Lovely Professional University

Summer-pep project

Name: - Anuj Verma

Reg: - 12218174
Batch: - SKO2-31

Implementing and visualizing the n-Queen Problem in Java using Swing

Table of Contents

1. Introduction

The n-Queen problem is a classic combinatorial problem that involves placing n queens on an n x n chessboard such that no two queens threaten each other. This means that no two queens share the same row, column, or diagonal. This problem is a common example used to illustrate backtracking algorithms in computer science.

In this report, we present a solution to the n-Queen problem implemented in Java. Additionally, we visualize the solution using the Swing library to provide a graphical representation of the board and the queens' placements.

2. Problem Statement

The n-Queen problem can be defined as follows:

- Given an n x n chessboard, place n queens on the board so that no two queens threaten each other.
- A queen can attack another queen if it is in the same row, column, or diagonal.
- The task is to find all possible ways to place the queens on the board or find one valid solution.

The challenge is to devise an algorithm that can efficiently find a solution to this problem for any given n.

3. Algorithm

The algorithm used to solve the n-Queen problem is a classic backtracking algorithm. Backtracking is a general algorithm for finding all (or some) solutions to computational problems, notably constraint satisfaction problems.

Steps of the 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.
    - If placing the queen in the current row leads to a solution, return true.
    - If placing the queen does not lead to a solution, unmark this cell and backtrack to the previous column.

4. If all rows have been tried and none worked, return false to trigger backtracking.

Pseudocode:

```
function solveNQueens(board, col):
    if col >= N:
        return true
    for each row in board:
        if isSafe(board, row, col):
            placeQueen(board, row, col)
            if solveNQueens(board, col + 1):
                return true
            removeQueen(board, row, col)
    return false
```

isSafe Function:

The isSafe function checks if a queen can be placed on the board at a given row and column without being attacked.

4. Implementation

The implementation is done in Java using the Swing library for visualization. The key components of the implementation are:

Setup and Initialization

The main class, NQueenVisualizer, extends JFrame to create a window for the visualizer. The board size (n) and an array (queens) to store the position of queens are initialized. The queen image is loaded using ImageIO.

Visualization with Swing

The paint method is overridden to draw the chessboard and queens. The board is painted using a two-tone color scheme, and the queens are represented by an image loaded earlier.

Enhancements and Improvements

To improve the visualizer, several enhancements were made:

- **Anti-Aliased Rendering**: Enabled smoother graphics.

- **Color Scheme**: Used colors that resemble a real chessboard.
- **Queen Representation**: Improved visual representation using images.

Further improvements could include:

- **Dynamic Board Size**: Allow users to input the board size.
- **Animation Speed Control**: Provide a way to adjust the delay between steps.
- **Multiple Solutions**: Visualize all possible solutions, not just one.

5. Results

The implemented visualizer successfully displays the process of solving the n-Queen problem. The queens are placed on the board iteratively, and the backtracking process is visually represented. The enhancements made to the visual representation improve the overall user experience.

6. Conclusion

The n-Queen visualizer provides an interactive way to understand the backtracking algorithm used to solve the n-Queen problem. By using Java and Swing, we created a graphical representation that helps in visualizing the placement and backtracking steps. The enhancements made to the visual representation make the visualizer more appealing and easier to understand.

THANK YOU