

**The Superior University Lahore**

**Faculty of Computer Science & Information**

**Technology**

**Name: Alishba Haroon**

**Roll No: BSAI-116-4C**

**Date: 17 Feb 2025**

**Subject: PAI LAB**

**N-Queens Problem**

The N-Queens problem is a classic puzzle in chess and computer science. The objective is to place N queens on an N × N chessboard while following these rules:

**Rules of the N-Queens Problem**

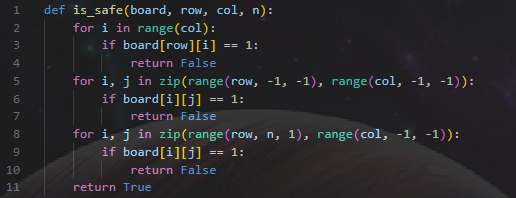
1. One Queen per Row & Column – Each queen must be placed in a unique row and column.
2. No Attacking Queens – No two queens can share the same row, column, or diagonal.
3. Board Size is N × N – The number of queens equals the board size (N).
4. Solutions Depend on N – Some values (like N = 2, 3) have no solutions, while others have multiple.
5. Backtracking is Used – The algorithm places queens, checks for safety, and backtracks if needed

**Code Explanation:**

**1. is\_safe(board, row, col, n):**

This function checks if a queen can be safely placed at a given position (row, col) on the board.

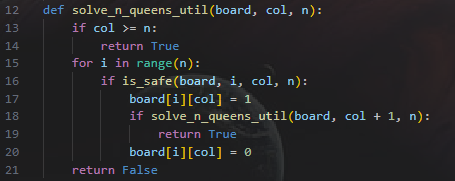
* First loop: Checks the left side of the current row to ensure no queen is placed before in the same row.
* Second loop: Checks the upper-left diagonal to see if any queen is already placed there.
* Third loop: Checks the lower-left diagonal to prevent diagonal attacks from below.
* If all checks pass, it returns True, meaning the position is safe.

****

**2. solve\_n\_queens\_util(board, col, n):**

This recursive backtracking function attempts to place queens in the given column and recursively solves the rest of the board.

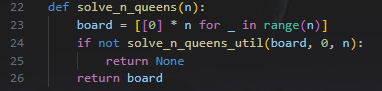
* Base Case: If all queens are placed (col >= n), it returns True, meaning a valid solution is found.
* Loop: Tries placing a queen in every row of the current column and checks if it's safe.
* If it's safe, the function places a queen (board[i][col] = 1) and recursively calls itself for the next column.
* If the recursive call returns True, the correct placement and the solution are found.
* Otherwise, the queen is removed (board[i][col] = 0) and the function backtracks.
* If no safe position is found in the column, it returns False, meaning the previous placement was incorrect, and backtracking occurs.

****

**3. solve\_n\_queens(n):**

This function initializes the chessboard and calls the recursive solver:

* Creates an N x N board filled with 0s.
* Calls solve\_n\_queens\_util(board, 0, n) to start placing queens from the first column.
* If no solution exists, it returns None; otherwise, it returns the solved board.

****

**4. print\_board(board):**

This function prints the chessboard in a readable format:

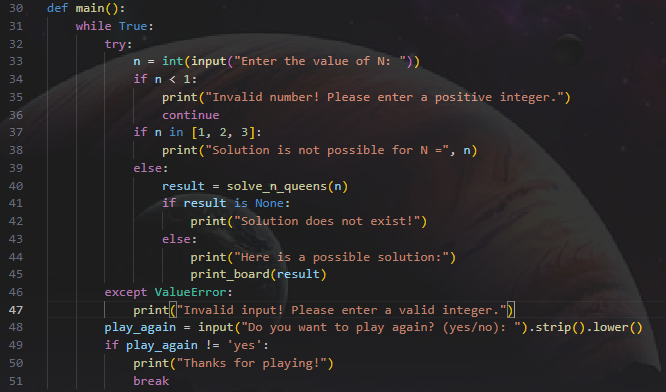
* It replaces 1s with "Q" to represent queens.
* It replaces 0s with "." for empty spaces.

****

**5. main() Function**

This function manages user input and game flow:

* It asks the user for N.
* If N < 1, it asks for a valid number.
* If N = 1, 2, 3, it informs the user that no solution is possible.
* Otherwise, it solves the problem and prints the board.
* After each attempt, it asks the user if they want to play again.

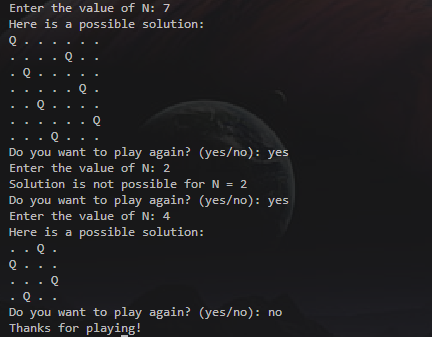
****

**6. if \_\_name\_\_ == "\_\_main\_\_": main()**

This ensures that the script runs only when executed directly, not when imported as a module.

****

**OUTPUT:**

****