Description

This maze generator visualizes the maze using a 2D Tilemap. The generation is made in such a way that you can see the algorithm at work, adding to the maze one tile at a time. You can select what algorithm to use and you can set the size of the maze you want to generate.

If you want to view the maze, you can use WASD or arrow keys to move the camera, and you can zoom in/out using your scroll wheel. When you click on a tile within the maze, you can see a path that goes from the bottom left of the maze to where you clicked.

Currently, there are 2 implemented algorithms: Prim and Wilson's algorithm.

Prim:

I used a modified version of the Iterative Randomized Prim's Algorithm. I keep a list of tiles that is adjacent to the part of the maze that is already generated, and then select a random tile from that list to connect to the maze. This is repeated until every tile has been added to the maze. This algorithm tends to generate many shorter branches. As it only iterates on each tile once, it runs in O(n) time, with n being the total number of tiles in the maze.

Wilson:

This algorithm works by doing a random walk until a tile that is already part of the maze is reached. If the path from the random walk runs into itself, it erases the loop (every tile between the tile it ran into and the last tile of the random walk). This prevents the walk from locking itself into a corner of a maze. Due to its random nature, there is no bias towards short or long paths. However, its randomness also means that some tiles will be iterated on multiple tiles. As it needs to find a tile already part of the maze, especially the first random walk can be particularly inefficient, especially in larger mazes. This means that while it will terminate eventually, it could theoretically take an infinite amount of time.

How it works

The maze itself is represented by a 1D array of tiles. Each tile is a struct consisting of: its position in the Tilemap, its index in the array, whether it has been visited yet, and the index of the previous tile in the maze, as well as a list of walls on that tile. That list of walls is represented by a custom data structure, which stores all the walls in a single integer (unfortunately bit operations don't work on bytes) by setting a certain bit. This is probably much more complex than just keeping 4 bools, but for some reason I decided I really wanted to use bitwise operations, so I stuck with it.

I'm using a one-dimensional array to store the maze because it is slightly faster than multi-dimensional arrays and, more importantly, because I kept in mind that if I wanted to switch to Burst to speed up the generation process, then I would need to use a 1D Native Array, which is easier to refactor to when I already use a managed 1D array.

The algorithms are set up so that running them will only do a single step at a time, so that you can see it work step by step. Not generating the entire maze at once also has the benefit of not freezing the program when generating very large mazes, though you can still skip to completion which tends to freeze for less than a second on 250x250 grids.

The generator itself is created and held by the MazeGrid, which is a MonoBehaviour and acts as the interface through which the generator/maze is accessed. When the generator runs, it returns a list of all the tiles that were changed during the iteration. The MazeGrid then feeds that list into the TileBuilder, which then makes sure the correct sprite and rotation is applied to the changed tile in the Tilemap.

I also have a few more classes, which I'll quickly summarise their function of below:

MazeSetupUI and MazeIterationUI both take input from the UI and send it the MazeGrid.

ValueDisplay simply sets the text of a text element to the value that it was given, used with the sliders that set the maze size.

CameraMotor2D is a script I was working on previously in another project to move the camera around. It is functional, but still incomplete, and I have many features I still plan to add.

PathDrawer simply fetches the path from the start of the maze to the hovered tile, and inputs the path in a line rendered.