Day 16 and 17 Assignment

1. **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.**

```java
package m5_core_java_programming.day_20;

/*
   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.
*/
public class Assignment_1 {

   public static boolean SolveKnightsTour(int[][] board, int moveX, int moveY,
int moveCount, int[] xMove, int[] yMove) {
       if (moveCount == board.length * board.length) {
           for (int i = 0; i < board.length; i++) {
               for (int j = 0; j < board[0].length; j++) {
                   System.out.printf("%3d", board[i][j]);
               }
               System.out.println();
           }

System.out.println("-----------------------------------------------------------
--------");
           return true;
       }

       for (int i = 0; i < xMove.length; i++) {
           int x = moveX + xMove[i];
           int y = moveY + yMove[i];
           if (isValid(x, y, board)) {
               board[x][y] = moveCount;
```

```
                if (SolveKnightsTour(board, x, y, moveCount + 1, xMove, yMove))
{
                    return true;
                } else {
                    board[x][y] = 0;
                }
            }
        }

        return false;
    }

    public static boolean isValid(int x, int y, int[][] board) {
        if (x < 0 || y < 0 || x >= board.length || y >= board[0].length ||
board[x][y] != 0) {
            return false;
        }
        return true;
    }

    public static void main(String[] args) {
        int n = 8;
        int[][] board = new int[n][n];
        System.out.println("Current Board ");
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                System.out.printf("%3d", board[i][j]);
            }
            System.out.println();
        }

System.out.println("----------------------------------------------------------
--------");
        int[] xMove = {-2, -1, 1, 2, 2, 1, -1, -2};
        int[] yMove = {1, 2, 2, 1, -1, -2, -2, -1};

System.out.println("----------------------------------------------------------
--------");
        System.out.println("After Knights Tour");
        board[0][0] = 1;
        SolveKnightsTour(board, 0, 0, 1, xMove, yMove);
    }
}
```

Output

```
C:\Users\coolr\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Program Files\JetB
Current Board
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
------------------------------------------------------------------------
------------------------------------------------------------------------
After Knights Tour
  1 37 54 33  2 35 18 21
 53 46  1 36 19 22  3 16
 38 55 32 45 34 17 20  9
 47 52 39 56 23 10 15  4
 58 31 44 51 40 25  8 11
 43 48 57 24 61 14  5 26
 30 59 50 41 28  7 12 63
 49 42 29 60 13 62 27  6
------------------------------------------------------------------------

Process finished with exit code 0
```

---

2. **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.**

```
package m5_core_java_programming.day_20;

/*
   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.
*/
```

```java
public class Assignment_2 {

   public static boolean SolveMaze(int[][] maze) {
       System.out.println("Maze before travelling ");
       for (int i = 0; i < maze.length; i++) {
           for (int j = 0; j < maze[0].length; j++) {
               System.out.printf("%3d", maze[i][j]);
           }
           System.out.println();
       }

System.out.println("------------------------------------------------------------
--------");
       int[] moveX = {-1, 0, 1, 1};
       int[] moveY = {0, 1, 0, -1};
       boolean res = ratMove(maze, 0, 0, moveX, moveY);
       return true;
   }

   public static boolean ratMove(int[][] maze, int x, int y, int[] moveX, int[]
moveY) {
       if (x == maze.length - 1 && y == maze.length - 1) {
           return true;
       }
       for (int i = 0; i < 4; i++) {
           int newX = moveX[i] + x;
           int newY = moveY[i] + y;
           if (isValid(newX, newY, maze)) {
               int prev = maze[newX][newY];
               maze[newX][newY] = 2;
               if (ratMove(maze, newX, newY, moveX, moveY)) {
                   return true;
               } else {
                   maze[newX][newY] = prev;
               }
           }
       }
       return false;
   }

   public static boolean isValid(int x, int y, int[][] maze) {
       if (x < 0 || y < 0 || x >= maze.length || y >= maze[0].length ||
maze[x][y] != 1) {
           return false;
       }
       return true;
   }

   public static void main(String[] args) {
```

```
        int[][] maze = {
                {1, 0, 0, 0, 0},
                {1, 0, 0, 0, 0},
                {1, 1, 0, 0, 0},
                {0, 1, 1, 1, 1},
                {0, 0, 0, 1, 1}
        };

        System.out.println("Can rat reach destination " + SolveMaze(maze));
    }
}
```

Output

```
C:\Users\coolr\.jdks\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Program Fi
Maze before travelling
  1  0  0  0  0
  1  0  0  0  0
  1  1  0  0  0
  0  1  1  1  1
  0  0  0  1  1
------------------------------------------------------------------------
Can rat reach destination true

Process finished with exit code 0
```

---

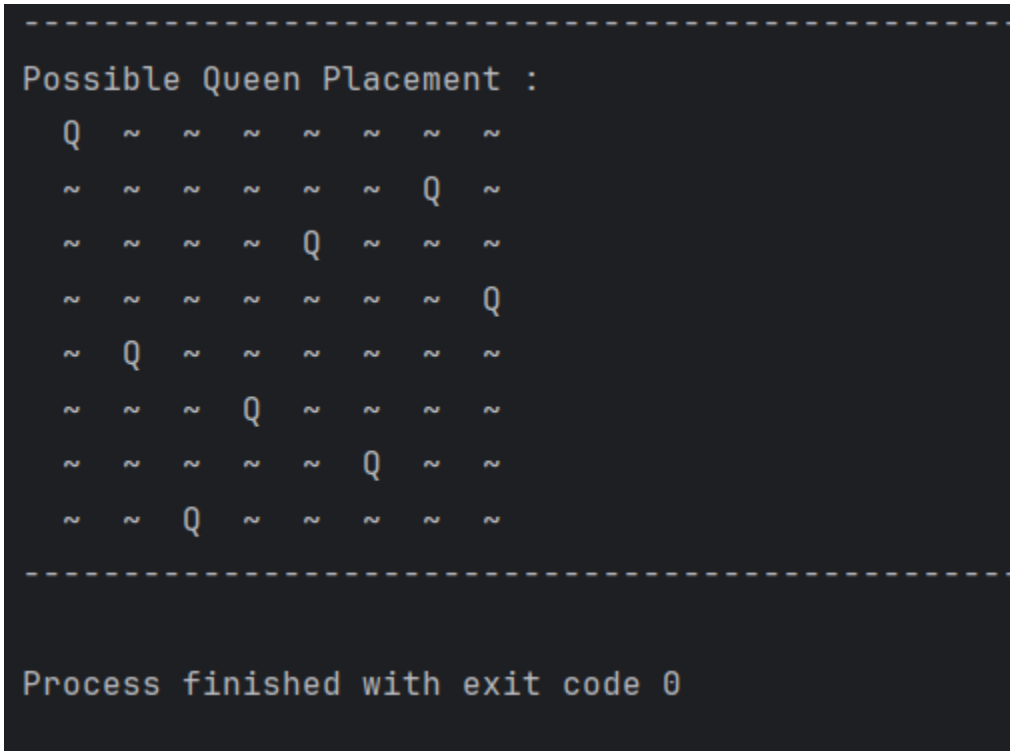
3. **Write a function bool SolveNQueen(int[,] board, int col) in Java that places N queens on an N x N chessboard so that no two queens attack each other using backtracking. Place N queens on the board such that no two queens can attack each other. Use a standard 8x8 chessboard.**

```
package m5_core_java_programming.day_20;

import java.util.Arrays;

/*
   Write a function bool SolveNQueen(int[,] board, int row, 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.
*/
public class Assignment_3 {
```

```java
   public static boolean SolveNQueen(int[][] board, int col) {
       if (col >= board.length) {
           return true;
       }

       for (int i = 0; i < board.length; i++) {
           if (isValid(board, i, col)) {
               board[i][col] = 81;
               if (SolveNQueen(board, col + 1)) {
                   return true;
               } else {
                   board[i][col] = 126;
               }
           }
       }

       return false;
   }

   public static boolean isValid(int[][] board, int x, int y) {
       int i, j;
       for (i = 0; i < y; i++)
           if (board[x][i] == 81)
               return false;

       for (i = x, j = y; i >= 0 && j >= 0; i--, j--)
           if (board[i][j] == 81)
               return false;


       for (i = x, j = y; j >= 0 && i < board.length; i++, j--)
           if (board[i][j] == 81)
               return false;

       return true;
   }

   public static void main(String[] args) {
       int[][] board = new int[8][8];

System.out.println("------------------------------------------------------------
--------");
       for (int i = 0; i < 8; i++) {
           Arrays.fill(board[i], 126);
       }
       SolveNQueen(board, 0);
       System.out.println("Possible Queen Placement : ");
       for (int i = 0; i < board.length; i++) {
           for (int j = 0; j < board[0].length; j++) {
```

```
                System.out.printf("%3c", (char) board[i][j]);
            }
            System.out.println();
        }

System.out.println("------------------------------------------------------------
--------");
    }
}
```

Output



Tools Used :
IntelliJ IDE
java version "1.8.0_411"
Java(TM) SE Runtime Environment (build 1.8.0_411-b09)
Java HotSpot(TM) Client VM (build 25.411-b09, mixed mode, sharing)