****

**Superior University Lahore**

**Programming for AI (Lab)**

**Lab Task 1**

**N\_Queen Solver (Dynamic)**

**BS in Artificial Intelligence**

*Department of Software Engineering*

*Faculty of Computer Science & Information Technology*

*The Superior University, Lahore*

**Submitted by:**

|  |  |
| --- | --- |
| **Student Name:** | **Alisha Kanwal** |
| **Roll Number** | **Su92-bsaim-f23-105** |
| **Section** | **BSAI-4B** |

**Report**

# **Introduction:**

## Problem Statement:

The N-Queen Problem is a classic combinatorial puzzle that challenges users to place N queens on an N×N chessboard such that no two queens threaten each other horizontally, vertically, or diagonally. This report details an interactive Python-based solver that allows users to manually place queens on a customizable grid while enforcing the rules of the N-Queen problem. The program validates each move in real time, ensuring no conflicts arise during placement.

## Problem Understanding:

Given a grid of size rows×columns, the objective is to place queens sequentially such that:

* No two queens share the same row, column, or diagonal.
* The user interactively selects positions for queens, receiving immediate feedback on validity.

The program terminates when the user quits or successfully places queens without conflicts (though it does not enforce a full N-Queen solution unless the user manually achieves it).

# **Methodology**:

The solver employs an interactive validation approach:

* Grid Initialization: The user specifies the grid dimensions (e.g., 4×4, 8×8).
* Board Display: The chessboard is printed with row/column labels for clarity.
* User-Driven Placement: The user inputs queen positions one at a time.
* Conflict Checks: Each placement is validated against existing queens to ensure no row, column, or diagonal conflicts.
* Immediate Feedback: Invalid placements are rejected with an explanation, while valid ones update the board.

Unlike automated backtracking algorithms, this implementation emphasizes user interaction and real-time rule enforcement.

# **Implementation Details:**



The Python program follows these steps:

* Grid Input Handling:

The user enters grid dimensions in rowsxcolumns format (e.g., 4x4).

Input is validated for correct syntax and positive integers.

* Board Initialization:

A 2D list represents the board, initialized with empty cells (.).

* Board Display:

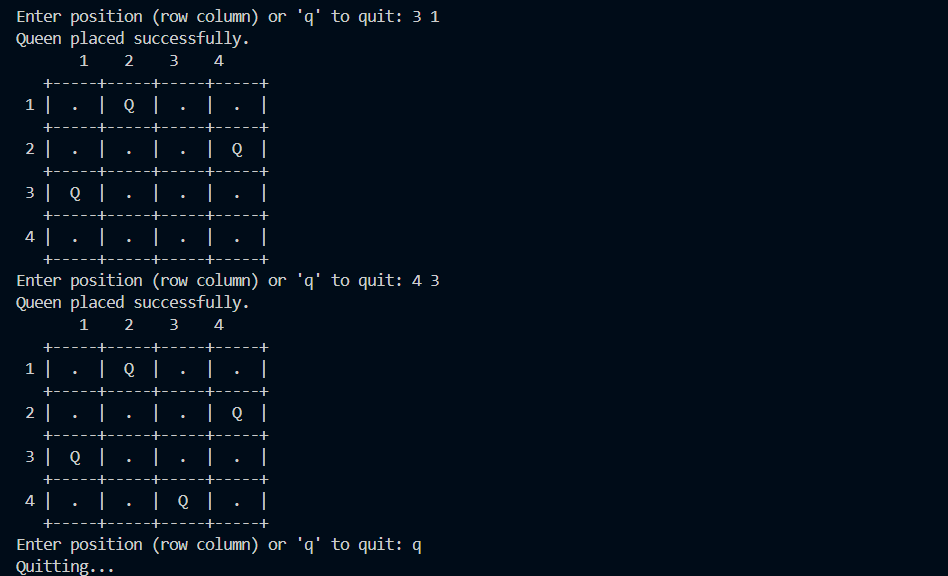
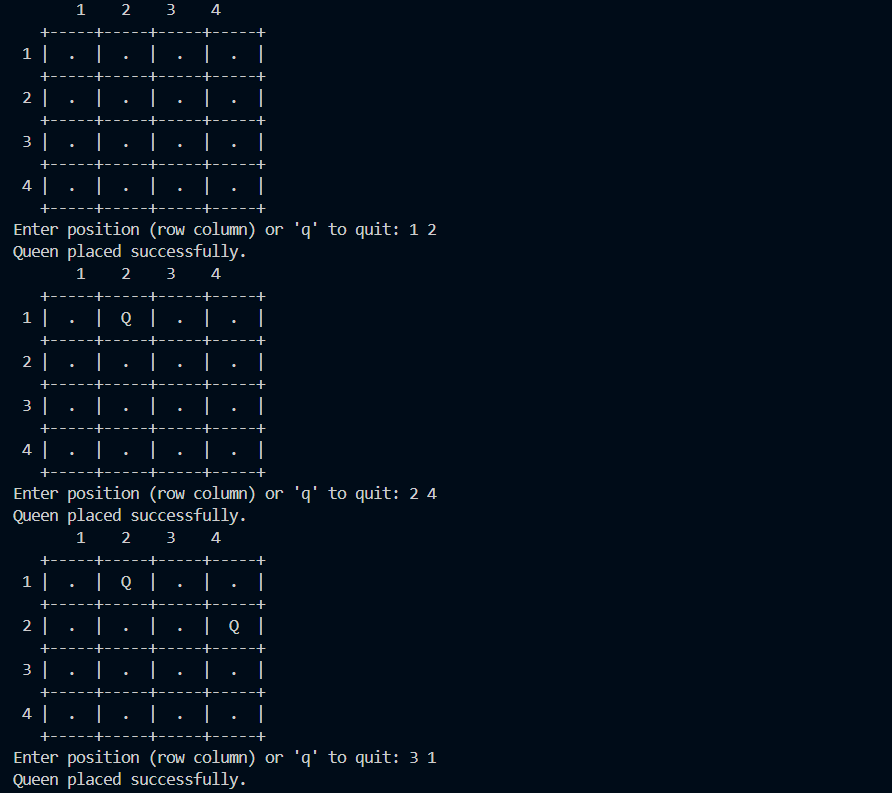
Columns are labeled numerically, and rows are separated by horizontal lines.

1. **Queen Placement Logic:**

* The user inputs positions as row column (e.g., 2 3).
* Inputs are validated for numeric values and grid boundaries.
* Conflicts are checked using:
* Same Row/Column: Compare coordinates of existing queens.
* Diagonal Check: |r1 - r2| == |c1 - c2| for any existing queen.

# **Example Execution:**

* Grid dimensions: 4x4
* Placement sequence:



# **Conclusion:**

This interactive N-Queen solver enforces the puzzle’s constraints in real time, allowing users to explore valid placements while learning the rules. Key features include:

* Clear visualization of the board with row/column labels.
* Immediate feedback on invalid moves.
* Support for custom grid dimensions.

The program demonstrates how user-driven problem-solving can be combined with algorithmic validation to create an educational tool for constraint-based puzzles. Future enhancements could include automated solution generation or a win condition for full N-Queen placement.