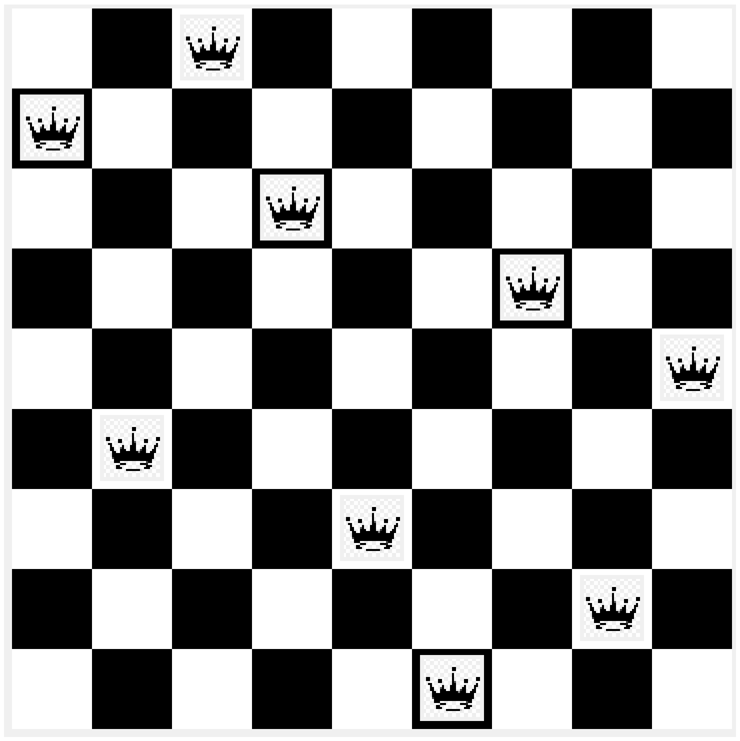
# N-Queens



## 1.Project description:

The N-Queens Problem Solver project aims to develop an

intelligent system capable of solving the N-Queens problem for various board sizes. The N-Queens problem is a

classic chessboard puzzle where the objective is to place N queens on an N×N chessboard in such a way that no

two queens threaten each other. Threatening means no two queens share the same row, column, or diagonal.

**But how can we achieve that?**

We can use backtracking to solve that problem, below we will see a fast summary about the backtracking:

Backtracking is a systematic algorithmic technique for finding all (or some) solutions to a computational problem that incrementally builds candidates for the solution, and abandons a candidate ("backtracks") as soon as it determines that the candidate cannot possibly be completed to a valid solution. This method is particularly useful for solving problems with a combinatorial nature, such as the N-Queens problem, Sudoku, and the Knight's Tour problem.

In the context of the N-Queens problem, backtracking involves placing queens on the chessboard one by one, and if the current placement violates any constraints, the algorithm backtracks and explores other possibilities.

**Now let us take a deep dive on the details of backtracking implementation:**

In our code, we implemented backtracking using mainly three functions:

## 1. solveNQ

The function initializes a 2D array board to represent the chessboard with dimensions n x n.

It populates each row of the board with a new array of size n, effectively creating an empty chessboard.

The solver method (solver) is then called with the initialized board, starting from the first row (0), and whichone to select a specific board to the thread to do operations with as parameters.

The function returns after calling the solver.

## 2. solver:

The function takes a chessboard represented by a 2D array (board), the current row (row), current column (col), and an additional parameter (whichOne). It returns a boolean indicating whether a solution has been found.

The main logic involves recursively attempting to place queens on the board, backtracking if necessary. The function checks for a valid placement using the isSafe method. The placement and removal of queens are tracked on a chessboard, and additional methods (placeXOnChessboard and removeElementFromChessboard) are called to update an external representation of the chessboard.

**Key points:**

- The base case (row >= n) is reached when all queens are successfully placed, and the function returns true.
- Queens are placed in the first row (row == 0) one at a time. If a valid position is found, the function recursively proceeds to the next row.
- For subsequent rows, a loop iterates through the columns (i) attempting to place queens and backtracking if needed.
- Then we call placeXOnChessboard function to place the queen.
- The isSafe method checks if placing a queen in the current position is safe based on the rules of the N-Queens problem.
- If the current position isnot safe, then we call removeElementFromChessboard  function to remove the queen from that position.
- The function uses backtracking by resetting the board if a solution is not found for a particular placement.

## 3. isSafe

The function checks whether it is safe to place a queen on a given position (row, col) on a chessboard represented by the 2D array board.

**Key points:**

- The function checks the column to ensure there is no queen in the same column (col) in any of the rows above the current row (row).
- It checks the left diagonal to ensure there is no queen in the path from the current position (row, col) to the top-left corner of the board.
- It checks the right diagonal to ensure there is no queen in the path from the current position (row, col) to the top-right corner of the board.
- If no conflicts are found in any of the checks, the function returns true, indicating that it is safe to place a queen at the specified position.

**But, what about the GUI?**

Gui implementation was quite complex as we should interact with the board as soon as the thread do its work, and as we know that the GUI is executing on the Event Dispatch Thread (EDT), so we cannot call the threads on the same thread as the GUI, or it will freeze and will not be updated on real time.

So, we handled that problem by updating the GUI and creating the n threads by using the power of SwingWorker for Background Processing, so we will not disturb the EDT

## Let's see a fast brief about the GUI implementation functions:

**Constructor NQueensss:**

- Initializes the GUI by setting up the JFrame and its components.
- Waits for the user to enter board size(n), and then shows him n boards with cell size = 50 pixels, where each board tries to reach to a solution.
- Sets up the layout, labels, text fields, and buttons for user interaction.

**createChessboardPanels:**

- Generates a list of panels to represent individual squares on the chessboard.
- Uses a GridLayout to arrange panels in a grid, alternating colors for a chessboard pattern.

**updateChessboard:**

- Updates the GUI with new chessboards and separators based on the current board size (n).
- Called when the board size changes or when initializing the GUI.

**updateChessboardInBackground:**

- Initiates background processing using SwingWorker.
- Creates multiple threads to solve the N-Queens problem for different chessboards in parallel.
- Updates the GUI during the solving process.

**createYourElement:**

- Scales and loads an image to represent the queen.

**createSeparator:**

- Creates a separator panel to add spacing between chessboards for better visualization.

**removeElementFromChessboard:**

- Removes the visual representation of a queen from the specified position on a chessboard.
- Uses threads and sleep to control the display timing.

**placeXOnChessboard:**

- Places a visual representation of a queen on the specified position of a chessboard.
- Uses threads and sleep for controlled display timing.

## Team Members Role

| | |
|---|---|
| GUI: | Ali Mohamed, Abdelrahman Safwat & Abdelrahman Salah |
| Back Tracking Implementation: | Omar Ahmed, Mohamed Ahmed & Abdelrahman Tawfik |
| Testing the Code: | Omar Ahmed, Abdelrahman Safwat & Abdelrahman Salah |
| Documentation: | Ali Mohamed, Mohamed Ahmed & Abdelrahman Tawfik |