Development Documentation

Maze algorithm, Unity development

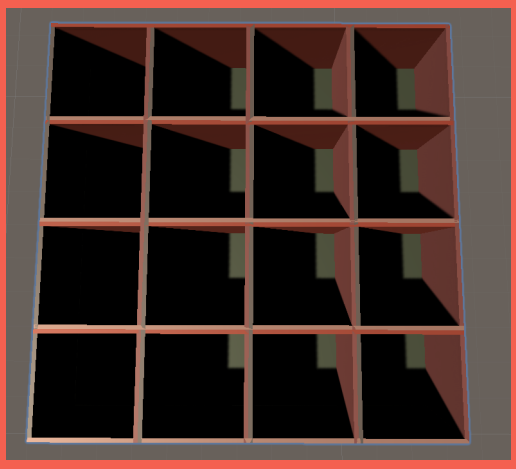
The Start

After research a few algorithms, I decided to go with the Hunt and Kill algorithm. The reason why was because the solution proposed by the algorithm proves to be pretty straight forward and logical.

It's divided into two parts, as the name suggests, but after dive into the algorithm, I made my wall and my floor, also a grid to comport everything.

The Grid

The first step was creating the grid that would serve as a base to the maze, and the cells. Each cell has four walls, in the four directions (Up, Down, Left, and Right), making it possible to remove only the one needed to form the path. By that point, the "Maze" would look like the image on the right.

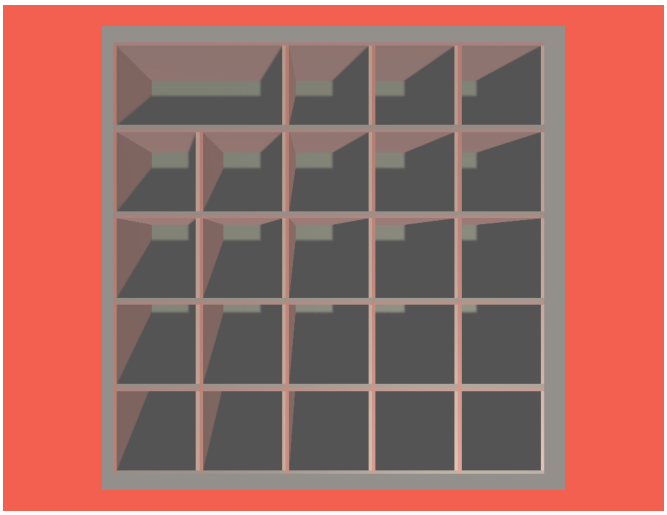


Random Walk

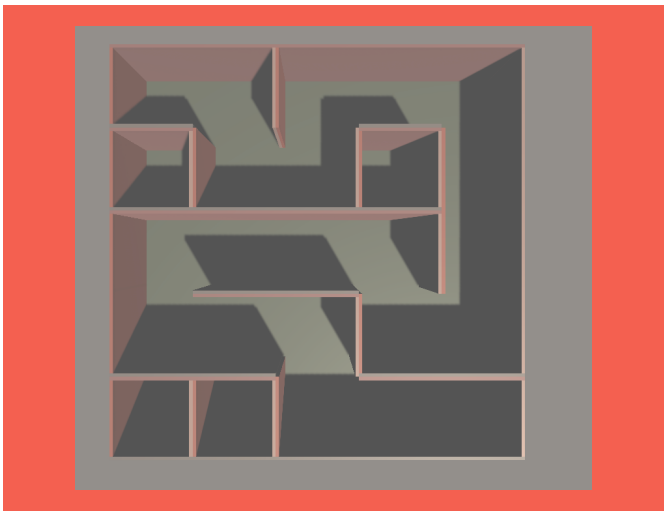
To generate different mazes the algorithm should work randomly, choosing different positions to advance each time. Each cell has four options, that's why the range of Direction is 0 to 4. When in a cell, before trying to carve a path, I had to check if the current cell has any neighbour, and it's not the last one from the Row or Column, otherwise, I would try to advance for an empty space.

The second check is when the algorithm starts the kill phase, it will check if the chosen direction is available, and the next cell on that direction has never been visited before. If that's is the case will destroy the chosen direction wall of the current cell and the opposite one from the next, creating a path between the two rooms.

Kill Phase



Carving the first passage, in this case, could have chosen to go Right or to the Bottom.



Repeat the first steps until it finds a dead end.

The While Loop

Every time a path is carved advances to the next cell, in the respective direction, till a wall is destroyed. The loop will keep going infinite if you attempt to go through all the maze because when reaches the dead-end will not comply with any of the if statements, never updating the Collum or Row. Because of that, the loop has to check if it's an unvisited neighbor in advance. If this neighbor exists, so for sure a path can be carved, if not a dead-end was reached, and we need to advance towards the second phase of the algorithm.

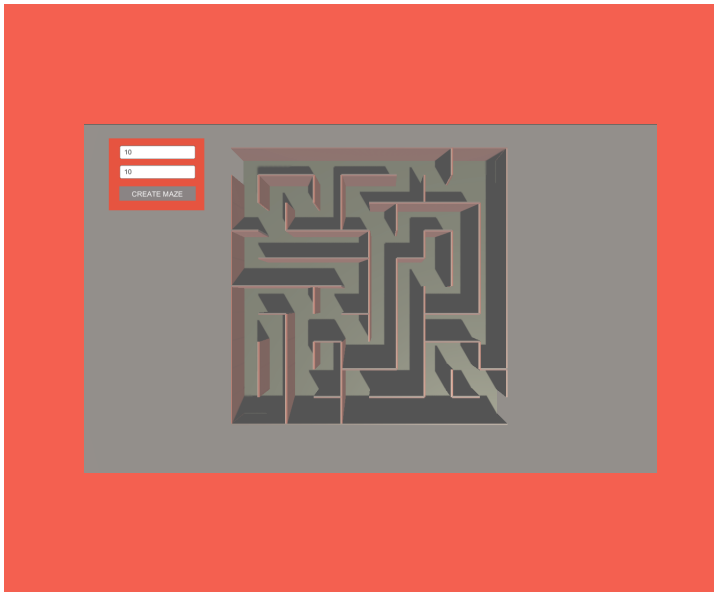
Let's go Hunt!

When a dead-end is found we need to loop through the entire maze. We will hunt for the next cell that has not been visited before, and connect it with one that has been to carve a path. So this time our check has to be inverted, we have to be sure - before starting to looking for a random direction to go - that the current cell has a visited neighbor already.

After carving this connection with the current path will return to the kill phase, the new start position been the unvisited cell we found. Repeating the entire process again. If the hunt phase does not find any unvisited cell that's mean the maze is done, and it will inform the main loop stoping it.

Review

That was the solution I applied to make a perfect maze. Besides that my program also accepts values from the user to change the width and the height, plus the camera adjustment to fit the screen. The right image shows the final result.



Primarily resource

Hunt and Kill algorithm:
https://weblog.jamisbuck.org/2011/1/24/maze-generation-hunt-and-kill-algorithm