MAZE GENERATION

Assignment 3: Generating Content

Submitted by
Govardhan Padmanabhan
(17069496/1 | WID170715)
as part of WID3009: Artificial Intelligence Game Programming

TABLE OF CONTENTS

SYNOPSIS

Unity is a game engine that allows developers to create 2D, 3D, VR/AR games, as well as simulations and other experiences. The prominent language used in unity is C#, an object-oriented scripting language.

Mazes are collections of passages/paths/hedges linked together, designed as a puzzle through which one can visit every other area from anywhere. The layout of the area and how they are connected defines the maze's characteristics.

The project aims at developing a procedural content generation (PCG) method to generate a maze using Unity. This project report focuses on maze generation using Prim's algorithm.

PROCEDURAL CONTENT GENERATION

The method used here is the Prim's algorithm. It is an Minimum Spanning Tree (MST) algorithm, and that it selects edges at random, thus allowing the creation of random mazes.

An overview of the Prim's algorithm can be overviewed as follow:

1) Start at any random node in the graph:
    - The starting node will be marked as reached
    - The other nodes will be unreached
2) Randomly find an edge 'e' that connects the current node to one of its neighboring nodes:
    - If the current/reached node is 'x' and the neighboring/unreached node is 'y',
    - The edge 'e' with minimum cost in the graph connects reached node 'x' to unreached node 'y'
3) Add the edge 'e' found in the previous step to the MST:
    - Mark the unreached node 'y' as reached
4) Repeat steps 2 & 3 until all nodes in the graph have been reached.

EVALUATION

For evaluating the method, the total number of cul-de-sacs (or dead ends) is taken into consideration.

Initially, a distance metric was considered, but this posed several problems.

- In this project, we are only generating a maze without any exit point(s). This caused a problem, as if we're to calculate distance, we need a starting point and an ending point.
- Even if we were to assign exits randomly, we are also using the starting point (to generate the maze, and also the player starting point) randomly. So I believe this wouldn't be a fair evaluation.
- If we assign the exit point as the last initialized maze cell, different algorithms go about generating the maze differently. So, in my humble opinion, this wouldn't be a fair evaluation.

For these reasons, the use of total number of dead-ends in the maze was considered instead. Creating cells that are blocked or closed off are generated solely by algorithms, which can be useful in evaluation. More cul-de-sacs in the maze, the better.

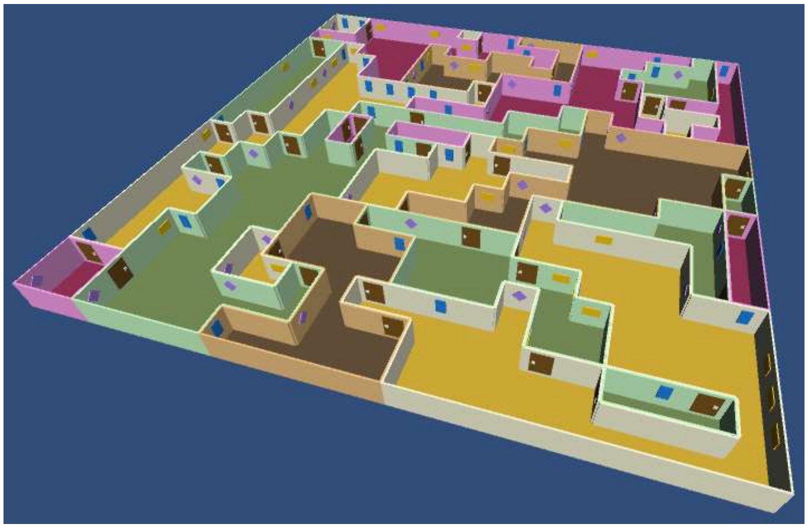
Scores

The following table represents the comparison of the number of dead ends generated by the implemented prim's algorithm and the recursive backtracking algorithm. Each algorithm was executed 5 times, and the average value (rounded to the nearest whole number) is represented here. For every iteration, the score is stored and overwritten on a csv file ("Evaluation Result.csv")

PCG Method	***Number of Cul-de-sacs***
Recursive Backtracking	14
Prim's Algorithm	31

Based on our limited testing, it can be inferred that the simple randomized Prim's algorithm implementation can produce a bit more difficult and competitive maze, compared to the recursive backtracking algorithm.

Generated Mazes



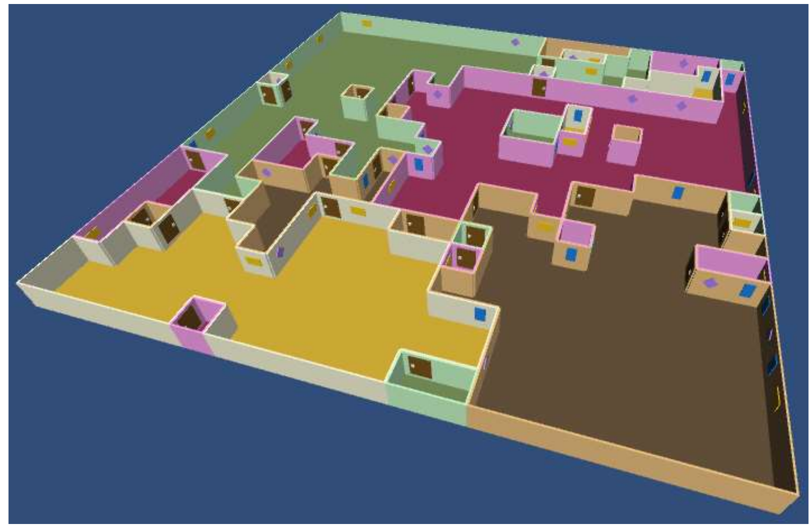


Fig. 1: Maze generated using Recursive Backtracking

Fig. 2: Maze generated using Prim's algorithm

Figures 1 and 2 are snapshots of mazes generated using recursive backtracking and Prim's algorithm respectively.

The backtracking algorithm is faster, more pleasant, and has very few cul-de-sacs. The prim's algorithm generates mazes that have wide and long passage, and far more cul-de-sacs.

While these mazes are generated using the mentioned algorithms, these do not accurately represent them. In this project, a simple randomized Prim's algorithm implementation is implemented. However, if a more optimized version is used (for example, using a binary heap data structure for the tree), the resulting maze could be more complex.

CONCLUSION

In this report, the implementation of a randomized Prim's algorithm is discussed. The algorithm's brief description and overview is discussed.

For evaluation, the number of dead ends generated were considered.

The guide provided for the maze generation does not provide evaluation metrics, nor does it provide any starting and ending points in the maze. And, since there weren't any available standards regarding the ending points (as discussed in the evaluation section), the distance metric was not considered.

However, using only the cul-de-sac count does not pain the whole picture. From the evaluation section, though the randomized Prim's algorithm generated more dead ends than the default backtracking algorithm, the maze generated tells a different story. From the images, the Prim's algorithm maze has more wide areas with more closed off cells, which in some cases, might not be ideal.

Thus, it isn't always recommended to evaluate PCGs solely using this method along. Here, due to specified reasons and other constraints and limitations, evaluation was done that way it was. However, a better evaluation score can be evaluated using an AI agent to play the content or even a human player.

The randomized Prim's algorithm implemented does generate more cul-de-sacs, but it also generates wide passages. For larger mazes, this algorithm will surely generate larger cul-de-sacs in size and quantity, making it more difficult.