Project Report

N-Queens Visualizer in C++

By Prateek Bhardwaj

Table of Contents

Introduction

Overview

The N-Queens problem is a well-known combinatorial problem in which the objective is to place N queens on an N×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.

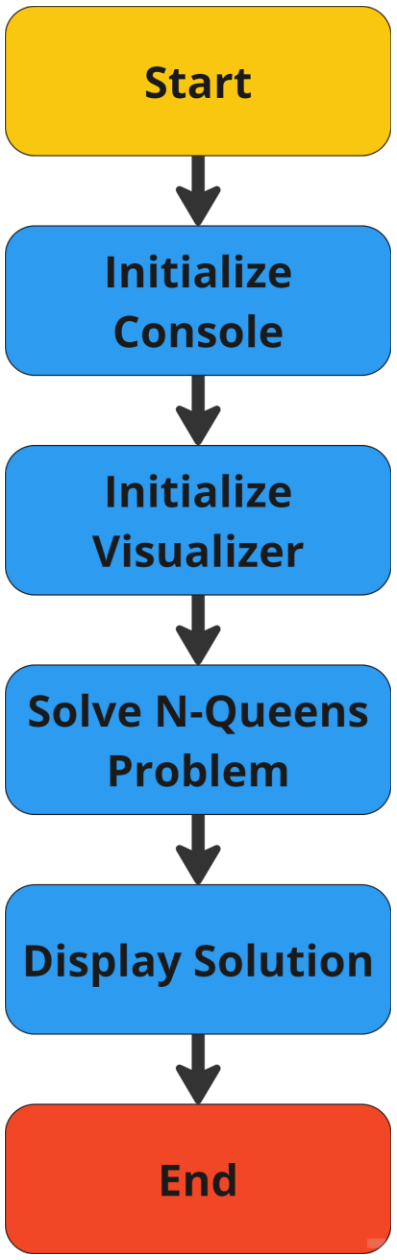
System Design

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

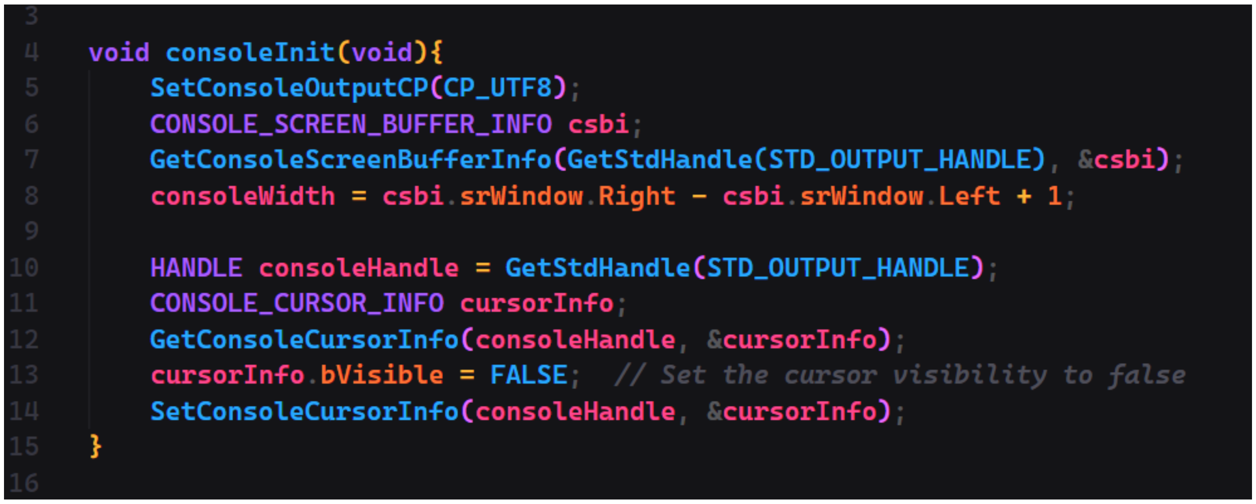
Flowchart


Start
Initialize Console
Initialize Visualizer
Solve N-Queens Problem
Display Solution
End

Implementation

Console Initialization

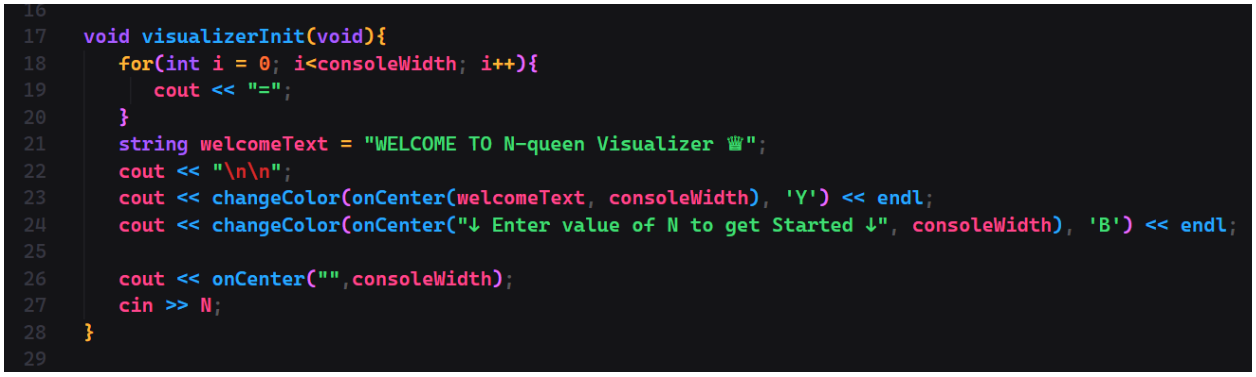
The *consoleInit* function sets the console to UTF-8 encoding, determines the width of the console for proper formatting and sets the cursor visibility to false.



```c
void consoleInit(void){
    SetConsoleOutputCP(CP_UTF8);
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE), &csbi);
    consoleWidth = csbi.srWindow.Right - csbi.srWindow.Left + 1;

    HANDLE consoleHandle = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_CURSOR_INFO cursorInfo;
    GetConsoleCursorInfo(consoleHandle, &cursorInfo);
    cursorInfo.bVisible = FALSE;  // Set the cursor visibility to false
    SetConsoleCursorInfo(consoleHandle, &cursorInfo);
}
```

Visualizer Initialization

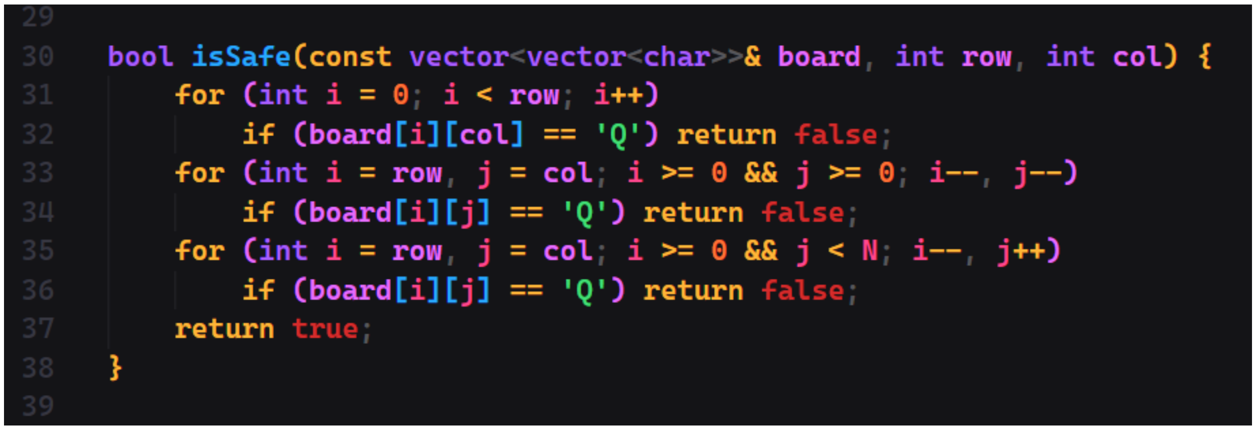
The *visualizerInit* function displays a welcome message and prompts the user to enter the value of N.



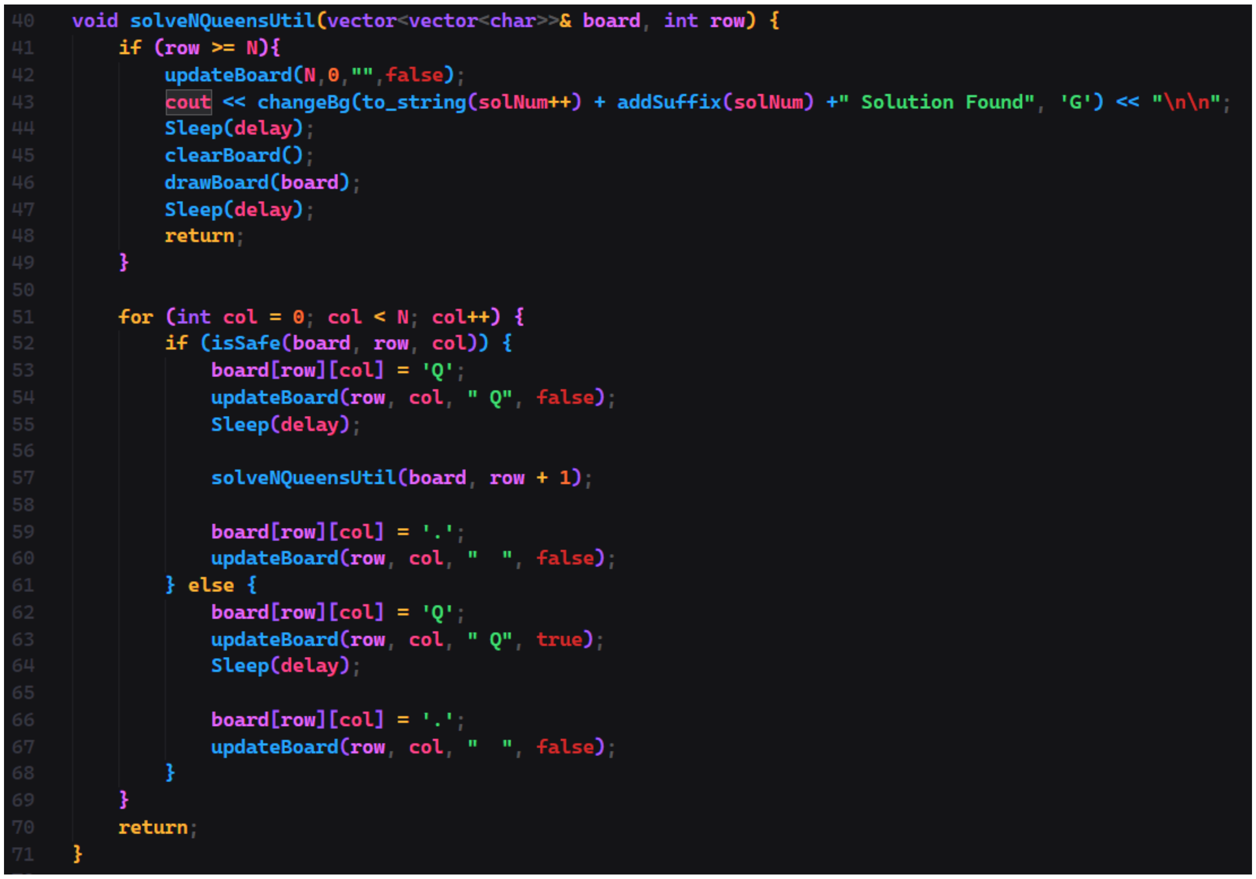
```cpp
void visualizerInit(void){
    for(int i = 0; i<consoleWidth; i++){
        cout << "=";
    }
    string welcomeText = "WELCOME TO N-queen Visualizer ♛";
    cout << "\n\n";
    cout << changeColor(onCenter(welcomeText, consoleWidth), 'Y') << endl;
    cout << changeColor(onCenter("↓ Enter value of N to get Started ↓", consoleWidth), 'B') << endl;

    cout << onCenter("",consoleWidth);
    cin >> N;
}
```

Solver

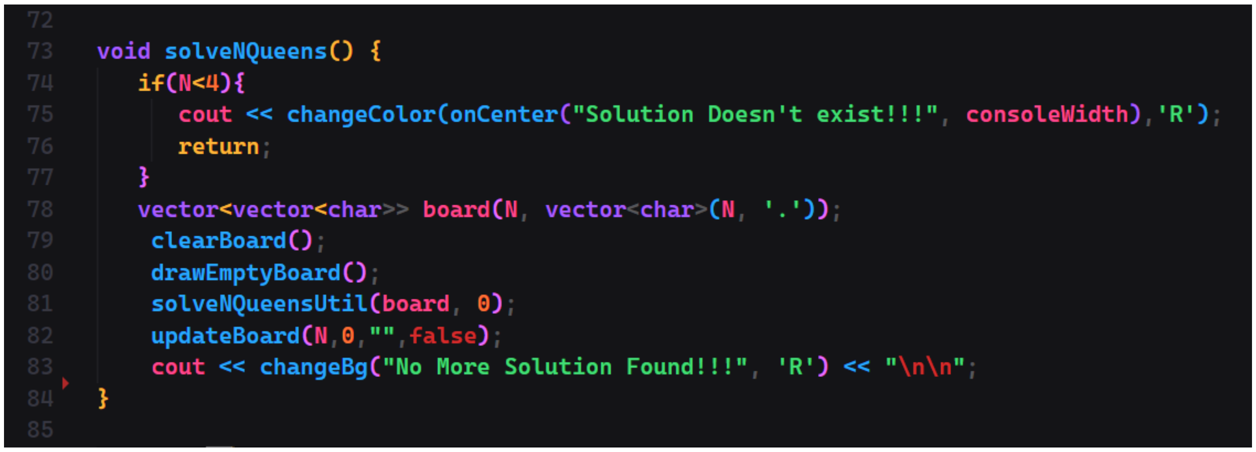
The solver uses a backtracking algorithm to find a solution to the N-Queens problem.



```cpp
bool isSafe(const vector<vector<char>>& board, int row, int col) {
    for (int i = 0; i < row; i++)
        if (board[i][col] == 'Q') return false;
    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j] == 'Q') return false;
    for (int i = row, j = col; i >= 0 && j < N; i--, j++)
        if (board[i][j] == 'Q') return false;
    return true;
}
```



```cpp
void solveNQueensUtil(vector<vector<char>>& board, int row) {
    if (row >= N){
        updateBoard(N,0,"",false);
        cout << changeBg(to_string(solNum++) + addSuffix(solNum) +" Solution Found", 'G') << "\n\n";
        Sleep(delay);
        clearBoard();
        drawBoard(board);
        Sleep(delay);
        return;
    }

    for (int col = 0; col < N; col++) {
        if (isSafe(board, row, col)) {
            board[row][col] = 'Q';
            updateBoard(row, col, " Q", false);
            Sleep(delay);

            solveNQueensUtil(board, row + 1);

            board[row][col] = '.';
            updateBoard(row, col, "  ", false);
        } else {
            board[row][col] = 'Q';
            updateBoard(row, col, " Q", true);
            Sleep(delay);

            board[row][col] = '.';
            updateBoard(row, col, "  ", false);
        }
    }
    return;
}
```



```cpp
void solveNQueens() {
    if(N<4){
        cout << changeColor(onCenter("Solution Doesn't exist!!!", consoleWidth),'R');
        return;
    }
    vector<vector<char>> board(N, vector<char>(N, '.'));
     clearBoard();
     drawEmptyBoard();
     solveNQueensUtil(board, 0);
     updateBoard(N,0,"",false);
     cout << changeBg("No More Solution Found!!!", 'R') << "\n\n";
}
```

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.

Future Work

- Develop a graphical user interface (GUI) for better visualization.
- Explore more efficient algorithms for solving larger instances of the N-Queens problem.
- Add features to allow users to step through the solution process interactively.

References

- N-Queens Problem - Wikipedia
- Backtracking - GeeksforGeeks
- ANSI Escape Codes - Wikipedia