CSCI 630-02—Lab 1

Nikhil Pandey
np7803@rit.edu

March 07, 2019

Contents

1 Reading Input File

readers.grid_reader reads the puzzle from the given file name or a string. The input is plain text containing ASCII characters with lines terminating in single `\n` character at the end of each line. The file consists of:

- One line containing the following natural numbers seperated by single space
  - **R** is the number of rows
  - **C** is the number of columns
- $R \times 2 + 1$ lines describing the puzzle. Each line containing $C \times 2 + 1$ characters describing the cells of each row. Characters can be one of the following:

.	empty cell
0-9	prefilled value for the cell
<**space**>	no wall
-	horizontal wall
|	vertical wall

cell class is used to store value at each position. A 2-D list of size $R \times C$ is created to store the cells. A main queue is initialized with all the cells. A Depth-First-Search is performed until the queue is empty. During each iteration, one cell is taken from the queue. All the cells reachable from that cell are enumerated using Depth-First-Search. All the cells that were reachable lie in one region/room. *room* class is used to store the cells. They are also removed from the main queue. This process is repeated until the main queue is empty.

2 The Generic Solver

3 Dependency Injection

Also explain how you integrated the MRV heuristic and forward checking into your code.

If you wish to implement this check, you may do so, and incorporate it into your writeup (explain how it was done, and the effect on the empricial results over different puzzles) for up to an additional 10

3.1 Cell Selector

3.1.1 next_empty_cell

3.1.2 next_mrv_cell

3.1.3 next_human_like_mrv_cell

3.2 Move Selector

3.2.1 default_next_move

3.2.2 human_like_next_move

3.3 Validator

3.3.1 naive_validator

3.3.2 localized_validator

3.4 Pruner

3.4.1 default_pruner

3.4.2 value_pruner

3.4.3 forward_pruner

4 Solvers

5 Puzzles

6 Results

report on the empirical results for the different puzzles (including any beyond the ones given above, if you have tested any) is the improvement solely dependent on the size of the puzzle, or is something else going on? If you were to write a program to create Ripple Effect puzzles, how might you use this information?