**N-Queen**

**Operating systems “N-Queen” project documentation**

Project Description

The code utilizes two threads to concurrently solve the N-Queens problem. The first thread traverses every square on the chessboard, using the guess function in the Search class. The guess function examines the feasibility of placing a queen in each square on the board. If a square is deemed legal, it is recorded as a potential solution.

The second thread validates the recorded solutions. It does so by employing functions in the Legal class, particularly the posibleQ function, which determines whether placing a queen at a specific location is legal or not. The second thread ensures that the recorded solutions adhere to the rules of the N-Queens problem.

This strategy represents a parallel approach where the first thread works on gathering potential solutions, while the second thread verifies the correctness of these solutions. The balance between the two threads is carefully managed to ensure the overall harmony of the N-Queens problem-solving process.

-We used 5 classes legal , Search

,TimerDraw ,HMap and User Interface

**What we have actually did:**

The program solves the N-Queens problem using two threads. One thread explores potential solutions, and the other validates them. Solutions are recorded in a hash map. The graphical user interface (GUI) allows users to navigate solutions, adjust the grid size, and view solving statistics. The program dynamically updates the GUI and provides timing information for the solving process.

**Team members roles:**

**Testing the code**: Khaled Ali, Abdelrahman Fathy, Ahmed Ayman

**Documentation**: Khaled Ali, Mohamed Allam, Eslam Ahmed

**Gui:** Mohamed Hussin, Sameh Mohamed

**Code Documentation:**

**Class: Legal**

Method: posibleQ(int i, int[][] queenBoard)

Parameters:

int i: Index representing the position of the queen on the chessboard.

int[][] queenBoard: 2D array representing the current state of the chessboard.

Return Type:

boolean: Indicates whether placing a queen at the specified position is possible.

Functionality:

Checks if placing a queen at the given position (i) on the chessboard (queenBoard) is legally possible.

Validates the position based on the presence of other queens on the same row, column, and diagonals.

**Class: Search**

Method: masterController()

Functionality:

Initiates the N-Queens puzzle-solving process.

Resets various parameters and data structures.

Calls the guess method to explore possible solutions.

Method: guess(int x, int y, int[][] queenBoard)

Parameters:

int x: Row index indicating the current row being explored.

int y: Column index indicating the current column being explored.

int[][] queenBoard: 2D array representing the current state of the chessboard.

Functionality:

Recursively explores possible solutions by placing queens on the chessboard.

Checks if placing a queen at the current position (x, y) is legal.

If a solution is found, adds the solution to the HMapSolutions data structure.

Method: unique()

Functionality:

Identifies unique solutions among the set of solutions.

Eliminates solutions that are rotations of each other.

Method: drawToArray(int[][] queenBoard)

Parameters:

int[][] queenBoard: 2D array representing the current state of the chessboard.

Functionality:

For debugging purposes, prints the current state of the chessboard to the console.

Method: run()

Functionality:

Implements the Runnable interface, defining the execution flow for the thread.

Executes the puzzle-solving process, measures time, and updates the user interface.

**Class: TimerDraw**

Method: run()

Functionality:

Implements the Runnable interface, defining the execution flow for the thread.

Periodically triggers a repaint of the user interface to display real-time updates.

**Class: UserInterface**

Method: paintComponent(Graphics g)

Parameters:

Graphics g: Graphics object used for painting.

Functionality:

Renders the graphical representation of the N-Queens puzzle and displays various statistics on the user interface.

Handles user interaction, such as arrow key navigation and grid size adjustment.

Method: computerThink()

Functionality:

Initiates the puzzle-solving process by creating a new thread (SearchThread).

Method: newDimension()

Functionality:

Initiates a new puzzle-solving process with a different grid size.

Adjusts the size of chessboard squares.

Method: getSolution()

Functionality:

Retrieves a specific solution from the set of solutions and updates the user interface.

Method: main(String[] args)

Parameters:

String[] args: Command-line arguments (not used in this method).

Functionality:

Entry point of the program.

Initializes the user interface and starts the timer thread for periodic updates.