Survey Paper on Maze Generation Algorithms for Puzzle Solving Games

Ms. Shivani H. Shah, Ms. Jagruti M. Mohite, Mr. Anoop G. Musale, Mr. Jay L. Borade

Abstract- A maze is a sort of puzzle having many branch passages. Some are closed and some lead to the end position. Here solver's goal is to move from a starting position to an end position i.e. a valid passage between these two positions has to be revealed. Maze generation is popular in entertainment domain. Mazes are used to solve the navigational problems which indeed brought the need of automation of maze generation. This paper gives an overview of three basic two-dimensional maze generation algorithms: a) Depth-First Search (DFS), b) Kruskal's Algorithm and c) Prim's Algorithm. These algorithms describe three conceptually different approaches for generating maze.

Index Terms- Depth-First Search, Graph algorithms, Grid, Kruskal's Algorithm, Maze generation algorithms, Maze creation algorithms, Prim's Algorithm.

1 INTRODUCTION

MAZE is a complicated system of paths in which people try to find their way through way of entertainment.. Mazes are of various shapes, sizes and difficulty. Maze has become very popular in fun stuff and an interesting domain from a mathematical point of view that is mazes can also be used to train brain.

Maze generation includes designing the layout of passages and walls within a maze. There are many different approaches for generating maze such as recursive division method, with various maze generation algorithms for building it. Maze generation is a difficult task since there has to manage several stuff that has to be controlled during generation. Some of them are as follows:

1. The maze should have a Start point and Exit Point.
2. The path to the exit must be reachable from the start.
3. Grids that are not included in the path must be reachable from start.
4. Maze should not consist of any cycle.
5. All the grids should be reachable.
6. The number of walls should be considerable so that the player will not reach the exit quickly.
7. Between any pair of grids there should only be one possible solution.

2 DEPTH FIRST SEARCH

Maze generation uses the randomized version of depth-first search algorithm. This algorithm is the simplest one and easy to implement using recursive method and stack [1]. Frequently it is implemented with a stack. Consider the space as a large grid of cells, each cell having four walls. Starts by selecting any random cell from the grid, the program then selects any random neighboring cell that has not yet been visited. The program removes the 'wall' between the two cells that are not connected and adds the new cell to a stack [2].

The program continues this process till the time, none of the unvisited neighbors being considered as a dead-end. Once it reach the dead end it backtracks through path till the time it reaches a cell with an unvisited neighbor and continuous the path generation by visiting this new, unvisited cell. This process continues until every cell has been visited, causing a program to backtrack all the way back to the start. This approach guarantees that all the cells are covered and the maze space is completely visited.

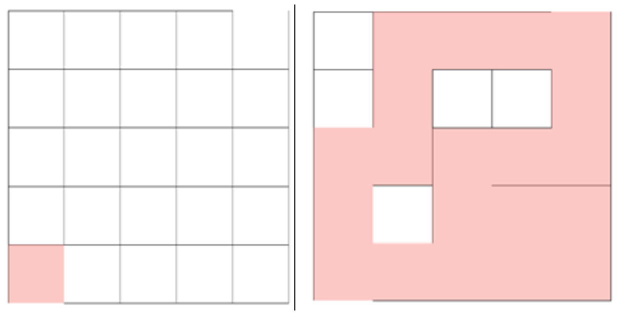
2.1 Algorithm

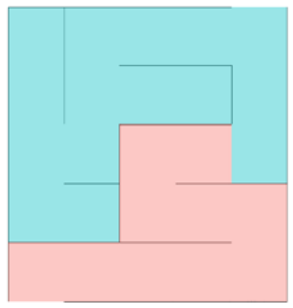
The algorithm proceeds as follows:

```
1. Mark the current cell as 'Visited'
2. If the current cell has any neighbors which have
   not been visited
      1. Choose randomly one of the unvisited
         neighbors.
      2. Add the current cell to the stack.
      3. Remove the wall between the current
         cell and the chosen cell.
      4. Make the chosen cell the current cell.
      5. Recursively call this function.
3. else
      1. Remove the last current cell from the
         stack.
      2. Backtrack to the previous execution of
         this function.
```

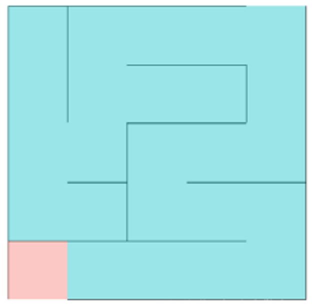
After execution of this algorithm, a 'perfect maze with no dead end' is generated and has a single solution [3].

(1) (2)





(3)



(4)

3 Kruskal's Algorithm

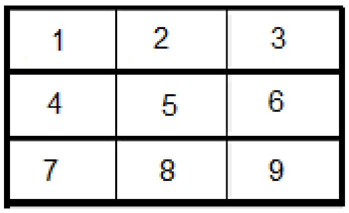
Kruskal's algorithm is a method for producing a minimal spanning tree from a weighted graph. Randomized Kruskal's algorithm is used for generating the maze form grids, it works by randomly selecting an edges from the graph, and adding them to the maze if they are connecting the disjoint trees. Usually it is done in an increasing distance but in case of maze all the cells of grids are at same distance [4]. Visually, it has the effect of growing the maze from many random points across the grid [3].

It works something like this:

- Create a list of all the available wall
- Select any random wall from random cell and remove it from the list.
- In case if there are is no way to move between the two cells then remove that particular wall which is blocking the way.
- Repeat it until there are no more cells left.

3.1 Example

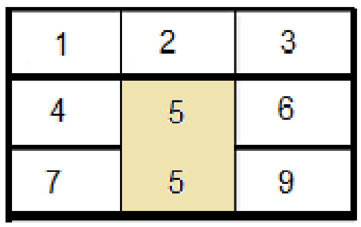
For this example we will use a simple 3×3 grid.
We've assigned each cell a number, indicating which set it belongs to.



1	2	3
4	5	6
7	8	9

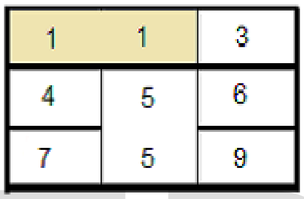
This algorithm is straightforward:

Starts by simply selecting an edge at random and join the cells it connects if they are not already connected by a path. We can know if the cells are already connected if they are in the same set. So, let's choose the edge between (2, 2) and (2, 3). The cells are in different sets, so we can join the two into a single set and connect the cells as:

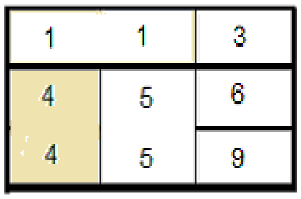


1	2	3
4	5	6
7	5	9

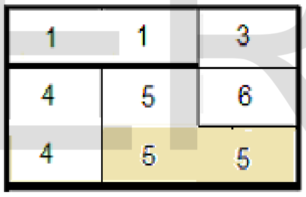
Let's do a few more passes of the algorithm, to get to the interesting part:



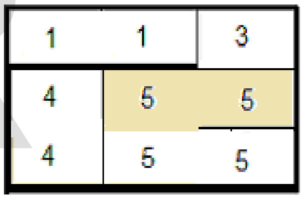
1	1	3
4	5	6
7	5	9



1	1	3
4	5	6
4	5	9

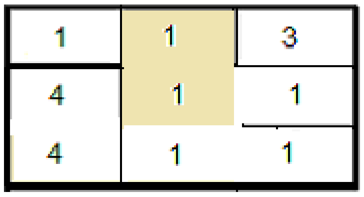


1	1	3
4	5	6
4	5	5



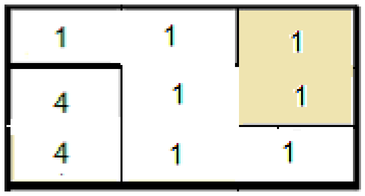
1	1	3
4	5	5
4	5	5

Here's where things start to move fast. Note what happens when the edge between (1, 2) and (2, 2) is pulled from the bag:

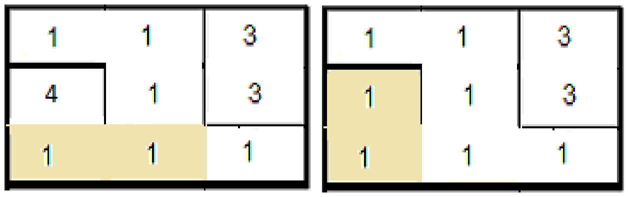


1	1	3
4	1	1
4	1	1

Now we will join these two trees, 1 and 5, into one set, 1, by implying that any cell in 1 is reachable from any other cell in 1. So we will try joining (1, 3) and (2, 3) now:



Now, consider the edges (1, 1)–(1, 2) and (1, 2)-(2, 2). None of these has been drawn from the bag yet .Let's see what would happen if one of them were? Well, in both cases, the cells on either side of the edge belong to the same set. Connecting the cells in either case would result in a cycle, so we will discard that edge and try again.



Kruskal's algorithm gets ended when there is no more edges are left to consider (which, in this case, is when there is only a single set left). And the result is a perfect maze!

4 Prim's Algorithm

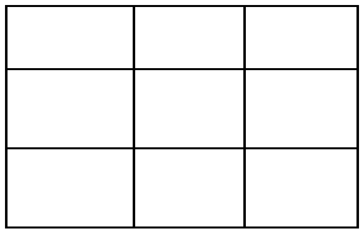
Prim's algorithm, starts by selecting any random cell from the grid and detects its neighboring cell and mark it as frontiers, one of which is randomly added to the maze. When the cell is added to the maze, the new frontiers are detected. The algorithm goes on until all the grid is converted into maze. That is it starts at one point and grow outwards from that point. So the sequence of the actions is in the following way [5].

4.1 Algorithm

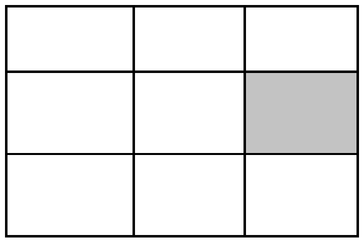
1. Start by selecting a random cell.
2. Mark that cell as "in" and al l its cell as "frontier" cell.
3. Choose random cell from "frontier". If it is not in the set, delete the wall between current cell and the neighbor.
4. Go to step one (but now the random cell is selected among cells in the "frontier").
5. Repeat until all the cells in the maze are covered and no mare cells are left in "frontier".

For implementing this algorithm [5] we will require two datasets. One for the "in" cell and another for the "frontier" cell. Let's start with a simple example by taking 3x3 a grid.

4.2 Example

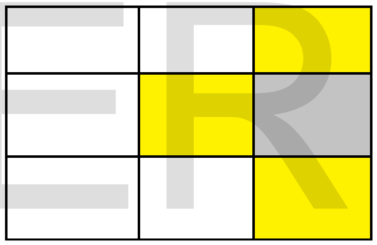


Now, we will select a point at random and add it to the "in".

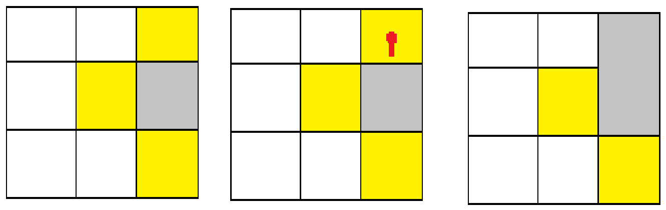


Grey color represents the "in" cell and Yellow color represents the "frontier" cell.

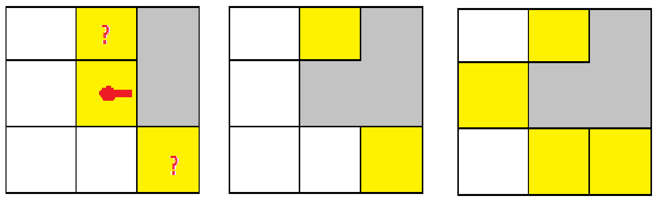
Just for sake of efficiency, let's call the set of all cells that are not present in the "in" data set, but are adjacent to a cell that are present in the stack, that is the "frontier". We'll color those cells with yellow.



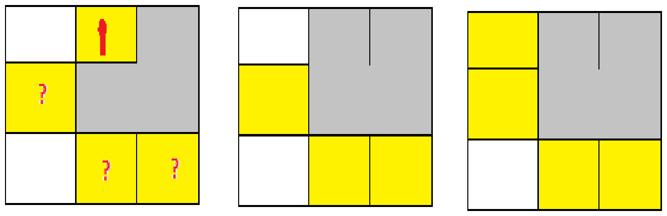
Now, we will select any one of these frontier cells at random and create a passage from that frontier cell to whichever adjacent cell that is already a present in the "in". Then we will mark the neighbors of the formerly front cell, as "frontier" cells.



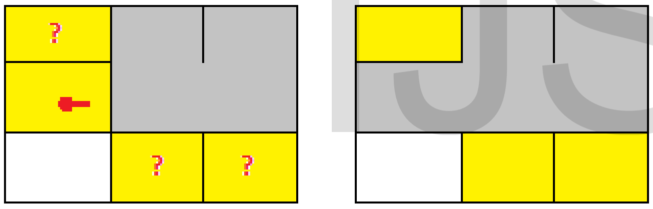
Flush and repeat the procedure:

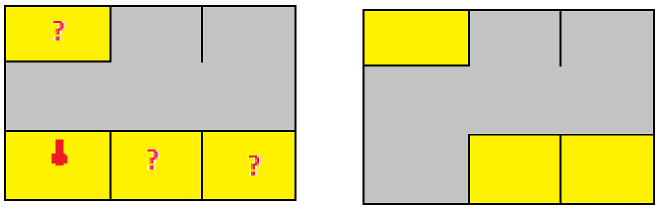


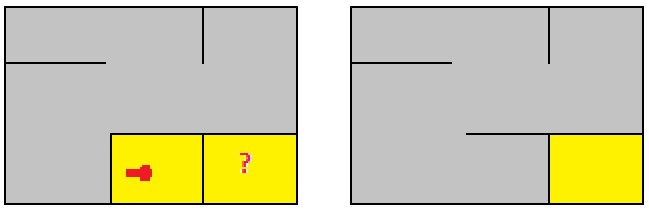
Now, here there is a little bit change. Let's see what happens if we randomly select the cell at (1, 0) that is the top middle. That is already adjacent to the two cells that are already "in" the stack. This algorithm resolves it by simply selecting one of the "in" neighbors at random. Prim's Algorithm doesn't care which neighbor is to be picked up, only care to be taken is that each front cell eventually be connected to a cell which is already present in the stack.[6]

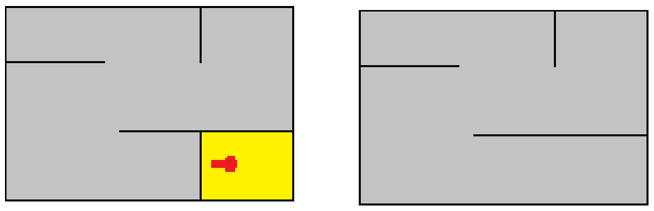


Keep on going it till the end:









This algorithm will terminate when there are no more front cell to select from, giving us the perfect maze.

5 CONCLUSION

Maze acts as an unknown environment that helps in tackling path finding, puzzle solving and path solving related problem. Due to its advantage of increasing the complexity of the maze as per required. Thus, maze acts as the basic milestone that is to be achieved for creating a self-automated moving robot [5]. These above mentioned algorithms have their own advantages and disadvantages that help in creating variety of mazes as per required. As DFS is one of the most basic algorithms to be implemented. It generates different mazes every time even if the number of rows and columns are the same. This is not the same in the case of Kruskal's algorithm and Prim's algorithm. But, Kruskal's algorithm and Prim's algorithm create a perfect maze every time and at input [7]. Whereas, DFS might create an imperfect maze i.e. the maze generated might not have any path to the goal.

REFERENCES

[1] Aliona Kozlova, and Joseph Alexander Brown, "Examination of Representational Expression in Maze Generation Algorithm," IEEE Conference on Computational Intelligence and Games, At Tainan, Taiwan, 19 August 2015.
[2] http://www.algostructure.com/specials/maze.php
[3] Nimesh Patel, Dr. K. M. Patel, "Survey on: Enhancement of Minimum Spanning Tree," Journal of Research and Application, Vol.5, No.1, January 2015, pp.06-10.
[4] Nirav J. Patel, Prof. Shweta Agrawat, "Survey paper on Different techniques for Minimum Spanning Tree," International Journal of Engineering Development and Research. ISSN: 2321-9939, pp. 22-25.
[5] Martin Foltin, Automated maze Generation and Human interaction, Masaryk University Faculty of Informatics, 2011 (Diploma Thesis).
[6] http://weblog.jamisbuck.org/2011/1/10/maze-generation-prim-s-algorithm
[7] https://en.wikipedia.org/wiki/Maze_generation_algorithm