LOVELY PROFESSIONAL UNIVERSITY

N Queen Visualizer

Terminal based visualizer in C++

Name: Rishav Raj Singh

Reg. no: 12223287

Faculty: Rahul Singh

https://github.com/rishavd3v/NQueen_Visualizer

1. Introduction

The N Queen problem is a classic algorithmic challenge that involves placing N chess queens on an N×N chessboard so that no two queens threaten each other. This means no two queens share the same row, column, or diagonal. The problem has significant importance in the fields of computer science and artificial intelligence, particularly in the context of constraint satisfaction problems.

The "N Queen Visualizer" project aims to solve this problem using C++ and provide a visual representation of the solution process. The visualization helps in understanding the backtracking algorithm, which is often used to solve this problem.

2. Problem Statement

The N Queen problem is defined as placing N queens on an N×N chessboard in such a way that no two queens can attack each other. The constraints are:

1. No two queens share the same row.
2. No two queens share the same column.
3. No two queens share the same diagonal.

3. Project Objectives

The primary objectives of the N Queen Visualizer project are:

1. Implement the N Queen problem solution using a backtracking algorithm in C++.

2. Visualize the process of placing and removing queens on the board.
3. Provide a clear and interactive way to understand the backtracking algorithm.
4. Enable users to input different values of N and see the corresponding solutions.

4. Methodology

The project uses a backtracking algorithm to solve the N Queen problem. Backtracking is a depth-first search approach that incrementally builds candidates to the solutions and abandons each partial candidate as soon as it determines that this candidate cannot possibly lead to a valid solution.

The visualization is implemented using C++, with the console being used to display the board state. The methodology involves:

1. Initializing the chessboard.
2. Recursively attempting to place queens on the board.
3. Backtracking when a conflict is detected.
4. Displaying each step of the algorithm visually.

5. Implementation

5.1 Environment Setup

To implement the N Queen Visualizer, the following environment setup is used:

- **C++ Compiler**: GCC
- **Integrated Development Environment (IDE)**: Visual Studio
- **Operating System**: Windows

5.2 Core Logic

The core logic revolves around the backtracking algorithm. The algorithm can be summarized as follows:

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 this row, mark this cell and move to the next column.
    - If placing the queen in the current cell leads to a solution, return true.
    - If placing the queen doesn't lead to a solution, unmark this cell and backtrack.
4. If no row works, return false.

Here is a snippet of the core logic in C++:

```
#include <bits/stdc++.h>
#include <windows.h>
using namespace std;

void printboard(vector<string> &board) {
    system("cls"); // Clear the console
    for (int i = 0; i < board.size(); i++)
{
        for (int j = 0; j <
board[i].size(); j++) {
            cout << board[i][j] << " ";
        }
        cout << "\n";
    }
    cout << "\n";
```

```
    Sleep(500); // Wait for 500
milliseconds to visualize the backtracking
}

void backtrack(int r, int n,
unordered_set<int>& col,
unordered_set<int>& posdiag,
unordered_set<int>& negdiag,
vector<string>& board,
vector<vector<string>>& res) {
    if (r == n) {
        res.push_back(board);
        return;
    }

    for (int c = 0; c < n; ++c) {
        if (col.count(c) || posdiag.count(r
+ c) || negdiag.count(r - c)) {
            continue;
        }

        col.insert(c);
        posdiag.insert(r + c);
        negdiag.insert(r - c);
        board[r][c] = 'Q';
        printboard(board); // Print the
current board state
        backtrack(r + 1, n, col, posdiag,
negdiag, board, res);
        col.erase(c);
        posdiag.erase(r + c);
        negdiag.erase(r - c);
        board[r][c] = '.';
```

```
            printboard(board); // Print the
board state after backtracking
        }
    }

    vector<vector<string>> solveNQueens(int n)
    {
        unordered_set<int> col;
        unordered_set<int> posdiag; // (r + c)
        unordered_set<int> negdiag; // (r - c)

        vector<vector<string>> res;
        vector<string> board(n, string(n,
    '.'));
        printboard(board); // Print the initial
    empty board
        backtrack(0, n, col, posdiag, negdiag,
    board, res);
        return res;
    }
```

5.3 Visualization

To visualize the board, the `printboard` method is used. This method clears the console and prints the board's current state. This simulates the process of placing and removing queens, aiding in the visualization of the backtracking algorithm.

```
void printboard(vector<string> &board) {
    system("cls"); // Clear the console
    for (int i = 0; i < board.size(); i++)
{
        for (int j = 0; j <
board[i].size(); j++) {
```

```
            cout << board[i][j] << " ";
        }
        cout << "\n";
    }
    cout << "\n";
    Sleep(500); // Wait for 500
milliseconds to visualize the backtracking
}
```

6. Results

The N Queen Visualizer successfully implements the N Queen problem solution using backtracking. It provides a visual representation of each step in the solution process, making it easier to understand how the algorithm works. Users can input different values of N and observe the board's state as the algorithm progresses.

7. Challenges and Solutions

Challenge 1: Clearing the Console

- **Solution**: Using the `system("cls")` command to clear the console in Windows environments.

Challenge 2: Visualizing the Process

- **Solution**: The `printboard` method includes a delay using `Sleep(500)` to allow users to see each step of the algorithm.

8. Future Enhancements

Future enhancements to the N Queen Visualizer may include:

1. **Graphical User Interface (GUI)**: Implementing a GUI using libraries such as Qt or SFML for a more interactive visualization.
2. **Performance Optimization**: Optimizing the backtracking algorithm to handle larger values of N more efficiently.
3. **Additional Features**: Allowing users to pause, resume, or step through the visualization.

9. Conclusion

The N Queen Visualizer project demonstrates an effective way to solve and visualize the N Queen problem using C++. The project provides an interactive and educational tool for understanding the backtracking algorithm. Future enhancements could further improve the user experience and performance.

10. References

1. LeetCode. N-Queens Problem. Available at: https://leetcode.com/problems/n-queens/
2. GeeksforGeeks. Backtracking | Set 3 (N Queen Problem). Available at: https://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/