OS2 N Queen Problem documentation

project description:

solving N queen problem using multithreading

what we actually did:

2 classes **N_Qeeens_solver** that begins by creating an instance of the **InputScreen** class to collect user input for the size of the chessboard (**N**). Once the user provides the input, the program initiates a set of threads, each responsible for solving the N-Queens problem starting with a queen placed in a different column of the first row.

The program creates an array of threads (**threadsArray**) and launches a thread for each initial column placement. Each thread is an instance of the **ThreadSolver** class, which is responsible for finding solutions for the N-Queens problem using backtracking. The main thread waits for all these threads to complete their execution using the **join** method.

After all threads finishes, a new instance of the solutionsScreen class is created to display the solutions found for the N-Queens problem.

This **ThreadSolver** class represents a thread that recursively solves the N-Queens problem using backtracking. It explores possible queen placements on a chessboard, avoiding conflicts in rows, columns, and diagonals. The class maintains a list of solutions, and each thread contributes to this list when it finds a valid solution.

team members role:

ThreadSolver class: Abanoub Ibrahim, Ahmed Ibrahim

N_Qeeens_solver class: Basmala Hassan, Habiba Ahmed

GUI: Ahmed Ayman, Abobakr Khaled

code documentation:

In our code we have 6 classes:

1)InputScreen,: takes the input from the user when entered in the gui

2)N_Qeeens_solver: this is the main class where N is the size of the chessboard. it use multithreading, create separate threads to explore different starting positions for the first queen in the first row. Then call The ThreadSolver class

3)ThreadSolver: it solves the N-Queens problem concurrently using multiple threads.

It has 5 methods:

1. IsValid

Checks whether placing a queen at a specific position is valid, ensuring no other queens threaten it in the same column, diagonals, or row.

2. theBt

Implements the backtracking algorithm to find solutions recursively. It explores different combinations of queen placements on the chessboard.

3. run

Overrides the run method from the Thread class. When a thread is started, it executes the backtracking algorithm.

4. addSolution

Adds a synchronized solution to the shared **solutions** list, making sure that multiple threads can safely update the list without conflicts.

5. getAllSolutions

Returns the list of all solutions found by the threads.

4)Sol_OBJ: is a simple data structure representing a solution object for the N-Queens problem. It holds information about the configuration of the chessboard and the thread number associated with the solution.

5)Cell: this class is designed to represent a cell in a chessboard for the N-Queens problem visualization. It defines different states for the cell, such as being a white or black block and optionally containing a queen.

6)SolutionsScreen: it relies on the ThreadSolver class and the Cell class for functionality.

The constructor initializes the JFrame with various properties such as size, location, and layout, It adds a KeyListener (the class itself) to the frame,It then calls View_solution(0) to display the initial solution.

It has mainly 2 methods:

1. View_solution

- This method updates the frame to display the solution specified by the **solutions_counter**.
- It sets the title of the frame based on the solution number and thread number.
- It iterates over the N*N grid, checking the state of each cell in the solution, and adds a **Cell** to the frame accordingly.
- The frame is made visible at the end.

2. KeyPressed

- This method handles key events. If the right arrow key is pressed and there is a next solution, it removes all components from the frame and displays the next solution.
- If the left arrow key is pressed and there is a previous solution, it does the same to display the previous solution.