Senior Project

By Charles Mogan

Intro

My goal with this project was to make a top down shooter in which the player traverses a maze-like environment and defeat enemies. The environment in the game consists of a series of random rooms filled with various enemies that the player must defeat. At the end of each floor there is a boss that the player must defeat before they can advance to the next floor. Through this project I explored the basics of the unity game engine, c# and procedural generation, all subjects I had no experience with at all up until now. I decided to focus on mechanics and ignore things like textures, animations, and audio. This is because I can't make these things myself and would likely be handled by other people on any type of commercial project.

I decided to use Unity for my project because it is widely considered to be the easiest to use of the "real" game engines. Once I decided on the engine my choices for programing language were narrowed to c# and javascript, because those are the two languages that Unity supports. I decided to use C# because that is what most of the tutorials I found were written in, and because my limited experience with javascript was not very positive.

The hardest part of the project was just forcing myself to work on it. Working on a project with abstract far off deadlines and no oversight is always difficult, but this project was much larger than others I have worked on so the temptation to fall into the "andy class trap" was everpresent.

Another difficulty, somewhat related to the first, was getting oriented with c# and the unity environment. At the

beginning there were not any short term goals I could achieve in a coding session because I didn’t have a good idea of where I was going or how to use the development environment. Once I knew enough that I was able to break things up into small chunks then itemize daily tasks from those chunks things went much more smoothly.

Movement

I tried several different basic movement systems. The first was applying a force on the player through Unity’s physics system. This resulted in the player accelerating when a button was pressed, which gave the player a sort of floaty feel. This was acceptable for the player, but had odd results when applied to the enemies. An enemy that was supposed to go towards a player would orbit the player instead. The second movement system I tried was directly moving the player by setting their position each frame based on player input. This meant that the player now moved at a constant velocity. Enemies also move towards the player directly instead of orbiting. The major problem with this movement system is that it breaks anything that relies on measuring the character's velocity. This is because the character does not move so much as teleport very small distances, so their measured velocity is always 0.

The biggest remaining problem with the movement system in my game is that I'm using the collision system to stop the player and enemy from phasing through walls. This works fine, but is jittery and not very realistic. If I wanted to make the movement more legitimate I would switch to Unity’s built in movement system. I didn’t use it for my project because it would have required dynamically creating and modifying and combining navigation meshes as blocks in the map are created and destroyed. In the future however this would allow me to make somewhat more advanced enemy behaviors.

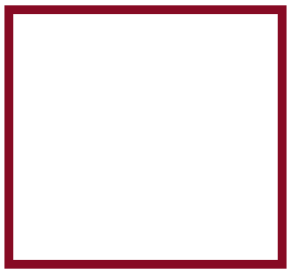
Shooting System

Creating the bullets that the player fires was not very difficult. I simply instantiate a basic sphere gameobject at the player's position with a tiny script for dealing damage attached to it, and give it an initial velocity.

Physically The laser’s physical structure is just made of a basic unity cylinder object, elongated, rotated and made translucent. In order to make using the laser risker the player is paralyzed for a short time when they use it. In order to do this I use the unity API’s StartCoroutine() function to disable the players movement script while the laser charges and fires. StartCoroutine() does not start up a new thread, which is is important because game objects can not be instantiated outside of the main thread. Projectiles do damage on contact, so to have the laser do consistent damage over time it is actually composed of 13 sub-lasers that are instantiated over the second the laser is firing.

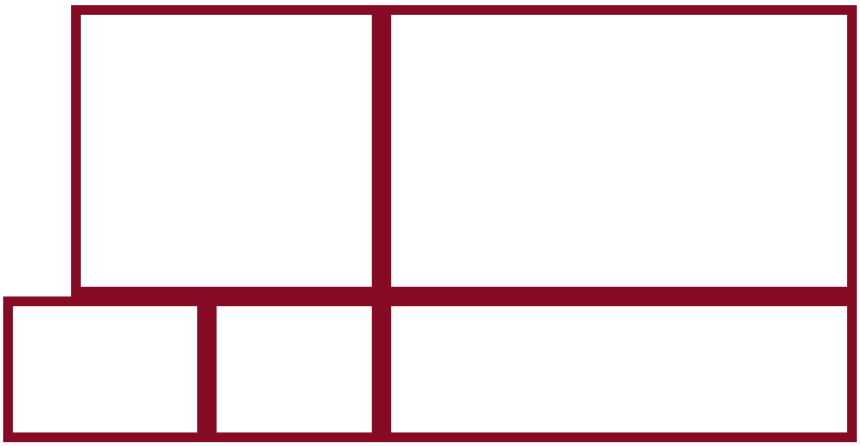
Level Generation

The level generation starts by generating the outline of a fixed size square room. This is the room that the player starts in. it is the only one without enemies and other rooms and will tend to be near the center of the worldspace.



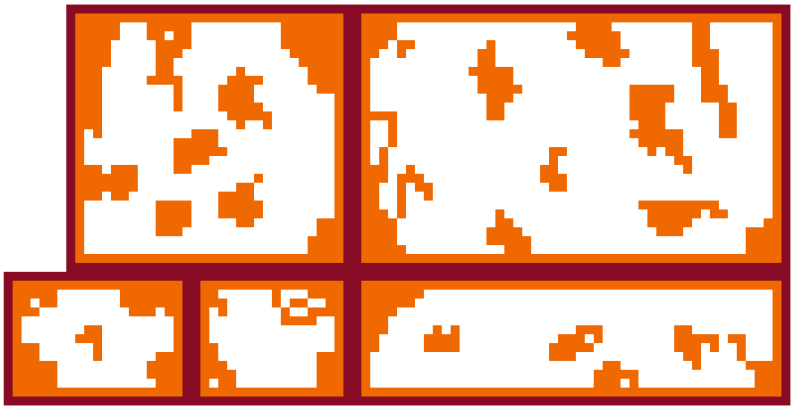
After the first room the outer walls of all other rooms are proposed so each subsequent room shares a wall random with a random existing room. The room's other dimension is a random size so that there is some more diversity in the rooms being generated.

The proposed room has its location checked against the location of all existing rooms, if a collision is found a new random room is proposed, if no collisions are found the outer walls of that room are generated. If a collision is detected then a new room is proposed.

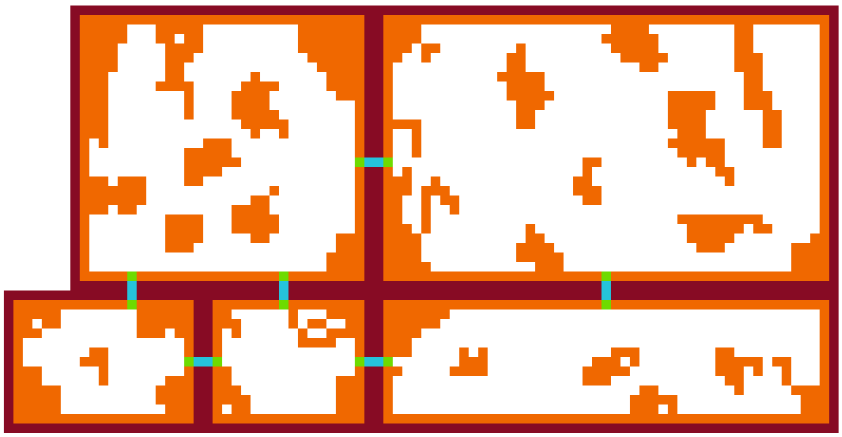


Once the desired number of rooms are generated the rooms are filled in parallel using a cellular automata algorithm that seeds the room with walls and then clumps them together. I tried many different rule sets for the cellular automata to see what types of interesting results I could get. I decided to keep the rooms relatively open because enemies do not have pathing. While

the ruleset I decided on works well for the 50X50 to 100X100 rooms that are currently in the game, adjustments to the ruleset would be needed for smaller or larger rooms. If I continue this project one change I plan on making is adding more room types with different sizes, classes, and rulesets.



After the rooms are filled, doors are added between the rooms. Doors are placed at the center of the contact points between any two rooms. A room can contact multiple other rooms on the same side, a door is made for each. After that the area around each door is cleared of walls and the trigger to activate the doors is added.



I originally wanted the wall segments to be composed of many more, much smaller cubes. This would have given the rooms a much more organic look. It turns out that even with culling, and the game objects just being cubes there is a limit to how many game objects can exist and have the game still run smoothly. Using the script

```
for(int x = 0; x < howManyCubes; x++){
     for(int z = 0; z < howManyCubes; z++){
     GameObject myCube = Instantiate(cube, new Vector3(x,-20,z),
Quaternion.identity);
     }
}
```

From this I was able to determine that performance really started to suffer between 40k and 250k cubes depending on settings and the system the game is running on. Because of this the plan for 5000X5000 cube rooms was scrapped in favor of blocky 100X100 rooms.

Things I would/should have done differently

Parallelizing the level generation ended up being a giant pain. At first it seemed it would be easy, I found out that C# had a Parallel class that had a Parallel.For() method. The first problem I encountered was that Unity's API is not thread safe. This was a real problem because the longest part of the level generation was instantiating the unity game objects that made up the level, which ended up having to be done on a single thread. The other issue I encountered was that the Parallel class was introduced in .NET 4.0, while Unity uses .NET 2.0 by default. In order to switch the newer version of the framework I had to

download a new version of Unity and switch to the experimental .NET 4.6 runtime.

Parallelizing the cellular automata portion of the level generation yielded very little benefits, so little that I couldn't measure it normally. The cellular automata normally runs for 6 generations per room. To demonstrate that the parallelization was working at all I increased that to 500 generations and tested with and without parallelization. Even running it for 500 generations the time to produce the level was only reduced from 5.5 to 2.5 seconds (this was on a quad core processor without multithreading).

Rooms also must be seeded individually now, because random number generator is also not thread safe. Switching to the experimental runtime also made the development environment far less stable than it previously was, making the decision to parallelize even less sensible. Also, because parallelization was one of the very last problems I worked on I didn’t end up using the .NET 4.6 features anywhere else.

There were other mistakes that arose from a lack of design foresight. One of the things I overlooked in my original plans was that the doors were going to be the triggers for the room locking down and the enemies spawning. The problem with this plan is that the player character could be inside the door when it spawns, or potentially even in the wrong room. This didn’t end up being a big problem, but led to code that was less clean than it could have otherwise been.

Grade

While the implementation may not have turned out as clean as I would like, I did manage to accomplish almost everything I set out to do. I never implemented a teleporting enemy mostly

because there are many edge cases to deal with like enemies teleporting inside of other enemies, or inside of walls or outside of the map. The other feature I put points on but did not implement was having the character’s laser be stopped by walls. I had intended to implement this by using the ray casting system in unity to find the distance to the nearest wall and setting the laser’s length to that distance. However having a ray collide with some objects (walls) but not others (enemies) required using the Unity's layer masking system, which ended up being too confusing to be worth the three points.

I appear to have made an error in my proposals rubric by stating the program was out of 100 points when the points actually added up to 105.

Grade	
Player Behavior	18/21
UI/Menus	6/6
Map Generation	50/50
Pickup System	10/10
Enemies	15/18
Total	99/105