());**

**}**

**@Override**

**protected void paintComponent(Graphics g) {**

**super.paintComponent(g);**

**Color lightColor = new Color(255, 248, 220); // Cream color**

**Color darkColor = new Color(139, 69, 19); // Dark brown color**

**Color queenColor = new Color(220, 220, 220); // Dark white (light gray) for queen**

**Color highlightColor = new Color(255, 215, 0); // Darker yellow for highlighting**

**// Set the thickness of the border**

**int borderThickness = 4;**

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

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

**if ((row + col) % 2 == 0) {**

**g.setColor(lightColor);**

**} else {**

**g.setColor(darkColor);**

**}**

**g.fillRect(col \* CELL\_SIZE, row \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);**

**if (queens[row] == col) {**

**// Draw a border around the square**

**g.setColor(highlightColor);**

**((Graphics2D) g).setStroke(new BasicStroke(borderThickness));**

**g.drawRect(col \* CELL\_SIZE + borderThickness / 2, row \* CELL\_SIZE + borderThickness / 2,**

**CELL\_SIZE - borderThickness, CELL\_SIZE - borderThickness);**

**g.setColor(queenColor);**

**g.setFont(new Font("SansSerif", Font.BOLD, CELL\_SIZE / 2));**

**FontMetrics fm = g.getFontMetrics();**

**String queenSymbol = "♛";**

**int x = col \* CELL\_SIZE + (CELL\_SIZE - fm.stringWidth(queenSymbol)) / 2;**

**int y = row \* CELL\_SIZE + (CELL\_SIZE + fm.getAscent()) / 2;**

**g.drawString(queenSymbol, x, y);**

**}**

**}**

**}**

**}**

**public static void main(String[] args) {**

**SwingUtilities.invokeLater(() -> {**

**String input = JOptionPane.showInputDialog("Enter the size of the board (n):");**

**int n;**

**try {**

**n = Integer.parseInt(input);**

**if (n <= 0) throw new NumberFormatException();**

**} catch (NumberFormatException e) {**

**JOptionPane.showMessageDialog(null, "Invalid input. Please enter a positive integer.");**

**return;**

**}**

**NQueensVisualizer visualizer = new NQueensVisualizer(n);**

**JFrame frame = new JFrame("N-Queens Solver");**

**frame.add(visualizer);**

**frame.pack();**

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

**frame.setLocationRelativeTo(null);**

**frame.setVisible(true);**

**new Thread(() -> {**

**visualizer.findAllSolutions();**

**visualizer.showTotalSolutions();**

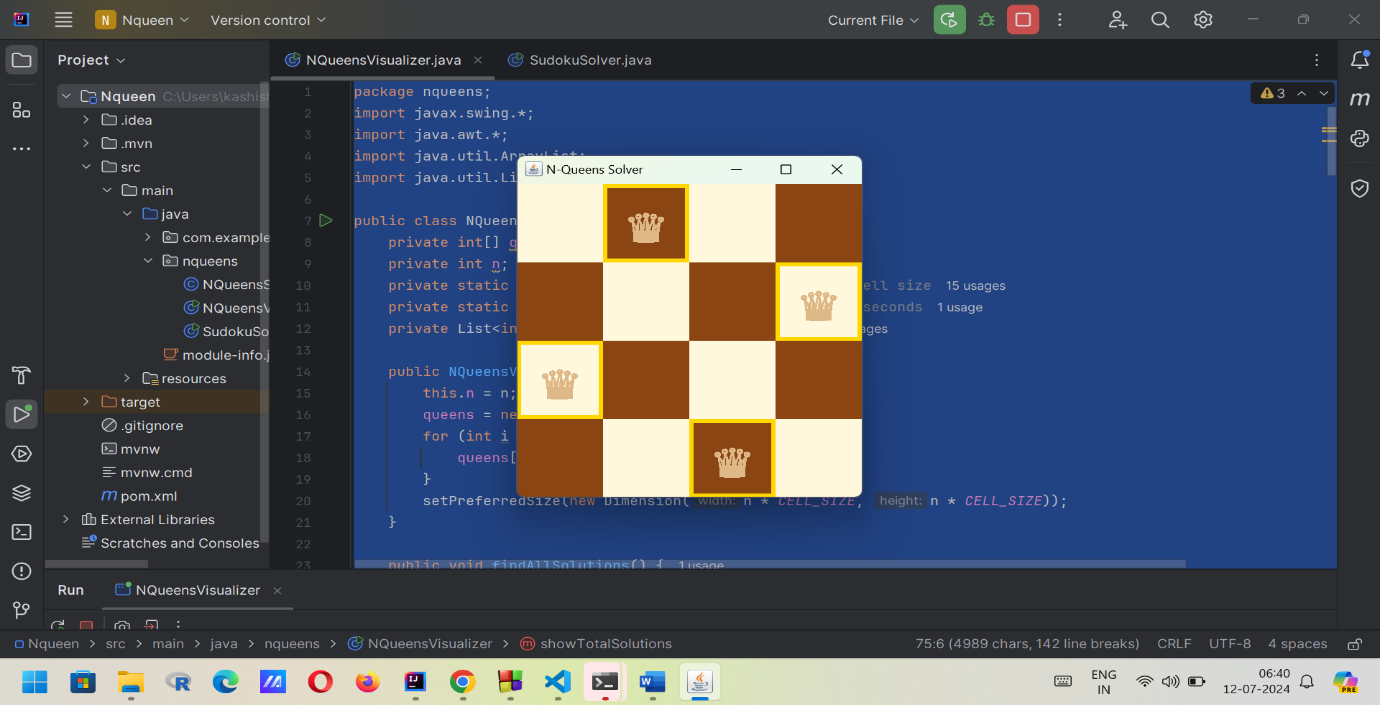
**}).start();**

**});**

**}**

**}**

**Output:**

****

****