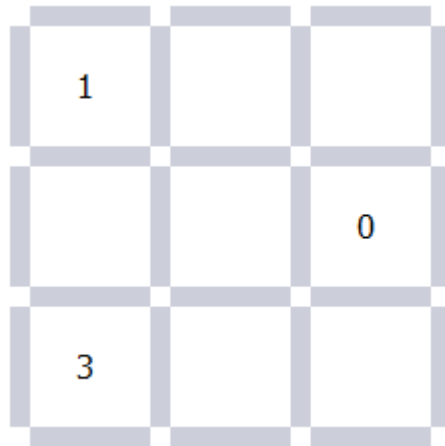


# CS 152 PROJECT REPORT

## Problem solved -

Loop-the-Loop, aka Slitherlink puzzle:

This solver solves a given  $n \times n$  Slitherlink puzzle (Puzzle was developed by Nikoli




The puzzle has a rectangular array of squares which have numbers ranging from 0 to 3 or nothing. The objective is to form a single loop with no loose ends by drawing lines vertically and horizontally. In addition, the number inside a square represents how many of its four sides are segments in the loop. A square with no number can have any number of lines around it. For a valid input only a unique loop is possible for this puzzle.

## Design of program


The program has two components –

### The backend solver description-

The solution to puzzle is obtained by designing a **simulation system** of objects as-

*block*  which contains a value acting as a constraint to the number of lines



around it, *the wires*  which surround the block and *junctions* where the wires meet. The simulation is similar to constraint propagation problem discussed in class. The entire square grid (in first figure at top) is represented as connections of blocks, junctions and wire. Each object of above type knows who are connected to them. The representation of loop is by state of wires which is true to show a line and false to show no line represented by it.

*The easy problems can be solved by logics – some of them being-*

1. Falsifying wires which are not part of solution loop.
2. A junction with two trues has others false
3. A junction with 3 false wires has other one also false, correspondingly for 3 wire and 2 wire junction.
4. A block with number 0 has all false wires around it.
5. A block with number 3 at corner has corner wires true and with 1 the wires false.

The easy problems have enough data that based on above logics they can be solved. *The hard problems require guessing and backtracking as follows -*

1. A tracer object keeps track of all moves.
2. If no logical move is possible a guess is made and the further moves are recorded.
3. If we find a logical error based on constraints we backtrack.
4. Backtracking is achieved by storing with guesses the counter move to each move done and then executing them till a suitable point. If this is a last point then the alternative guess there becomes a surety.

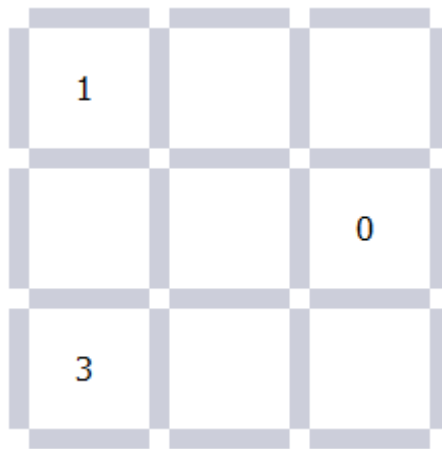
**Evaluation** (evaluation- how the events are processed) **technique** – is chained. Based on the constraints if a change is made the system goes to the changed point and checks if a further change is possible. This saves a lot of evaluation afterwards. To prevent the system into an infinite loop a strategy followed - first bring all change and move forward to changed points (the points are the objects).

## Front end interface design

The interface provides a platform to input to the backend of the system. There is a provision to select 'n' the size of grid. The various lines and dots seen apart from button and canvas are achieved by printing pictures in those places. The 'gui' is highly interactive, providing clear instructions over the game.

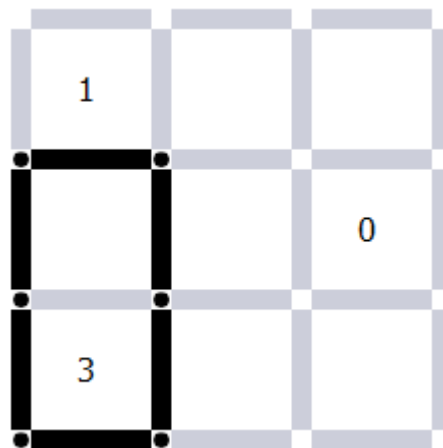
### Sample of input-

The input is achieved by clicking on the blocks to get the desired number on it. A



sample of the input looks as-

The ideal input in a loop the loop puzzle is a set of numbers which would give rise to a unique loop. But the system has been designed to solve any problem even when insufficient inputs are provided.



### The output of the system-

The solution output is a unique loop which can be drawn in the given canvas with constraints provided.

## Limitations and bugs of the program

For the various test cases we have exhausted the program was able to solve the given problem.

### LIMITATIONS

1. The guessing module is not designed to make efficient guesses, so the time required to solve problems (where insufficient input is provided) may take more time in certain cases eg. wherein only a single number is on board as input etc.
2. In the interface after a problem is solved one needs to click on reset to try another problem. Just giving another input and clicking solve causes mixing of dark lines creating confusion.
3. If more number of loops is possible for an input then only one loop is displayed. But essentially this is a faulty input. A valid input is one where only a unique loop is possible.

### Point of interests

1. Representing the entire board as linked objects and making them aware of the things they are connected to is nice simulation.
2. The implementation of backtracking by noting that each move has a counter move which nullifies it and thus storing those counter moves instead of actual moves and then executing them to achieve backtracking is nice concept.
3. To check for possibility of two or more loops, we set one of the loop wires to one color and the rest to another color. If a loop is detected with the other color they it signals that there are more loops than one in system.
4. Use of charge-propagation concept which sets the loose ends of the loop to same charge and if they are detected on adjacent junctions they are joined only when they are the only junctions left to be processed.