Maze Analysis

Michael Flood

After looking over the task and the disco floor that was supplied as the base program, I started off by researching different maze generation algorithms to figure out which one I wanted to base my maze off of. I ended up modifying Prim’s Algorithm because it seemed like one of the more straight forward options to me. Looking back on it afterwards, Depth-First Search seems like a quicker implementation but Prim’s clicked with me quicker in terms of how I wanted to implement it.

The way I handled Prim’s algorithm was by first starting off and assigning a random weight to each cell between 0-99. The other way it could have been done would be to get a random direction whenever the paths branch off, but I like the idea of generating all the numbers at the beginning rather than doing it during each step. Each cell also has a status attached to it which is used to determine whether it is a path, wall, hasn’t been checked, or if it is a frontier cell. Once a cell is a wall or a path then it is final. So before I start the loop I set the outer perimeter of cells to walls and marked off one of those cells at random as the exit point then I move onto the loop.

Throughout my loop the only thing that I care about is the frontier cells which are the adjacent cells of the most recently made path. So it starts at the exit and picks the frontier cell with the lowest weight and stretches the path in that that direction, this path continues winding through until the most recently made marked cell has no frontier cells. Then it goes through all the left over frontier cells and finds the lowest to start snaking through again. This process continues until there are no frontier cells left. The generator determines whether to turn the frontier cell into a path or wall by judging the amount of paths adjacent to the cell currently being checked. If there are two or more paths then it creates a wall, also if there are zero paths then it also creates a wall as well to prevent small pockets of unreachable pathways.

The most difficult part for sure was debugging the maze while I was adding steps into the generation process. While building it, I made it so the color of each cell represented whether it was a wall, floor, frontier, or hasn’t been checked. It made the process a bit easier so that when it was generated, I would be able to see if my loop was accessing each cell or if it was only getting part way through before quitting. I avoided setting up the maze generation in the update function which would have allowed me to watch it generate step by step because that would require the loops logic to change a bit to transfer it back and forth which could have led to more issues. One of the last things that I was having trouble with was my maze creating small pockets of unreachable paths which was the last problem I solved before finishing up the maze generation section of the project and moving onto part two. This ended up being something very small that was solved by checking whether the frontier cell had zero connected paths and to make it a wall if it did.

If I were to estimate my time spent on the project as a whole, I would say about half went to the maze generation algorithm while the other half was split up among adding the switching white panels, keys to the locked exit, and the ability to lower walls. When I was thinking about what to do with the last two features I wanted to do something that made it a bit more “gamey” which is why I added the locked exit that requires four keys and the ability to lower a wall pieces for every ten cubes you walk across. These two features added a bit of variation making it more than just a simple maze and they were fun to implement. Overall this was a fun and interesting project that I am happy to have completed, also having a working maze generator is a plus if I ever need to do something similar in the future.