Procedural Map Generation

# By Randy Hash

Procedural map generation is a method in which a computer uses pseudo-random number generators and an advanced algorithm to create a map or level that adheres to a set of rules and guidelines set by the developer. Games that have employed such tactics have been around since the early 1970’s and these algorithms continue to be used in games today, such as Path of Exile, Diablo III, and Quest of Dungeons, a simple turn-based Roguelike game released on Xbox One, to name a few. Some games have used this concept of procedural asset generation to even populate fine grain, decorative rendering to massive worlds, much like Star Wars Galaxies would generate and display grass, shrubbery and non-collision objects. For my project, I explored three different algorithms as well as using the Unity Game Engine to visualize the map generation. With that said, the aspects of the project that I will discuss are the visualization through Unity’s 3D game engine as well as my three algorithms: Columned Room Generation, Drunkard’s Walk, and Binary Space Partitioning.

The basic system I used in Unity 3D was a compilation of standard assets as well as custom made assets. The basic character model and flat grey material used to render him were standard assets of Unity. The scripts used to control the character and the following camera were taken from the standard assets but needed to be modified to fit the requirements of this project. I took the standard animations and added in events used to trigger playing of footstep sound clips. In addition to the footsteps, I added ambient background music as well as sound effects for the portal to progress to the next level. As for the physical representation, I have implemented multiple skins with different textures for walls, ceilings, and flooring. I also implemented mirrors, but they are not optimized so it causes a bit of lag. All of these physical components are used and generated with each random map generation algorithm.

The first algorithm I implemented was a simple Columned Room Generator. This algorithm I derived from a basic algorithm from Unity’s official 2D Roguelike tutorial, in which I modified so that it would be able to work with the 3D system I was working with. The structure of scripts as well as how those scripts access the prefabricated models were also taken from Unity’s Roguelike tutorial. This algorithm first creates a square room with height and width directly correlated to the current level. The algorithm then fills the interior of that square with floor tiles. A pseudo-random number generator then assigns random floor tiles with columns; the number of columns scales directly with size of room. The player is then placed into a corner of the square room and the goal is randomly located on a free floor tile.

The second algorithm I implemented was a Drunkard’s Walk, also known as Random Walk. This algorithm is good for generating more organic or cave-like structures. The algorithm starts at the origin position and uses a pseudo-random number generator to choose a direction: 0 for North, 1 for East, 2, for South, and 3 for West. The algorithm then places a floor tile there, and does the same operation from that position. The exact direction the algorithm chooses does not matter, but one way to skew a map generated with this algorithm is to increase the modulus value, say to 10 instead of 4, and then limit certain directions to less probable chance. For example, to create a more path like map, you could use modulus 10, mapping the number 0-3 to North, 4-7 to South, and leave 8 and 9 to East and West respectively. This gives a much longer, path-like map. This part of the algorithm continues for a set length that scales with the level. After all floor tiles are placed, the algorithm then follows the path a second time to search for where walls need to be placed and places them. Lastly, the algorithm places the player at the origin of map creation and places the end goal randomly in the map.

The last algorithm I implemented was a Binary Space Partitioning algorithm. This algorithm starts with a large square, which scales with the level, and recursively splits the square and then all subsequent rectangles into relatively half until the algorithm has separate rectangles, call leaves, that fit into the parameters set for room size, which for my algorithm I used a minimum height/width of 6 tiles and maximum height/width of 20 tiles for all sizes of maps. Once all leaves fit those parameters, random heights and widths as well as positions are picked for a room within a leaf. This leaves space between rooms. Now the algorithm places hallways between related rooms, meaning rooms that have been separated from the same leave in the first step of the algorithm. Once halls and rooms are constructed, the algorithm lays all floor tiles and uses the same wall placing algorithm as in the Drunkard’s Walk algorithm. Lastly, the algorithm places both the player and end tile randomly within the map. The downfall to this algorithm is that it does not create any cycles in rooms. Each room is typically only connecting to at most 2 other rooms as well as no room ever connects to a room that it did not share a leaf with at one point. This causes the maps to look more like a base hallway with rooms along it with hallways going in the same direction stemming off from each room on the primary hallway.

These algorithms are just the tip of the iceberg when it comes to random map generation. Through this project, I have found a real passion for game development as well as random map generation. For future development, I plan on implementing at least 2 more algorithms that will give a more maze-like map as well as one that will produce a map with rooms and corridors that can contain cycles. I plan on using this project as a base system to develop further into a game I would like to create for my boys. We are going to make a classic turn-based RPG with random encounters as well as creature-cued encounters. This will be my project over Christmas break and probably a bit into next semester.

///Unity Project and Source Code can be found on my GitHub. Project too large to upload