Operating system-2 fall 2023-2024

Operating system-2 project documentation.

Project description:

Our project is more like a game called "N Queen" The N-Queens Solver is a program designed to find and visualize solutions to the classic N-Queens problem. The N-Queens problem involves placing N chess queens on an N×N chessboard so that no two queens threaten each other. The objective is to explore and generate all possible configurations where queens can be placed without violating the rules of the game.

the program takes the number of queens ('nQueens') as user input. The user is prompted to enter this value, and then the program proceeds to solve the N-Queens problem for the specified number of queens.

Example

If the user enters "8", the program will attempt to solve the 8-Queens problem using multithreading. The output will display one of the solutions (if found) and will be printed by one of the threads. Keep in mind that the program may find different solutions on different runs due to the multithreading approach.

What we have did:

Our journey with the N-Queens project was indeed challenging, but the collaborative effort of our team made it a success. In this project, we leveraged object-oriented programming and Java threads to implement a solution to the classic N-Queens problem, The primary goal of our project was to find solutions to the N-Queens problem by intelligently placing queens on an N×N chessboard, ensuring that no two queens threaten each other. To tackle this task, we utilized a 2D array to represent the chessboard and employed object-oriented principles to organize our code effectively. Multithreading Implementation is One of the key aspects of our implementation was the use of Java threads. We adopted a multithreaded approach to parallelize the search for solutions. With five threads running concurrently, each responsible for exploring different starting points, we aimed to optimize the efficiency of our solution space exploration, also Interrupt Mechanism

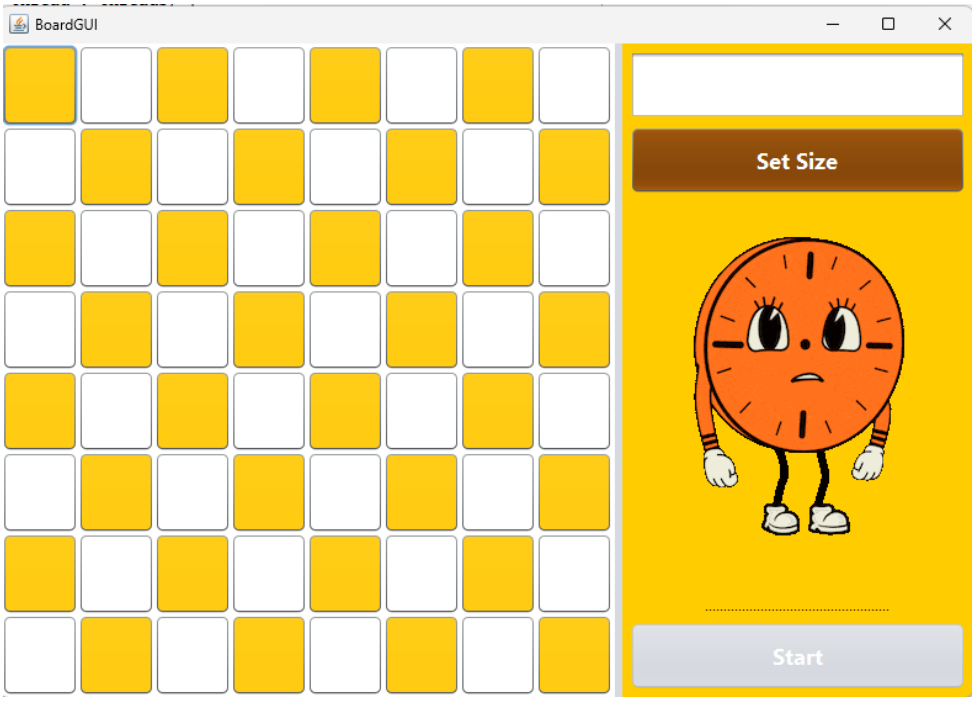
Team members roles:

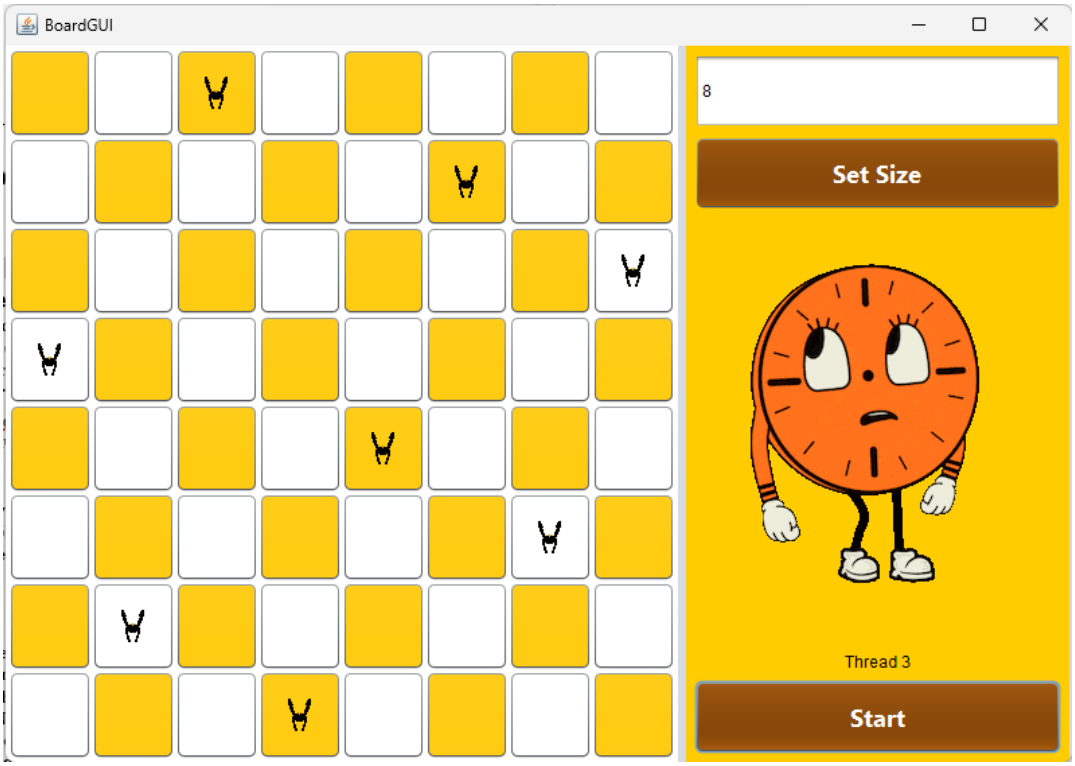
- **Alaa & Mohamed Ali: coding**
- **Mina: Algorithm**
- **Yousef & Sherif: GUI**
- **Nezar: Documentation**

GUI:

The N-Queens problem involves designing a visual representation of the chessboard and queens, allowing users to interact with the program.

Our GUI design aims to provide a seamless and intuitive experience for users interested in exploring solutions to the N-Queens problem. The combination of visual elements and interactive controls makes the project more accessible and user-friendly. The integration of solving logic with GUI responsiveness ensures a dynamic and engaging user experience.





Methods:

1) Class Multi-Threaded NQueens Solver:

This class encapsulates the main functionality of the N-Queens Solver.

- Takes user input for the number of queens (‘**nQueens**’).
- Checks if the input is positive and initializes the chessboard.
- Calls the’ **solveNQueensbythreading** ‘method.

```
1  package MultiThreadedNQueensSolver;
2  /**
3   *
4   * @author joola
5   */
6
7  public class MultiThreadedNQueensSolver {
8      public Thread Threads[];
9      private final chessGUI myBoard;
10     private final int[][] board;
11     private final int boardSize;
12     private final Object lock = new Object();
13     public MultiThreadedNQueensSolver(chessGUI myBoard,int boardSize){
14         this.myBoard=myBoard;
15         this.boardSize=boardSize;
16         this.board = new int[boardSize][boardSize];
17     }
18
```

2) is Safe Method:

Checks for existing queens in the same row and checks for conflicts in diagonal and anti-diagonal directions.

- **Checks if it is safe to place a queen in each cell ('row', ‘col').**
- **Examines the horizontal, diagonal, and anti-diagonal directions to ensure safety.**
- **Returns ‘true’ if safe, ‘false’ otherwise.**

```
19
20     private boolean isSafe(int row, int col) {
21         //  checker  yenfa3 a7ot queen wala la
22         for (int i = 0; i < col; i++) {
23             // byshof el queens ely mawgoden abl kda 34an y3rf y7ot elqueen elgdeda fen
24             if (board[row][i] == 1) {
25                 return false;
26             }
27         }
28
29         for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
30             // check diadonal
31             if (board[i][j] == 1) {
32                 return false;
33             }
34         }
35
36         for (int i = row, j = col; i < board.length && j >= 0; i++, j--) {
37             if (board[i][j] == 1) {
38                 return false;
39             }
40         }
41
42         return true;
43     }
```

3)print Final Queen Solution Method:

Prints the final placement of queens on the chessboard.

- **Synchronized method to ensure that only one thread at a time can print the solution.**
- **Iterates through the ‘board ‘array and prints each cell.**

```
44
45      private void  printFinalQueenSolution() {
46          synchronized (lock) {
47              // da el code ely byzhar fel Running t7t
48              ////////////////////////////////////////////////////////////////////////
49
50              for (int[] row : board) {
51                      //enter in row to fill
52                  for (int cell : row) {
53                      //fill every cell in this row
                        System.out.print(cell + " ");
55                  }
                    System.out.println();
57              }
                System.out.println();
59
60              ////////////////////////////////////////////////////////////////////////
61
62              try{
                    Thread.sleep(200);
64                  myBoard.updateChess(board,Thread.currentThread().getName());
                }catch(Exception ex){
66                  ///
67              }
68
69          }
70      }
71
```

4) Is All Queens Is Allocated Method:

Input: ‘col’ representing the current column being considered for queen placement.
Output: Returns ‘true’ if a solution is found, ‘false’ otherwise.

- **Recursive method that attempts to place queens in each row of the current column.**
- **If a solution is found, it prints the solution and returns ‘true’.**
- **Backtracks if no solution is found in the current branch.**

```
73
74          // btd5l el queens w ttcheck colum colum
75          if (col == board.length) {
76              printFinalQueenSolution();
77              return true;  // Indicate that a solution is found
78          }
79              printFinalQueenSolution();
80          // bnbda2 hna n insert lw zero fa mkan fadi laken lw wa7ed yb2a bn3ml
81          // backtracking 34an hya kda mlyana w msh hynf3 nzwd
82          for (int i = 0; i < board.length; i++) {
83              if (isSafe(i, col)) {
84                  board[i][col] = 1;
85                  if( IsAllQueensIsAllocated(col + 1)) {
86
87                      return true;  // Propagate the success signal
88                  }
89                  board[i][col] = 0;// backtrack
90              }
91
92
93          }
94
95          return false;  // Indicate that no solution is found in this branch
96      }
```

5) Interrupt Other Threads Method:

Array of threads (‘threads’) and the current thread (‘currentThread’).

Iterates through the array of threads and interrupts any alive threads that are not the current thread.

```
    public void interruptOtherThreads(Thread[] threads, Thread currentThread) {
99      // awl ma ytl3li el answer by3ml interruption lba2y el threads la2nna
100     //msh m7tagenhom 5las fa mlhomsh lazma enohm ysht8lo delwa2ty
101     for (Thread thread : threads) {
102         if (thread != currentThread && thread.isAlive()) {
103             // Interrupt other alive threads
104             thread.interrupt();
105         }
106     }
107 }
```

6) solve NQueens by threading Method:

Input: n representing the number of queens.

- **Creates an array of threads (‘threads’) with each thread responsible for solving the N- Queens problem for a specific starting row.**
- **Each thread starts by placing a queen in the first column and then calls**
- **IsAllQueensIsAllocated to find a solution.**
- **If a thread finds a solution, it interrupts other threads using “interruptOtherThreads”**
- **Threads are started and joined to the main thread.**

```
109     public void solveNQueensbythreading() {
110         // enna bnsh8l elthreads 3ala 3add el boardSize like 8x8 then 8 threads
111         //b3d kda byshof hal kol queen fe mkanha wala la
112         //lw fe mkanha yb2a 5las howa da el7l e3ml interrupt ll threads
113         // lw la2 yb2a e3ml
114         Thread[] threads = new Thread[this.boardSize];
115         Threads = threads;
116         for (int i = 0; i < this.boardSize; i++) {
117             final int row = i;
118             threads[i] = new Thread(() -> {
119                 board[row][0] = 1;
120                 if (IsAllQueensIsAllocated(1)) {
121                     // If this thread found a solution, interrupt other threads
122                     interruptOtherThreads(threads, Thread.currentThread());
123                 }
124
125                 board[row][0] = 0;
126             });
127                 threads[i].setName("Thread " + i);
128             threads[i].start();
129         }
130     }
131 }
132
```