Day 16&17

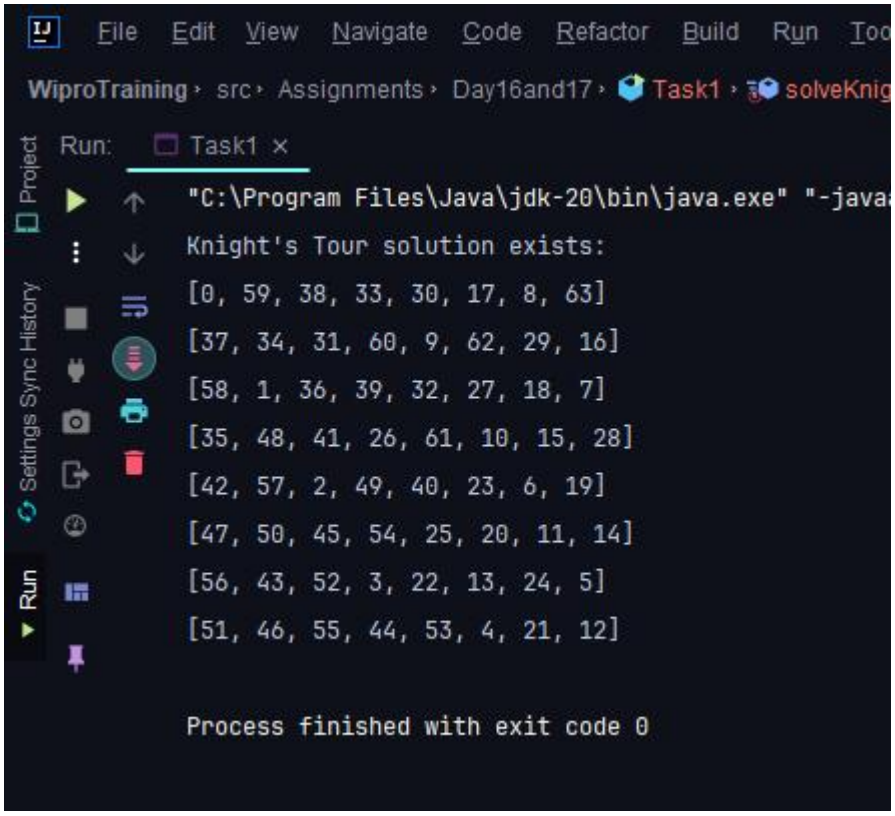
Task 1: The Knight's Tour Problem
Create a function bool SolveKnightsTour(int[,] board, int moveX, int moveY, int moveCount, int[] xMove, int[] yMove) that attempts to solve the Knight's Tour problem using backtracking. The function should return true if a solution exists and false otherwise. The board represents the chessboard, moveX and moveY are the current coordinates of the knight, moveCount is the current move count, and xMove[], yMove[] are the possible next moves for the knight. Fill the chessboard such that the knight visits every square exactly once. Keep the chessboard size to 8x8.

Program:

```
package Assignments.Day16and17;

import java.util.Arrays;

public class Task1 {

   private static final int N = 8; // Chessboard size

   public static boolean isValidMove(int x, int y, int[][] board) {
      return x >= 0 && x < N && y >= 0 && y < N && board[x][y] == -1;
   }

   public static boolean solveKnightsTour(int[][] board, int x, int y, int
moveCount, int[] xMove, int[] yMove) {
      if (moveCount == N * N) {
         // All squares visited
         return true;
      }

      for (int i = 0; i < N; i++) {
         int nextX = x + xMove[i];
         int nextY = y + yMove[i];
         if (isValidMove(nextX, nextY, board)) {
            board[nextX][nextY] = moveCount;
            if (solveKnightsTour(board, nextX, nextY, moveCount + 1, xMove,
yMove)) {
               return true;
```

```
            }
            board[nextX][nextY] = -1; // Backtrack
        }
    }

    return false;
  }

  public static void main(String[] args) {
    int[][] board = new int[N][N];
    for (int i = 0; i < N; i++) {
        Arrays.fill(board[i], -1);
    }

    int[] xMove = {2, 1, -1, -2, -2, -1, 1, 2};
    int[] yMove = {1, 2, 2, 1, -1, -2, -2, -1};

    board[0][0] = 0;
    boolean solutionExists = solveKnightsTour(board, 0, 0, 1, xMove,
yMove);

    if (solutionExists) {
        System.out.println("Knight's Tour solution exists:");
        for (int[] row : board) {
            System.out.println(Arrays.toString(row));
        }
    } else {
        System.out.println("No Knight's Tour solution exists.");
    }
  }
}
```

Output:



Task 2: Rat in a Maze

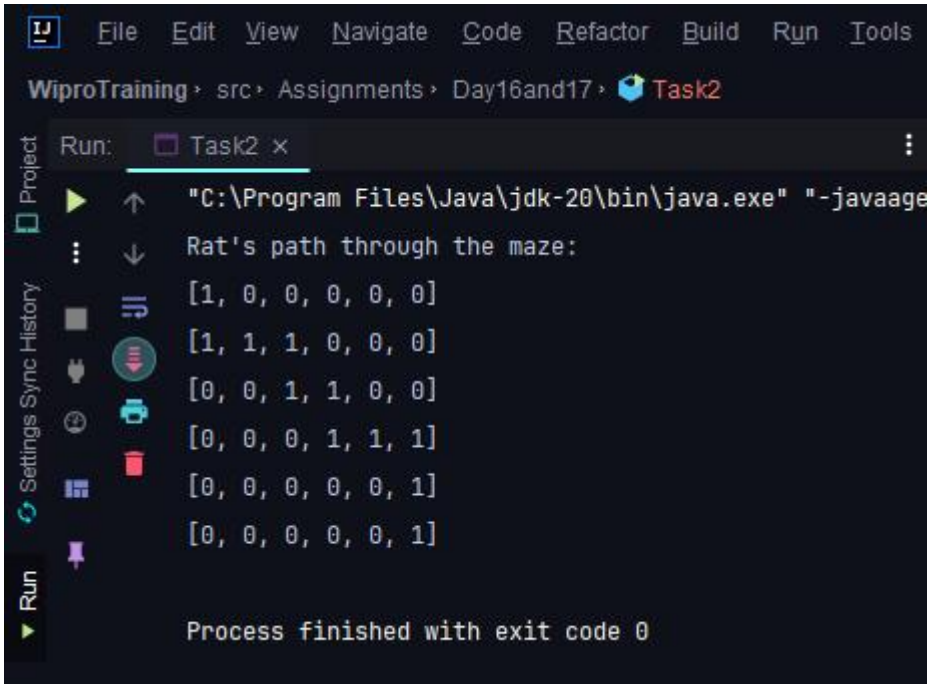
Implement a function bool SolveMaze(int[,] maze) that uses backtracking to find a path from the top left corner to the bottom right corner of a maze. The maze is represented by a 2D array where 1s are paths and 0s are walls. Find a rat's path through the maze. The maze size is 6x6.

Program:

```
package Assignments.Day16and17;

import java.util.Arrays;

public class Task2 {

    private static final int N = 6; // Maze size

    public static boolean isValidMove(int x, int y, int[][] maze) {
        return x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1;
    }

    public static boolean solveMaze(int[][] maze, int x, int y, int[][] path) {
        if (x == N - 1 && y == N - 1) {
            // Reached the bottom right corner
            path[x][y] = 1;
```

```
            return true;
        }

        if (isValidMove(x, y, maze)) {
            path[x][y] = 1;

            // Move right
            if (solveMaze(maze, x, y + 1, path)) {
                return true;
            }
            // Move down
            if (solveMaze(maze, x + 1, y, path)) {
                return true;
            }

            // Backtrack
            path[x][y] = 0;
        }

        return false;
    }

    public static void main(String[] args) {
        int[][] maze = {
                {1, 0, 1, 1, 1, 0},
                {1, 1, 1, 0, 1, 1},
                {0, 0, 1, 1, 0, 1},
                {1, 0, 0, 1, 1, 1},
                {1, 1, 1, 0, 0, 1},
                {1, 1, 1, 1, 1, 1}
        };

        int[][] path = new int[N][N];
        boolean solutionExists = solveMaze(maze, 0, 0, path);

        if (solutionExists) {
            System.out.println("Rat's path through the maze:");
            for (int[] row : path) {
                System.out.println(Arrays.toString(row));
            }
        } else {
            System.out.println("No solution exists.");
        }
    }
}
```

Output:



Task 3: N Queen Problem

Write a function bool SolveNQueen(int[] board, int col) in java that places N queens on an N x N chessboard so that no two queens attack each other using backtracking. Place N queens on the board such that no two queens can attack each other. Use a standard 8x8 chessboard.

Program:

```
package Assignments.Day16and17;

import java.util.Arrays;

public class Task3 {

    private static final int N = 8;

    public static boolean isValidMove(int[][] board, int row, int col) {

        for (int i = 0; i < row; i++) {
            if (board[i][col] == 1 ||
                    (col - (row - i) >= 0 && board[i][col - (row - i)] == 1) ||
                    (col + (row - i) < N && board[i][col + (row - i)] == 1)) {
                return false;
            }
        }
```

```
        return true;
    }

    public static boolean solveNQueens(int[][] board, int row) {
        if (row == N) {
            return true;
        }

        for (int col = 0; col < N; col++) {
            if (isValidMove(board, row, col)) {
                board[row][col] = 1;
                if (solveNQueens(board, row + 1)) {
                    return true;
                }
                board[row][col] = 0;
            }
        }

        return false;
    }

    public static void main(String[] args) {
        int[][] board = new int[N][N];
        boolean solutionExists = solveNQueens(board, 0);

        if (solutionExists) {
            System.out.println("N-Queens solution:");
            for (int[] row : board) {
                System.out.println(Arrays.toString(row));
            }
        } else {
            System.out.println("No solution exists.");
        }
    }
}
```

Output:

File Edit View Navigate Code Refactor Build Run To

WiproTraining › src › Assignments › Day16and17 › Task3

Run: Task3 ×

```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-jav
N-Queens solution:
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]

Process finished with exit code 0
```