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 requires placing n queens on an n× n\times n×n chessboard such that no two queens can attack each other. This implies that:

1. No two queens can be in the same row.
2. No two queens can be in the same column.
3. No two queens can be on the same diagonal.

The objective is to find all possible arrangements of n queens on the chessboard that satisfy these constraints. The problem serves as a quintessential example for understanding and implementing backtracking algorithms in computer science.

In this report, we aim to:

1. Develop a Java-based solution to the n-Queen problem.
2. Visualize the solution using the Swing library to provide a graphical representation of the chessboard and the placement of the queens.

3. Algorithm

To solve the n-Queen problem, we employ a backtracking algorithm, which systematically searches for solutions by exploring potential placements of queens on the chessboard and retracting invalid placements. The algorithm can be broken down into the following steps:

1. **Initialize the Board:**
   - Create an n×n chessboard, initially empty.
2. **Place Queens Recursively:**
   - Start placing queens row by row, beginning from the first row.
   - For each row, attempt to place a queen in each column, checking if it is safe to place the queen.
3. **Check Safety:**
   - A position is deemed safe if there are no other queens in the same column, and there are no other queens on the major and minor diagonals.

4. **Backtracking:**
   - If placing a queen in a particular column of a row leads to a solution, proceed to place the next queen in the subsequent row.
   - If no safe position is found in a row, backtrack by removing the queen placed in the previous

row and attempt to place it in the next valid column.

5. **Store Solutions:**
    - Once all queens are placed successfully on the board such that they do not threaten each other, store the current configuration of the board as a solution.
6. **Repeat:**
    - Continue the process until all possible configurations have been explored.

Pseudocode:

```
function solveNQueens(n):
   solutions = []
   board = createEmptyBoard(n)
   placeQueen(board, 0, solutions)
   return solutions


function placeQueen(board, row, solutions):
   if row == size(board):
      solutions.append(copy(board))
      return
```

```
  for col in range(size(board)):
    if isSafe(board, row, col):
      board[row][col] = 'Q'
      placeQueen(board, row + 1, solutions)
      board[row][col] = '.'  // backtrack




function isSafe(board, row, col):
  for i in range(row):
    if board[i][col] == 'Q':
      return false
  for i, j in diagonals(board, row, col):
    if board[i][j] == 'Q':
      return false
  return true


function diagonals(board, row, col):
  // returns coordinates of all squares on the major and minor diagonals
  // for the given (row, col)
  // ...

function createEmptyBoard(n):
  // returns an n x n board initialized with '.'
  // ...
```

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