Randomized Vertical Maze Cluster Generation

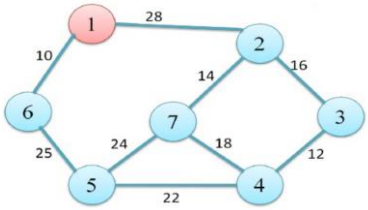
COMP 521
Thomas Wong
McGill University
April 2017

Introduction

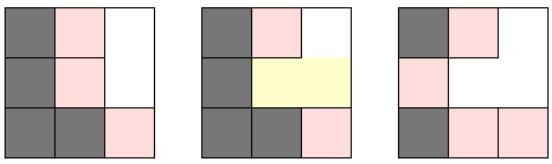
Procedural maze generation is a common topic within game research. It stems from the desire to create compelling, varying puzzles on every play through and give players a unique experience. In many cases, a maze is generated using a visual analogue to a solution to the minimum spanning tree problem; however, the resulting mazes tend to be flat, two-dimensional puzzles. With this project, my goal was to expand upon the logic given by a particular minimum spanning tree solution known as Prim's algorithm[1] to generate a randomized vertical maze cluster,[2] and build a game centered around navigating the cluster.

Background

The logic of Prim's algorithm for minimum spanning trees is very simple. Starting with a graph containing nodes and weighted edges, a random node is chosen as the starting node. This node is put into a set of used nodes. Among the nodes adjacent to



elements in our set of used nodes (frontier nodes), the node with the smallest weight is added into the set of used nodes. The addition of new nodes expands the set of frontier nodes and we continue to add frontier nodes into the set of used nodes until all the nodes are contained within the set of used nodes. When adapting this algorithm to maze generation, each node becomes a square unit of space. Additionally,



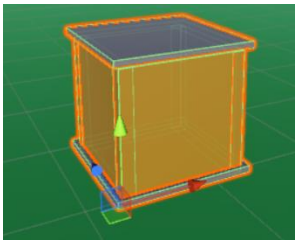
[1] Ijarcsms.com, I. (n.d.). Minimum Cost Spanning Tree Using Prim's Algorithm. Retrieved April 1, 2017, from https://www.academia.edu/4052642/Minimum_Cost_Spanning_Tree_Using_Prims_Algorithm

[2] a set of vertical mazes that contain some mazes expand upon the height of other mazes around it (i.e. the maze next to a maze ending at height = 3 would start at height = 3 and end at height = 6 expanding off the original maze height)

deciding which frontier node to expand into is randomized, and each node should remember which direction it came from. Expansion from a used node to a frontier node creates a path between them (the reason for storing direction) and once all frontier nodes have been put into the used nodes set, the resulting set of paths is a maze. In addition, this implementation of Prim's algorithm returns a maze restricted to a single plane, with no space between cells or variable cell sizes, and a 36% chance that any cell will be a dead end in the maze[3].

Methodology

Overview



Everything generated starts from a simple, empty cube (maze unit). Upon expansion to another unit, the wall between them is removed to create a path. For any unit, the four walls, the ceiling, and the floor could be removed, (this is important when needing to expand vertically). I started generating a standard maze using these maze units and modifying the algorithm to generate a random maze shape with fewer dead ends. I then identified and implemented my method of vertical maze expansion to simulate a building. Finally, I placed buildings of variable height and distance from each other to simulate a city (Vertical Maze Cluster).

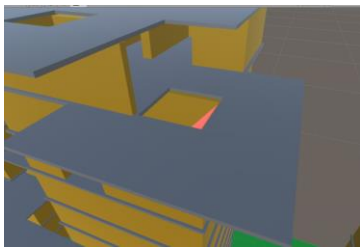
Randomizing shape and removing dead ends

The algorithm I implemented to create a single floor in a maze building is as the one described above but contains a couple modifications. Firstly, to create unique maze shapes that don't necessarily cover the full square of space, I abandoned the requirement to expand into an empty frontier node. Instead, I stored a list of active used

nodes, and for each node in the list the algorithm had three chances to generate a direction that pointed to a frontier node. If the node successfully generated a valid direction, the node would stay in the list, the frontier node would be added to the list, and the process would repeat using this new used node. This created the possibility that a node with adjacent frontier nodes could potentially not expand into any of them, allowing for variable, non-uniform maze shape. If the node generated a direction which pointed to an already occupied node, the wall, if it existed between these two nodes, would be eliminated. This modification provided the solution to the high number of dead ends as any dead-end node would no longer have restricted expansion due to adjacent occupied nodes. If the three directions generated did not result in the addition of a frontier node, then the generating node would be removed from the active list. Once there were no more nodes in the active list, the algorithm would remove all walls from cells with only one wall remaining.

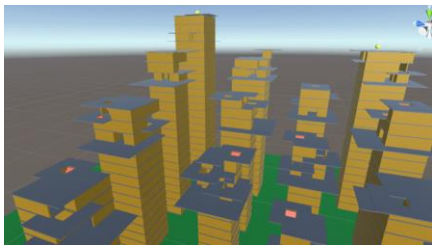
Expanding with vertical components

In my maze algorithm, the starting node is arbitrary and a single floor maze could start at any cell. Similarly, the final frontier cell added to the maze, given the modifications stated above, would always be either a dead end or a braid corner. This makes the vertical expansion of the maze very simple. A set of stairs would be placed on the final frontier cell; added and oriented according to the corresponding direction of the cell. The maze on the next floor would start with two cells: a cell without a floor right on top of the stair cell, and an initial expansion cell in the direction of the stairs. If the expansion cell was within the bounds of the maze, then the maze generation for this

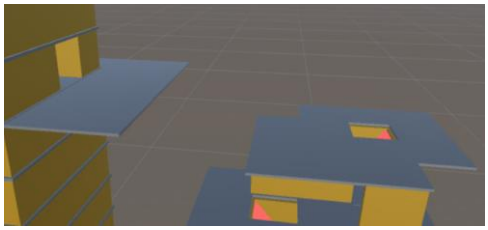


new floor would proceed exactly as stated earlier. If the expansion cell was not within the bounds of the maze, meaning that the stairs led to a cell that exceeded the limitations of the building, a platform would be created around the stair opening and the maze would start at a side opening adjacent to the exiting stairs. Simply put, I expanded my mazes vertically by treating each iteration of a maze as a layer, with a layer starting where the maze under it ended.

City building



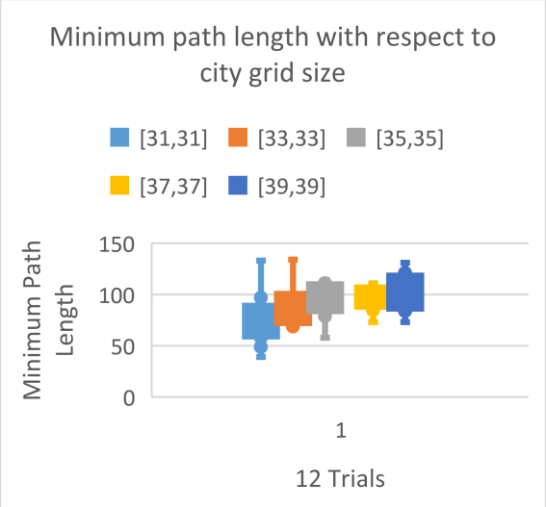
To generate a vertical maze cluster (VMC), I used another analogue to Prim's algorithm. I stored a list of active used buildings starting from a randomly-placed building occupying a square on a grid floor. For each building in the list, the algorithm had three chances to generate a combination of direction, distance, and size that point to an unoccupied square within the grid boundaries. If a valid combination is found, a new building of the corresponding size is placed on the grid that starts at a height one above the ending height of its source building. If a valid building placement isn't found, then the current building is removed from the active list. By including the extra randomized variable of distance, each cell (building) could have empty space around it, creating the illusion of a city. A building that stemming from a source building has a platform leading to the maze opening in the



direction of the source building so that a player can easily travel to and enter this new building. The result is a set of buildings that are all reachable by the player.

Results

I developed a game around navigating the generated VMC in which players have to collect orbs at the tops of multiple buildings in the maze as lava rises from the floor. This created multiple destinations to reach and an urgency to reach them. With this in mind, I generated the Total Minimum Path Length based on the sum of the minimum paths required to obtain all the orbs. I counted vertical traversal of any particular building only once, as players could easily jump from the top of a higher building to the top of a lower one adjacent to it.



Conclusion

Through implementing Prim's algorithm in many distinct ways, I successfully constructed a complex VMC and a playable game centered around it. Interestingly enough, as the minimum path distance increased, completing the goal of the game wasn't necessarily harder. Because players could jump the height of roughly two levels in the maze, I found that the more I played the game, the more I would bypass some of the maze floors by hopping to a platform created by a higher level in the building. Factoring this into calculating the shortest path length could be a potential lane of improvement with adjusting the lava travel speed. Additionally, being able to fuse two buildings to create a single braided maze building could greatly enhance the complexity of the maze.

References

1. Buck, J. (2011, January 10). Maze Generation: Prim's Algorithm [Web log post]. Retrieved April 5, 2017, from http://weblog.jamisbuck.org/2011/1/10/maze-generation-prim-s-algorithm
2. Ijarcsms.com, I. (n.d.). Minimum Cost Spanning Tree Using Prim's Algorithm. Retrieved April 1, 2017, from https://www.academia.edu/4052642/Minimum_Cost_Spanning_Tree_Using_Prims_Algorithm
3. F. H. (2016, July 26). Makalah IF2120 Matematika Diskrit – Sem. I Tahun 2012/2013 Implementation of Prim's and Kruskal's Algorithms' on Maze Generation. Retrieved April 3, 2017, from https://www.researchgate.net/publication/305652371_Implementation_of_Prim%27s_and_Kruskal%27s_Algorithms%27_on_Maze_Generation