# Complex Game Systems Brief

The final procedural dungeon generator that I have submitted uses Binary Space Partitioning to create a mesh and procedurally split the mesh into different rooms to act as the dungeon rooms. This is a considerably simple generator and isn’t what I had hoped to have as the end product.

## Issues encountered integrating the Modular Complex System

During the preparing stage of my modular complex system I outlined many goals that I wanted to achieve for my modular complex system. Features such as scalable levels, the ability to involve different algorithms within the procedural dungeon generator were core features that I wanted to involve within my complex system however were met with struggle and ultimately didn’t feel it was in my ability to include these features given the time I set out to complete this custom system.

### Algorithms

I encountered these various issues because of the information out there not being easily accessible and as well understandable. The implementation of algorithms such as the Delaunay Triangulation was something that I struggled to grasp and get a good understanding for how the math worked. Trying to learn implementations of Delaunay such as Bowyer-Watsons implementation was what I had set out to accomplish during the preparation stages and although I believe I had gotten some understanding on how it worked I decided given my current skillset it would not at all be possible to learn given the time frame. Attempting to learn these algorithms is something that I should’ve realised wouldn’t be easy and would challenge me during the preparation stage of this complex system. I am ultimately happy that I did pick a fall-back algorithms to implement for the dungeon generator as I don’t think I would be where I currently am If I had decided to stick with using an algorithm that I hadn’t become comfortable with. My fallback algorithm ended up being Binary Space Partitioning (bsp). Implementing bsp seemed a lot easier to get an understanding of especially because of the amount of easy to learn/ understand websites and YouTube videos that can cover how the algorithm works and different implementations of it.

### Scalable levels

I ultimately felt that although I deeply wanted scalable levels to be a core feature that would set out my generator from the rest that because of my bad time management skills I would not be able to include something like that. I feel because of the way I set out my generator being in 3D space rather than making it within a grid system like so many other generators that incorporating different dungeon levels would be a lot trickier to include and this is something that I needed to consider a lot more heavily during the preparation stage of this task.

## Performance of the system

I believe that in terms of performance of my procedural system it runs fairly well and doesn’t take long to generate a random dungeon given what is included. To ensure that the project runs well I made custom editor buttons that would clear what the user has already generated so that they are able to experiment with different layouts for how they want the dungeon to look like. This feature is something that I believe is necessary as it means users don’t need to manually delete each part of the dungeon as that can take some time to do. Other buttons my project features are a generate dungeon button which allows the user to view what the dungeon looks like within the editor.

I believe that my system has a lot to be desired when it comes to optimizations for how my system currently functions. Changes to the way the current algorithm works will need to be changed if users want to create more ways to traverse through the rooms. I have thought about how this could work/ be implemented however given the current time this isn’t something I’m interested in finishing but would be something to revisit in the future.

To test the performance of the dungeon generator I have decided to compare load times for my dungeon generator and a pre-existing dungeon generator, to get correct times for generating the dungeons I used the stopwatch function in my generator script to get an approximate time for how long each procedural dungeon takes to finish spawning in rooms. Below is a line graph on the differences.

From the data that I extracted I found that generally my dungeon had faster room load times compared to the dungeon generator I tested against, unfortunately this didn’t hold up and as I increased the number of rooms in my dungeon the algorithm for spawning and splitting rooms become skewed.

Where the system fails however, is when trying to generate more than a hundred rooms as the algorithm get’s a bit messed up and things such as corridors and rooms get mixed up and ultimately the dungeon begins to look weird. This is something that I decided to not fix currently and should be revisited later.

## Required changes for the system to function as intended

There are a list of changes that I would need to take for the system to function the way I intended for it to be made during my preparation stage.

### Changing Algorithms

Features that I have talked about before such as the implementation of more than one algorithm to generate the procedural dungeon would be the first step in realising my goals that were set out during the preparation stage. I believe that I was close with my implementation of including algorithms such as Delaunay Triangulation however I decided that given my limited math knowledge it would not be worth to go through all that effort in trying to include more than one complex algorithm especially when I wasn’t confident on how the system works. Other thoughts that I have had is adjusting the current bsp algorithm to create more ways for the user to traverse through the rooms to reach different rooms easier would be easy to accomplish I just don’t wish to visit it at this current time.

Other important features that never made it to the final stages of my procedural dungeon generator are things such as proper pathfinding, during the production process I had planned to use a minimum spanning tree to connect the procedurally spawning rooms and although again I believe I was close to this Implementation It ultimately didn’t get finished and I used a method to connect rooms together through bsp.