Project Report: N-Queens Problem Representation with GUI

Abstract

This project addresses the N-Queens problem by implementing a Python program that models the states of the problem using object-oriented design. The solution includes functionalities to determine safe positions, compute heuristic values, and verify goal states. Additionally, a graphical user interface (GUI) is developed using the tkinter library to enhance user interaction by visually representing the board, initial states, and the solution.

Introduction

The N-Queens problem involves placing N chess queens on an NxN chessboard such that no two queens threaten each other. This entails ensuring that no two queens share the same row, column, or diagonal. The problem has applications in optimization and constraint satisfaction, making it a cornerstone of artificial intelligence research.
This project provides:
1. A class-based implementation for representing and solving the problem.
2. A GUI for dynamic input and visualization.

Problem

The task involves:
1. Representing states of the N-Queens problem as a class.
2. Implementing methods to:
   - Calculate heuristic values.
   - Determine whether a state is a goal state.
   - Generate valid next states.
3. Providing an interactive GUI for entering board size, initial configurations, and solving the problem.

Scope

Functional Scope: Solving the N-Queens problem for any N >= 4.

Technical Scope:
- Use of object-oriented programming principles.
- Implementation of a user-friendly GUI for input, validation, and visualization.

Design

1. Backend Implementation

- Class `NQueens`:
  - Attributes:
    - `n`: Size of the board.
    - `board`: Current board configuration.

- Methods:
  - `is_safe(row, col)`: Validates if a queen can be placed at a given position.
  - `heuristic_value()`: Computes the heuristic value as the number of conflicting pairs of queens.
  - `is_goal_state()`: Checks if the current configuration satisfies the goal conditions.
  - `place_queens(row)`: Implements the backtracking algorithm to solve the problem.
  - `reset_board()`: Resets the board to an empty state.

2. GUI Implementation

Developed using tkinter.
Features:
- Input fields for board size and initial positions.
- Buttons for displaying the initial state and solving the problem.
- Visual representation of the board with queens.
Dynamic Interactions:
- Real-time board updates on input changes.
- Resizable canvas for adaptive visualization.

Illustration

1. Initial State:
   - Input: Board size = 4, Positions = `[1, 3, 0, 2]`.
   - Display: The board with queens placed at the specified columns.
   - Output: Heuristic value calculated and displayed.
   - Goal State Check: Confirms if the input is a solution.

2. Solved State:
   - Input: Solve button clicked for N = 4.
   - Display: Solution with queens placed without conflicts.
   - Output: Heuristic value = 0.

Validation

Input Validation: Ensures board size is at least 4 and column indices are valid.
Heuristic Calculation: Accurately computes the heuristic based on queen conflicts.
Goal State Verification: Confirms that the solution meets the problem's constraints.

Architecture

Backend: Python-based implementation using a class for core logic.
Frontend: GUI built with tkinter, providing a responsive and user-friendly interface.

Tools and Technologies

Python 3.x
`tkinter` for GUI development

Future Work

- Extend the GUI to allow dynamic adjustments of the board size during runtime.
- Implement additional heuristic algorithms for faster solutions.
- Integrate animations for queen placement during the solving process.

Conclusion

The project successfully implements a robust solution to the N-Queens problem with an intuitive GUI. The solution provides accurate heuristic calculations and visual feedback, making it both educational and practical for understanding the problem's dynamics.