جامعة حلوان HELWAN UNIVERSITY	Operating Systems 2 (Fall 2023) Project Discussion	كلية الحاسبات والذكاء الإصطناعي Faculty of Computers & Artificial Intelligence

Project Number: 4

Project Name: N Queen Problem

Project Description:

The N Queens problem challenges us to strategically place N chess queens on an N × N chessboard in such a way that no two queens can attack each other, either horizontally, vertically, or diagonally. To tackle this classic problem, the implementation will leverage the power of multi-threading, allowing for simultaneous exploration of solutions.

User Input:

- The user provides the board size (N), determining the dimensions of the N × N chessboard.

Output:

- The program generates and displays a chessboard with a legal placement of N queens that satisfies the problem constraints.
- Additionally, the program identifies and highlights the thread responsible for discovering the solution.

Team members roles:

	Team Member ID	Team member name (**in Arabic**)	role
1	20211057	يوسف احمد عبدالرؤف احمد	Threading and GUI
2	20211077	يوسف صلاح يوسف	Implement logic and Docs
3	20210520	عبدالرحمن عمرو محمد محمد	Implement logic and threading
4	20211036	هنا محمد مصطفى	Implement logic and testing
5	20211080	يوسف عبد المقصود محمد الحسيني	Threading and Docs
6	20211061	يوسف احمد محمود على	GUI – handling logic with GUI
7	20210322	رحاب ابراهيم علي	Implement logic and Docs

Code documentation:

```
import java.util.ArrayList;
public class NQueens implements Runnable{
    private  int[][] board;
    private final   int boardSize;
    private final BoardGUI newBoard;
    ArrayList<Thread> ThreadList = new ArrayList();
    Object sync = new Object();
    private boolean solved=false;
    private int updateTime;
    private String report;
    private int selectedColumn =0;
    public NQueens(int size,int updateTime,BoardGUI newBoard) {...9 lines }
    private void prepareBoard(int [][] b){...7 lines }
    public void printBoard(int [][] b) throws InterruptedException{...10 lines }
    private boolean isLegal(int [][] b,int row,int col){...4 lines }
    private boolean checkUpper(int [][] b,int row,int col){...7 lines }
    private boolean checkDiagonal(int [][] b,int row,int col){...16 lines }
    private boolean placeQueens(int row,int [][] local_board) throws InterruptedException{...30 lines }
    private void terminateOthers(int id){...8 lines }
    public void terminateAll(){...6 lines }
    public void solve(int column) throws InterruptedException{...19 lines }
    public void setUpdateTime(int updateTime){...3 lines }
    public void startQueen(){...6 lines }
    public void initQueen(){...7 lines }
    public void joinAll(){...9 lines }
    @Override
    public void run() {...15 lines }
}
```

Attributes:

1. board

- Type: int[][]
- Description: Represents the chessboard with queens placed or removed.

2. boardSize

- Type: int
- Description: Size of the chessboard (N).

3. newBoard

- Type: BoardGUI
- Description: GUI for displaying the chessboard.

4. ThreadList

- Type: ArrayList<Thread>
- Description: List to store threads for solving N-Queens problem.

5. sync

- Type: Object
- Description: Object for synchronization.

6. solved

- Type: boolean
- Description: Flag indicating if the N-Queens problem is solved.

7. updateTime

- Type: int
- Description: Time delay for updating the GUI.

8. report

- Type: String
- Description: Information about the solving process.

9. selectedColumn

- Type: int
- Description: Column selected for solving.

Methods:

```
public NQueens(int size,int updateTime,BoardGUI newBoard) {
    this.updateTime=updateTime;
    this.newBoard=newBoard;
    this.boardSize=size;
    newBoard.setVisible(b: true);
    board = new int[boardSize][boardSize];
    prepareBoard(b: board);
}
```

1. NQueens

- Header: public NQueens(int size, int updateTime, BoardGUI newBoard)
- Usage: Initializes the NQueens object with the specified parameters, sets up the chessboard, and prepares the GUI.

```
private void prepareBoard(int [][] b){
    for (int i = 0; i < boardSize; i++) {
        for (int j = 0; j < boardSize; j++) {
            b[i][j]=0;
        }
    }
}
```

2. prepareBoard

- Header: private void prepareBoard(int[][] b)
- Usage: Initializes the chessboard with zeros.

```
public void printBoard(int [][] b) throws InterruptedException{

    synchronized(sync){

            Thread.sleep(millis:updateTime);
            newBoard.updateBoard(myBoard: b,report);
```

3. printBoard

- Header: public void printBoard(int[][] b) throws InterruptedException
- Usage: Prints the chessboard to the GUI with a time delay.

```
private boolean isLegal(int [][] b,int row,int col){

    return (checkDiagonal(b,row,col) && checkUpper(b,row,col));
}
```

4. isLegal

- Header: private boolean isLegal(int[][] b, int row, int col)
- Usage: Checks if placing a queen at a given position is a legal move.

```
private boolean checkUpper(int [][] b,int row,int col){
    for (int i = row-1; i >= 0; i--) {
        if(b[i][col]==1)
            return false;
    }
    return true;
}
```

5. checkUpper

- Header: private boolean checkUpper(int[][] b, int row, int col)
- Usage: Checks if there is no queen in the upper part of the column.

```
private boolean checkDiagonal(int [][] b,int row,int col){
    int r=row-1,c=col-1;

    for (; r >= 0 && c>=0; r--,c--) {
        if(b[r][c]==1)
            return false;
    }
    r=row-1;
    c=col+1;

    for (; r >= 0 &&  c < boardSize; r--,c++) {
        if(b[r][c]==1)
            return false;
    }
    return true;
}
```

6. checkDiagonal

- Header: private boolean checkDiagonal(int[][] b, int row, int col)
- Usage: Checks if there is no queen in the diagonal positions.

```
private boolean placeQueens(int row,int [][] local_board) throws InterruptedException{
    //Base case
    if (row == boardSize)
        return true;

    for (int i = 0; i < boardSize; i++) {
        if(Thread.interrupted()) throw new InterruptedException();
        if(isLegal(b: local_board,row, col:i))
        {
            synchronized (sync) {  //check for it
                if(!solved)
                {
                    report="Thread "+Thread.currentThread().getName();

                    printBoard(b: local_board);
                }
            }  // check for it

            local_board[row][i]=1;

            //try the row after me (Recursion)
            if(placeQueens(row+1,local_board))
                return true;
            //if Q after me is not possible try next col and reset position
            local_board[row][i]=0;
        }
    }
    return false;
}
```

7. placeQueens

- Header: private boolean placeQueens(int row, int[][] local_board) throws InterruptedException
- Usage: Recursive method to place queens on the chessboard.

```
private void terminateOthers(long id){
    for (Thread t : ThreadList) {
        if(t.threadId() != id)
        {
            t.interrupt();
        }
    }
}
```

8. terminateOthers

- Header: private void terminateOthers(int id)
- Usage: Interrupts threads other than the current one.

```
public void terminateAll(){
        for (Thread t : ThreadList){
            t.interrupt();
        }
}
```

9. terminateAll

- Header: public void terminateAll()
- Usage: Interrupts all threads.

```
  public void solve(int column) throws InterruptedException{
      int[][] t_board = new int[boardSize][boardSize];
      prepareBoard(b: t_board);
      t_board[0][column]=1;
      boolean temp = placeQueens(row: 1, local_board: t_board);

      synchronized (sync){
          if(temp && !Thread.interrupted())
          {
              solved=temp;
              board=t_board;
              terminateOthers(id: Thread.currentThread().threadId());
              report="Thread "+Thread.currentThread().getName()+" Solved the Problem";
              printBoard(b: board);

          }
      }
  }
```

10. solve

- Header: public void solve(int column) throws InterruptedException
- Usage: Solves the N-Queens problem for a specific column.

```
public void setUpdateTime(int updateTime){
 this.updateTime=updateTime;
}
```

11. setUpdateTime

- Header: public void setUpdateTime(int updateTime)
- Usage: Sets the time delay for updating the GUI.

```
public void startQueen(){
    //Start all threads
    for (Thread t : this.ThreadList) {
        t.start();
    }
```

12. startQueen

- Header: public void startQueen()
- Usage: Starts all threads.

```
public void initQueen(){
    //Create ( n ) Number of threads
    for (int i = 0; i < boardSize; i++) {
        this.ThreadList.add(new Thread(task: this,""+i));
    }

}
```

13. initQueen

- Header: public void initQueen()
- Usage: Initializes and creates threads for solving the N-Queens problem.

```
public void joinAll(){
for(Thread t : ThreadList){
    try {
        t.join();
    } catch (InterruptedException ex) {

    }
}
}
```

14. joinAll

- Header: public void joinAll()
- Usage: Waits for all threads to finish.

```java
@Override
public void run() {
    int myColumn;
    synchronized(this){
        myColumn = selectedColumn;
        selectedColumn++;
    }
    try {
        solve(column:myColumn);
        System.out.println(x: "Thread done the job");
    } catch (InterruptedException ex) {
        System.out.println(x: "Thread Interrupted");
    }


}
```

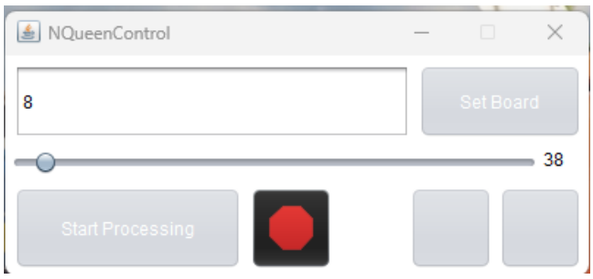
15. run

- Header: @Override public void run()
- Usage: Implements the run method from the Runnable interface, solves the N-Queens problem for a specific column when a thread is started.

GUI:

the control window

- here you can set the board size
- modify thread.sleep time
- start processing and intterupting it immediately
- change board colors from 2 default themes



the chessboard window

- a real time update for the board
- which thread is currently printing on the screen