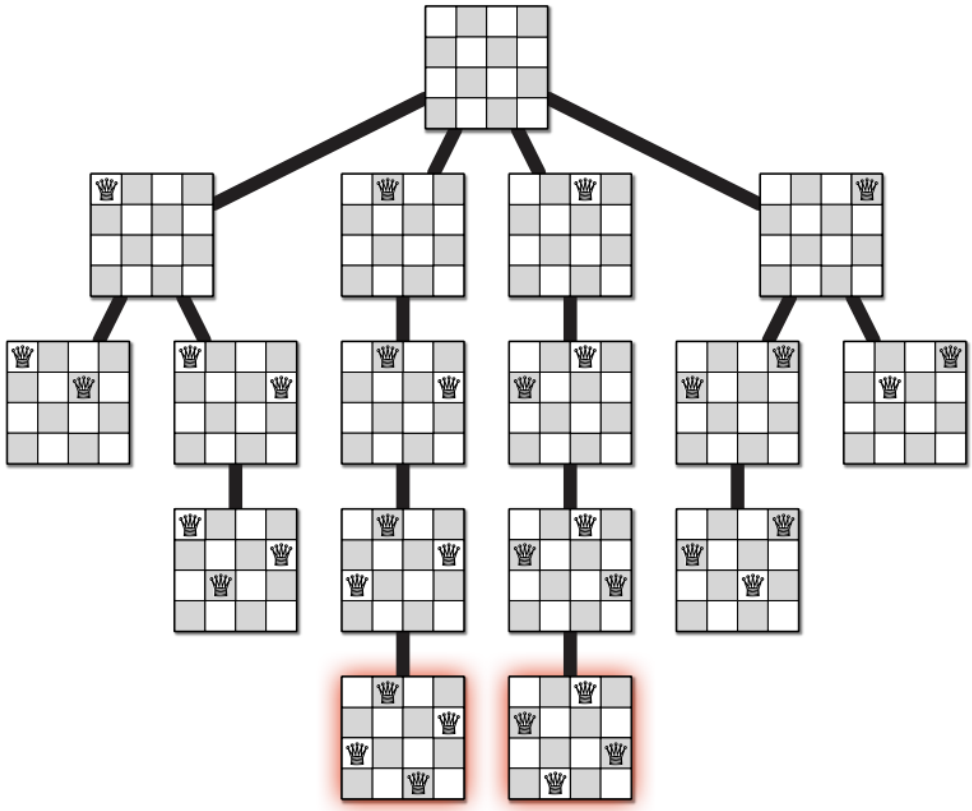
N-Queen problem solver (Demo is the last item on the page)

Solving N-Queens provlem using BackTracking Algorithm with Multiple Threading using Java



Problem Formulation

The N-queens problem involves placing 'n' chess queens on an n×n chessboard without threats. Solutions exist for all natural numbers 'n' except n=2 and n=3. A complete-state formulation is used, starting with all n queens and moving them to reach the goal state.

Algorithm

The backtracking algorithm for the N-Queens problem uses a recursive function to place queens on the board, backtracking if a position conflicts, and exploring alternative solutions.

Speed Enhancement

The BackTracking Algorithm can be improved by breaking down problems into smaller ones (each starting with a different column state), with each problem running on a separate thread, and the first solution stopping all others.

Tech Stack

Client: JavaSwing

Server: Java With Ant

Program Structure:

Program consists of 4 Classes:*

- NQueen
- GamePanel
- GameWindow
- Game

1) NQueen

NQueen class, designed to solve N-Queens problem using multithreading in a GUI with Java Swing, Class is well-structured and follows good practices. It emphasizes thread safety, using synchronization for shared variables and lock objects.

Contians 4 Methodes:

- findSol
- placeQueenOrNot
- setShouldExit
- run

  1- findSol(): recursive function used in solving the N-Queens problem. Let's break down its functionality

  - **Exit Condition**

    ```
    if (shouldExit == true) { throw new Exception("stop"); }
    ```

    This mechanism is a way to signal the threads that still running the backtracking to stop its execution.

  - **Base Case**

    ```
    if (col >= nQueens) { return true; }
    ```

    The method returns true if the current column (col) is equal to or greater than the total number of queens, indicating successful placement and solution.

  - **Recursive Exploration**

    ```
    for (int row = 1; row < nQueens; row++)
    ```

    The method checks if a queen can be placed at a specific position in a row, sets the icon, sleeps, and calls itself for the next column, returns true if successful, and clears the icon if not.

- **Backtracking**

  the method backtracks by returning false. This triggers the removal of the queen icon from the current cell .

  ```
  boardCells[row][col].setIcon(null)
  ```

2- placeQueenOrNot(): method ensures the safe placement of a queen on a specific cell in the boardCells array, detecting conflicts in the same column and diagonals.

- **Check Same Column**

  ```
  for (int i = 0; i < col; i++)
  ```

  The method iterates through elements in the same row from the leftmost column, checking if a non-null icon (queen) is present, returns false.

- **Check Left Top Diagonal**

  ```
  for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
  ```

  The method iterates upward and to the left, checking the left-top diagonal for a queen, and returns false if it finds one.

- **Check Left Bottom Diagonal**

  ```
  for (int i = row, j = col; j >= 0 && i < nQueens; i++, j--)
  ```

  The method iterates downward and to the left of the diagonal, checking for a queen, and returns false if it finds one.

- **Check Right Top Diagonal**

  ```
  for (int i = row, j = col; i >= 0 && j < nQueens; i--, j++)
  ```

  The method iterates upward and to the right, checking the right-top diagonal for a queen, and returns false if it finds one.

- **Check Left Bottom Diagonal**

  ```
  for (int i = row, j = col; j >= 0 && i < nQueens; i++, j--)
  ```

  The method iterates downward and to the left of the diagonal, checking for a queen, and returns false if it finds one.

- **Check Right Bottom Diagonal**

  ```
  for (int i = row, j = col; i < nQueens && j < nQueens; i++, j++)
  ```

  The method iterates downward and to the right, checking the right-bottom diagonal for a queen, and returns false if it finds one.

- **Return True if Safe**

If none of the above checks detect a queen in a conflicting position, the method returns true, indicating that it is safe to place a queen in the specified cell.

3- setShouldExit(): method is utilized to signal threads to stop their execution in the N-Queens problem.

- **Synchronization**

  The method employs the synchronized keyword to execute code atomically, ensuring data consistency when dealing with shared resources accessed by multiple threads.

- **Setting the shouldExit Flag**

  ```
  NQueen.shouldExit = shouldExit;
  ```

  The NQueen.shouldExit function sets the boolean value to exit, which is likely used by threads to determine whether to continue execution or stop.

- **Highlighting Solution on the Chessboard**

  The method iterates over chessboard cells, changing the background color of queen-colored cells to green, likely to visually indicate their positions in the solution.

4- run(): method is utilized to signal threads to stop their execution in the N-Queens problem.

- **Attempt to Find Solution**

  The findSol method is called to find a solution to the N-Queens problem, and if the result is false, the thread is terminated.

- **Signal Threads to Stop**

  ```
  NQueen.shouldExit = shouldExit;
  ```

  The NQueen.shouldExit function sets the boolean value to exit, which is likely used by threads to determine whether to continue execution or stop.

- **Highlighting Solution on the Chessboard**

  ```
  setShouldExit(true, currentThread().getName())
  ```

  If findSol returns true, then signal all threads should exit, prints a message indicating a solution, and changes queen cell background color to green.

2) GamePanel

The GamePanel class extends JPanel, is designed to represent the chessboard for the N-Queens problem, with key components and functionalities.

Contians 2 Methodes:

- setPanelSize
- setBoardCells

  1- setPanelSize(): This method used to set and enforce size constraints on a Swing component, in this case, a JPanel, ensuring it adheres to the specified width and height limits.

  2- setBoardCells(): The setBoardCells method initializes and configures JLabel components representing chessboard cells in the GamePanel. nested loops to iterate through each row and column of the chessboard, give it jpanal and background color.

  - **Setting Background Colors**

    ```
    If (i + j) % 2 == 1
    ```

    If True the cell is given a black background Otherwise, the cell is given a white background.

3) GameWindow

This class encapsulates the chessboard window by setting up a JFrame with specific configurations, adding the GamePanel, and making the window visible.

Contians 0 Methodes:

- Usage

  When creating a GameWindow instance, you specify a GamePanel and a thread name, which are then used to display the chessboard in the window.

4) Game

The Game class is a coordinating class that combines components related to the N-Queens problem and its graphical representation, with key components and functionalities.

Contians 2 Methodes:

- runProblemThread
- disposeWindow

  1- runProblemThread():

  ```
  public void runProblemThread()
  ```

  This method initiates the solving thread (nQueen) to find a solution to the N-Queens problem.

  2- disposeWindow():

  ```
  public void disposeWindow()
  ```

The dispose method on the gameWindow is used to close the graphical interface when it's no longer needed.

How To Use

- ***First Method:*** if you have netbeans you can clone it and run in there.
- ***Second Method:*** if you have jdk installed open cli, change current directory into the dist directory then run:

```
java -jar NQueen.jar
```

Demo:

Demo Click Here

Contributors

- Mohamed Samir Mohamed
- Mohamed Shaban Mohamed
- Mohamed Khalid Hassan
- Mahmoud Walid Mahmoud
- Youssef Mostafa Ahmed