Background- Previous attempts at dungeon generation and their drawbacks.

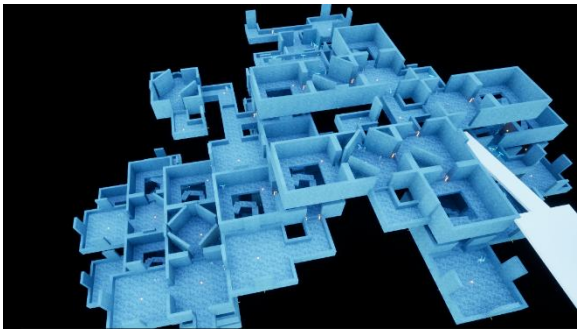


I started learning how to Blueprint in Unreal Engine over Winter Break. I had gotten into the STEC Overgrown class and wanted to do Systems but had 0 experience with Unreal Engine or coding outside of Processing. To study, I made a practice game with two of my friends called Maw.

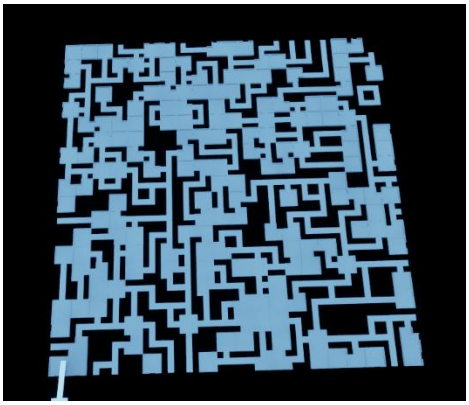
In Maw, the player is the last remaining cultist of the Great Devouring Maw. Unfortunately, the rest of their cult was recently devoured by the Great Devouring Maw. As the weakest of their cult and least worthy of being eaten, it is now the player's responsibility alone to continue to feed the Eldritch God. The player descends into randomly generated dungeons to kill monsters. The player brings their bodies back to their temple, prepares them as meals, and feeds them to the Maw. The player is then rewarded with "blood points" they may use to upgrade certain skills or weapons. My friends and I would like to continue working on Maw over the Summer.

I hit a wall, though, because random generation was just not cutting it. Dungeons are unsatisfying and aimless. I had attempted two versions that both have drawbacks.

The first version was taken from This Tutorial Series by REE Animation. It is multi-floored and the dungeon has branching paths that form occasional loops, which I like, but rooms must be a fixed size with a fixed doorway width, and it is extremely difficult to control where specific enemies spawn or create different feeling "biomes" in the dungeon.



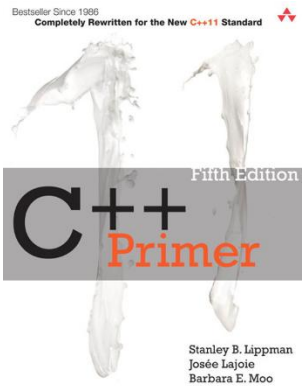
The second version used Wave Function Collapse. It was less successful but taught me more. While it created a variety of larger room shapes without the need for a unified hallway size between tiles, the hard square map cutoff disqualifies it as a viable option. I would still like to incorporate Wave Function collapse into a dungeon eventually, but to fill out rooms rather than generate its structure.



The TinyKeep Method and its Boons:

The Tiny Keep method is a popular dungeon generation method first posted to Reddit in 2013 by the developer of a game called Tiny Keep. This method uses Delauney Triangulation and Minimum Spanning Trees to generate a dungeon with a corresponding graph that keeps track of rooms and their neighbors. This graph allows for more intentional generation because it can be tweaked and rewritten to follow certain rules. For example, a boss room cannot connect to the player's start room. It must be the room with the greatest distance from the player's start. They key to a corridor cannot spawn in a room behind that corridor, etc.

Luckily for me, Github contributor SyMbolzz has recently posted a version of this generator for Unreal Engine: https://github.com/SyMbolzz/Triangulation-Dungeon-Generator-UE5



This is written in C++. I have never written anything in C++ before, so I am extremely grateful that I started reading the C++ Primer on the plane last weekend. I got to page 120, which is about a tenth of the way through it, and just enough for me to know that I don't know the difference between a const pointer and a pointer const.

My next steps are as follows:

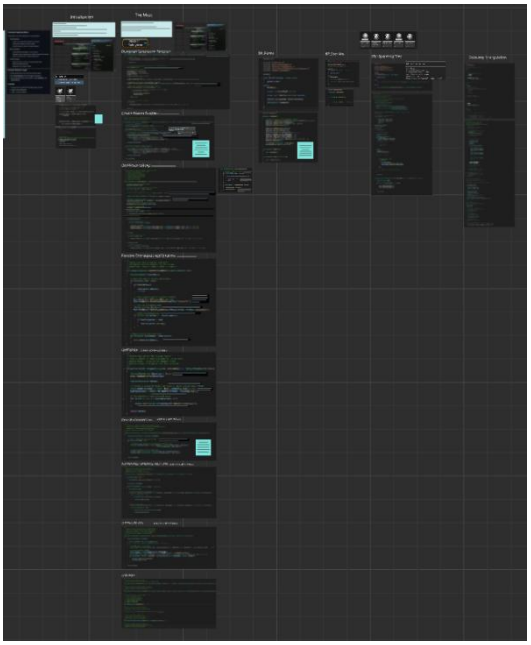
Open the project in engine. (this week)

Annotate and understand the C++. (this week)

Rewrite the C++ as Blueprints. (this week)

Once working as intended, figure out how to replace the cubes spawned by the generator with correctly scaled instanced static meshes. (future)

Figure out how to add room-type graph logic to the generator. (future)



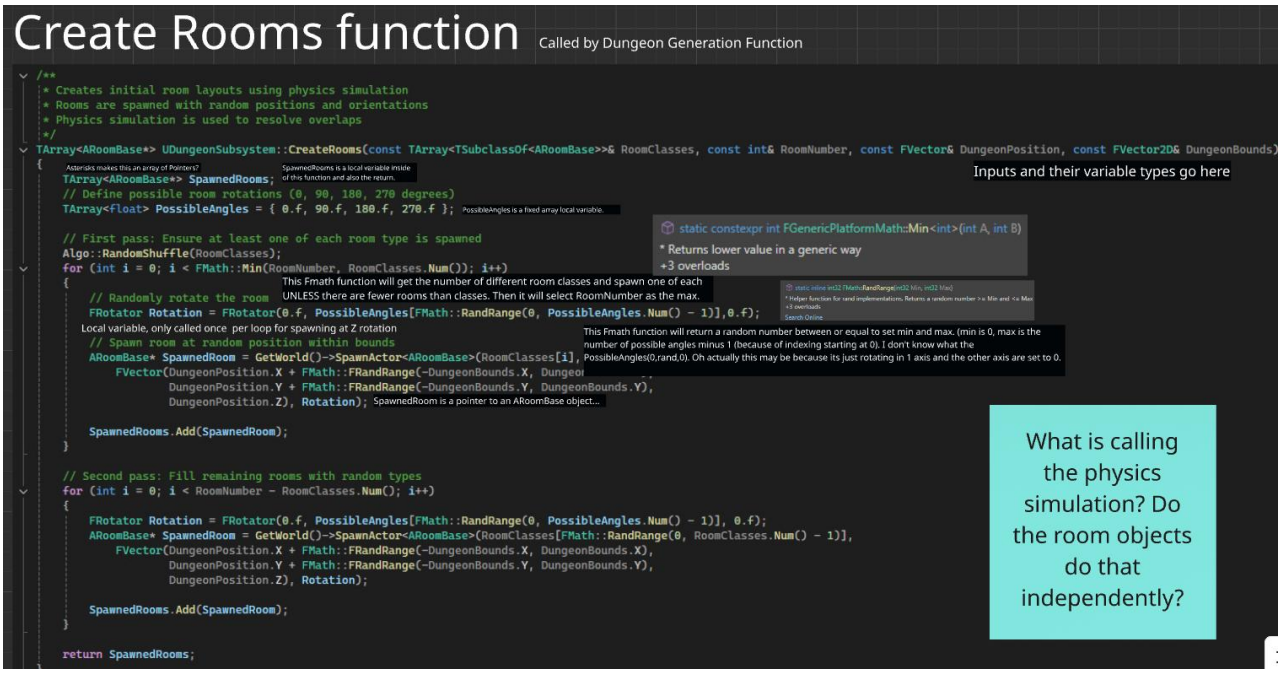
Opening the Project in Unreal Engine:

This was surprisingly difficult. I followed this tutorial and downloaded Microsoft Visual Studio extensions until it worked.

https://www.youtube.com/watch?v=6mFhtvdYQWI

Annotating and Understanding the C++

This has been one of my favorite parts. I was very excited to realize that I understood what was written, and as I annotated more, I wrote less, since I was seeing the same things over and over and becoming familiar with them. I store all my annotations on a Miro board to reference them later. Here are some of them.



Remaining questions after annotations:

I am curious about how to transform the basic cubes this generation spawns into real rooms with walls and floors that aren't awkwardly stretched.