SUPERIOR UNIVERSITY

NAME: MINAHIL IQBAL

ROLL NO: 062

SECTION: BSAI-4B

SUBJECT: PAI (LAB)

SUBMITTED TO: SIR RASIKH

Report on N-Queens Problem Implementation

Introduction

The N-Queens Problem is a classic combinatorial problem in which N queens must be positioned on an N × N chessboard in such a way that none of them can attack another. Its importance in AI and algorithmic software development stems from its numerous conditions and the intricacy of providing an answer.

The current report evaluates the provided implementation of the N-Queens Game in Python, which permits the user to interactively position the queens on the board with the guarantee that illegal placements will not be accepted.

Definition OF The Problem

The difficulty is to position N queens on an N × N chessboard while adhering to the following criteria: 1. At most one queen is permitted in each row. 2. At most one queen is permitted in each column. 3. At most one queen is permitted in each diagonal (both major and minor diagonals).

Program Overview

The given Python program allows a user to interactively place queens on the chessboard. It includes functions for:

- Displaying the board (print_board)
- Validating queen placement (is_valid)
- Checking if more queens can be placed (can_place_more_queens)
- Handling game logic (n_queens_game)

1. Board Representation

The chessboard is implemented as a 2D list (board), initialized with . to indicate empty spaces:

board = [['.' for _ in range(N)] for _ in range(N)]

Queens are represented by 'Q' when placed on the board.

2. Printing the Board

The function print_board(board) displays the board by joining row elements with spaces:

```
def print_board(board):
    for row in board:
        print(" ".join(row))
    print()
```

3. Validation of Queen Placement

The function is_valid(board, row, col, N) ensures that a queen is not placed in an attacking position. It checks:

- Column conflicts
- Row conflicts
- Diagonal conflicts (major and minor diagonals)

4. Checking for Available Moves

The function can_place_more_queens(board, N) iterates through the board to see if any valid moves are left.

5. Game Logic

The main function n_queens_game(N) implements the interactive game loop:

- Initializes an empty board.
- Prompts the user to input row and column positions.
- Validates and places queens based on user input.
- If no valid moves are left, prompts the user to restart or change board size.
- Ends when all N queens are placed correctly.

6. Handling User Input

The game accepts user inputs for row and column placement. It includes:

- Input validation (checking if input is numeric and within bounds)
- Restart option (if no moves are left, users can restart with a different board size)
- Exit option (quit command allows users to exit the game)

Strengths of the Implementation

- User Interaction: Allows users to manually play the N-Queens game interactively.
- Comprehensive Validation: Ensures that every queen placement follows game rules.
- Restart Mechanism: Offers the user an option to restart with a new board size.
- Error Handling: Prevents invalid moves and incorrect inputs.

Limitations and Possible Improvements

- Manual Placement Only: The program does not automatically solve the N-Queens problem.
- Inefficient Validation: is_valid function checks entire rows and columns, which can be optimized.
- Repeated Code: The program has some redundant functions that can be refactored.
- Stack Overflow Risk: Recursive calls in case of a restart can cause excessive memory usage.

Possible Enhancements

- Implement an Automated Solver: Use backtracking or other algorithms to automatically place queens.
- Optimize the is_valid function: Reduce unnecessary checks for better efficiency.
- Enhance User Interface: Improve display using a GUI (e.g., tkinter) for a better experience.
- Leaderboard or Move Counter: Track the number of moves taken to solve the game.

Conclusion

This implementation provides an interactive approach to the N-Queens problem, allowing users to manually place queens while enforcing validity rules. While effective, it can be enhanced with automated solving, performance optimizations, and UI improvements.

OUTPUT

```
Enter the size of the chessboard (N): 4
Welcome to the N-Queens Game!
. . . .
. . . .
. . . .
. . . .

Enter row (0-3) for queen 1 (or type 'quit' to exist):  0
Enter col (0-3) for queen 1 (or type 'quit' to exist):  1
. Q . .
. . . .
. . . .
. . . .

Enter row (0-3) for queen 2 (or type 'quit' to exist):  3
Enter col (0-3) for queen 2 (or type 'quit' to exist):  3
. Q . .
. . . .
. . . .
. . . Q

Enter row (0-3) for queen 3 (or type 'quit' to exist):  2
Enter col (0-3) for queen 3 (or type 'quit' to exist):  0
. Q . .
. . . .
Q . . .
. . . Q

No more valid moves! Restarting the game...
Do you want to play with a different board size? (yes/no/quit): ↑↓ for history. Search history with c-↑/c-↓
```