



# Project Report

## N-Queens Visualizer in C++

Reg No. 12217380

Submitted By : Ashif Ahamad

Submitted To : Rahul Sir

# Table of Contents

1. Introduction
2. Project Objectives
3. Literature Review
4. System Design
5. Implementation
6. Testing
7. Conclusion

8. Future Work

9. References

# Introduction

## Overview

The N-Queens problem is a well-known combinatorial problem in which the objective is to place  $N$  queens on an  $N \times N$  chessboard such that no two queens can attack each other. This project aims to solve the N-Queens problem and visualize the solution in a console-based application using C++.

## Project Objectives

- Implement an efficient algorithm to solve the N-Queens problem.
- Develop a console-based application to visualize the placement of queens on the board.

- Enhance the visual presentation using colors and formatted console output.
- Ensure the application is user-friendly and interactive.

# Literature Review

## N-Queens Problem

The N-Queens problem has been extensively studied in computer science and mathematics. It is an example of a constraint satisfaction problem and can be solved using various approaches such as backtracking, heuristic search, and genetic algorithms.

## Visualization Techniques

Console-based applications can utilize various techniques to enhance visual presentation, including ASCII art, ANSI escape codes for color, and formatted output.

# System Design

## Architecture

The system consists of the following components:

- **Console Initialization:** Sets up the console environment for UTF-8 encoding and determines the console width.
- **Visualizer Initialization:** Displays a welcome message and prompts the user for input.
- **Solver:** Implements the backtracking algorithm to solve the N-Queens problem.
- **Visualizer:** Displays the board with the queens placed according to the solution

# Implementation

## Console Initialization

The *consoleInit* function sets the console to UTF-8 encoding, determines the width of the console for proper formatting and sets the cursor visibility to false.

```
3
4 void consoleInit(void){
5     SetConsoleOutputCP(CP_UTF8);
6     CONSOLE_SCREEN_BUFFER_INFO csbi;
7     GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
8     consoleWidth = csbi.srWindow.Right - csbi.srWindow.Left + 1;
9
10    HANDLE consoleHandle = GetStdHandle(STD_OUTPUT_HANDLE);
11    CONSOLE_CURSOR_INFO cursorInfo;
12    GetConsoleCursorInfo(consoleHandle, &cursorInfo);
13    cursorInfo.bVisible = FALSE; // Set the cursor visibility to false
14    SetConsoleCursorInfo(consoleHandle, &cursorInfo);
15 }
16
```

## Visualizer Initialization

The *visualizerInit* function displays a welcome message and prompts the user to enter the value of N.

```
16
17 void visualizerInit(void){
18     for(int i = 0; i<consoleWidth; i++){
19         cout << "=";
20     }
21     string welcomeText = "WELCOME TO N-queen Visualizer 🍷";
22     cout << "\n\n";
23     cout << changeColor(onCenter(welcomeText, consoleWidth), 'Y') << endl;
24     cout << changeColor(onCenter("↓ Enter value of N to get Started ↓", consoleWidth), 'B') << endl;
25
26     cout << onCenter("", consoleWidth);
27     cin >> N;
28 }
29
```

## Solver

The solver uses a backtracking algorithm to find a solution to the N-Queens problem.

```
29
30 bool isSafe(const vector<vector<char>>& board, int row, int col) {
31     for (int i = 0; i < row; i++)
32         if (board[i][col] == 'Q') return false;
33     for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
34         if (board[i][j] == 'Q') return false;
35     for (int i = row, j = col; i >= 0 && j < N; i--, j++)
36         if (board[i][j] == 'Q') return false;
37     return true;
38 }
39
```



```

40 void solveNQueensUtil(vector<vector<char>>& board, int row) {
41     if (row >= N){
42         updateBoard(N,0,"",false);
43         cout << changeBg(to_string(solNum++) + addSuffix(solNum) + " Solution Found", 'G') << "\n\n";
44         Sleep(delay);
45         clearBoard();
46         drawBoard(board);
47         Sleep(delay);
48         return;
49     }
50
51     for (int col = 0; col < N; col++) {
52         if (isSafe(board, row, col)) {
53             board[row][col] = 'Q';
54             updateBoard(row, col, " Q", false);
55             Sleep(delay);
56
57             solveNQueensUtil(board, row + 1);
58
59             board[row][col] = '.';
60             updateBoard(row, col, " ", false);
61         } else {
62             board[row][col] = 'Q';
63             updateBoard(row, col, " Q", true);
64             Sleep(delay);
65
66             board[row][col] = '.';
67             updateBoard(row, col, " ", false);
68         }
69     }
70     return;
71 }

```

```

72
73 void solveNQueens() {
74     if(N<4){
75         cout << changeColor(onCenter("Solution Doesn't exist!!!", consoleWidth),'R');
76         return;
77     }
78     vector<vector<char>> board(N, vector<char>(N, '.'));
79     clearBoard();
80     drawEmptyBoard();
81     solveNQueensUtil(board, 0);
82     updateBoard(N,0,"",false);
83     cout << changeBg("No More Solution Found!!!", 'R') << "\n\n";
84 }
85

```

# Testing

## Test Cases

- **Test Case 1:**  $N = 4$  ○

Input: 4

- Expected Output: A valid 4x4 board with 4 queens placed such that no two queens attack each other.

- **Test Case 2:**  $N = 8$  ○

Input: 8

- Expected Output: A valid 8x8 board with 8 queens placed such that no two queens attack each other.

## Results

All test cases passed successfully, with the board displaying the queens in the correct positions and with the expected formatting and colors.

# Conclusion

The N-Queens Visualizer in C++ project successfully meets its objectives of solving the N-Queens problem and providing an enhanced console-based visualization. The project demonstrates a strong grasp of algorithmic problem-solving through the implementation of a backtracking algorithm, which efficiently finds solutions for various board sizes. Technically, the project leverages C++ capabilities and Windows API functions to manage console properties and UTF-8 encoding, ensuring broad compatibility, and improving the visual output. The implementation of ANSI escape codes for color output adds a layer of clarity, making the visualization more user-friendly. Additionally, functions like *onCenter* and *changeCoLoR* enhance the user interface, creating a polished and professional application.

