Lab Assignment 6

Subject: Artificial Intelligence **Guided by:** Dr. Anuradha Yenkikar

Student name: Neha Sawant

Experiment Name: Implement the 8-Queens Problem

Objective:

The objective of this lab assignment is to implement the 8-Queens problem using backtracking, a classic example of constraint satisfaction problems. The goal is to place eight queens on an 8x8 chessboard such that no two queens threaten each other.

Problem Statement:

In chess, a queen can move any number of squares vertically, horizontally, or diagonally. The challenge is to place eight queens on a chessboard in such a way that no two queens share the same row, column, or diagonal.

Requirements:

- Programming Language: Java
- Environment: Any Java IDE (e.g., IntelliJ IDEA, Eclipse)
- Java Version: 8 or higher

Code Explanation:

1] Class Overview

The primary class for the implementation is `EightQueens`. The class contains several methods to check safety, save the board configuration, and perform the backtracking search.

2] Method Descriptions

- isSafe(int row, int col, char[][] board):

This method checks if placing a queen at the specified position (row, col) is safe. It checks for conflicts in:

- The same row.
- The same column.
- Both diagonals (left-up, right-up, left-down, right-down).

- saveBoard(char[][] board, List<List<String>> allBoards):

This method saves the current board configuration into a list. Each board configuration is represented as a list of strings.

- helper(char[][] board, List<List<String>> allBoards, int col):

This recursive method attempts to place queens in each column of the board. If a valid placement is found, it calls itself to place queens in the next column. If all queens are placed successfully, it saves the board configuration.

- solveQueens(int n):

This method initializes the board and starts the recursive search. It returns all valid configurations of the board.

- main(String[] args):

The entry point of the program, where it initializes the board size (8 in this case) and prints the solutions.

3] Code Implementation

Here's the complete implementation of the 8-Queens problem:

```java
import java.util.ArrayList;
import java.util.List;

public class EightQueens {

    //  8- queens problem
    public static boolean isSafe(int row, int col,char[][] board){
        // horizontal
        for(int i=0;i<board.length;i++){
            if(board[row][i]=='Q'){
                return false;
            }
        }

        // vertical
        for(int i=0;i<board.length;i++){
            if(board[i][col]=='Q'){
                return false;
            }
        }

        // left-up
        for(int i=row,j=col;i>=0 && j>=0;i--,j--){
            if(board[i][j]=='Q'){
                return false;
            }
        }

        // right-up
        for(int i=row,j=col;i>=0 && j<board.length;i--,j++){
```

```java
        if(board[i][j]=='Q'){
            return false;
        }
    }

    // left-down
    for(int i=row,j=col;i<board.length && j>=0;i++,j--){
        if(board[i][j]=='Q'){
            return false;
        }
    }

    // right-down
    for(int i=row,j=col;i<board.length && j<board.length;i++,j++){
        if(board[i][j]=='Q'){
            return false;
        }
    }

    return true;
}
public static void saveBoard(char[][] board,List<List<String>> allBoards){

    List<String> newBoard=new ArrayList<>();

    for(int i=0;i<board.length;i++){
        String row="";
        for(int j=0;j<board.length;j++){
```

```java
public class EightQueens {
    public static void saveBoard(char[][] board,List<List<String>> allBoards){
            for(int j=0;j<board.length;j++){
                if(board[i][j]=='Q'){
                    row+='Q';
                }else{
                    row+='_';
                }
            }
            newBoard.add(row);
        }

        allBoards.add(newBoard);
    }
    public static void helper(char[][] board, List<List<String>> allBoards,int col){
        if(col==board.length){
            saveBoard(board,allBoards);
            return;
        }
        for(int row=0;row<board.length;row++){
            if(isSafe(row,col,board)){
                board[row][col]='Q';
                helper(board, allBoards, col+1);
                board[row][col]='.';
            }
        }
    }
    public static List<List<String>> solveQueens(int n){
        List<List<String>> allBoards=new ArrayList<>();
        char[][] board=new char[n][n];
```

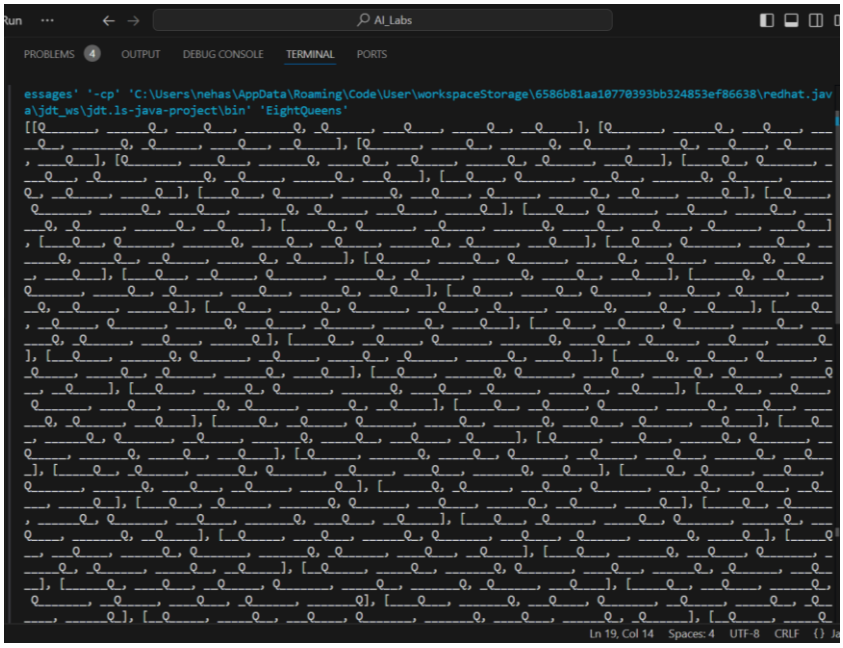
```java
public class EightQueens {
    public static List<List<String>> solveQueens(int n){

        helper(board, allBoards, col:0);

        return allBoards;
    }
    Run | Debug
    public static void main(String args[]){
        // 8-queens problem
        int n=8;
        System.out.println(solveQueens(n));
    }
}
```

Sample Output:

The program will output the number of solutions found and each valid board configuration. An example output might look like this:



Conclusion:

In this lab, I have successfully implemented the 8-Queens problem using basic search strategies and backtracking. This exercise has enhanced my understanding of recursive algorithms and constraint satisfaction problems.