
# Computer Games Development
# Project Report
# Year IV

Joshua Boyce Hyland
C00270917
11/0/2025

# Contents

# Introduction

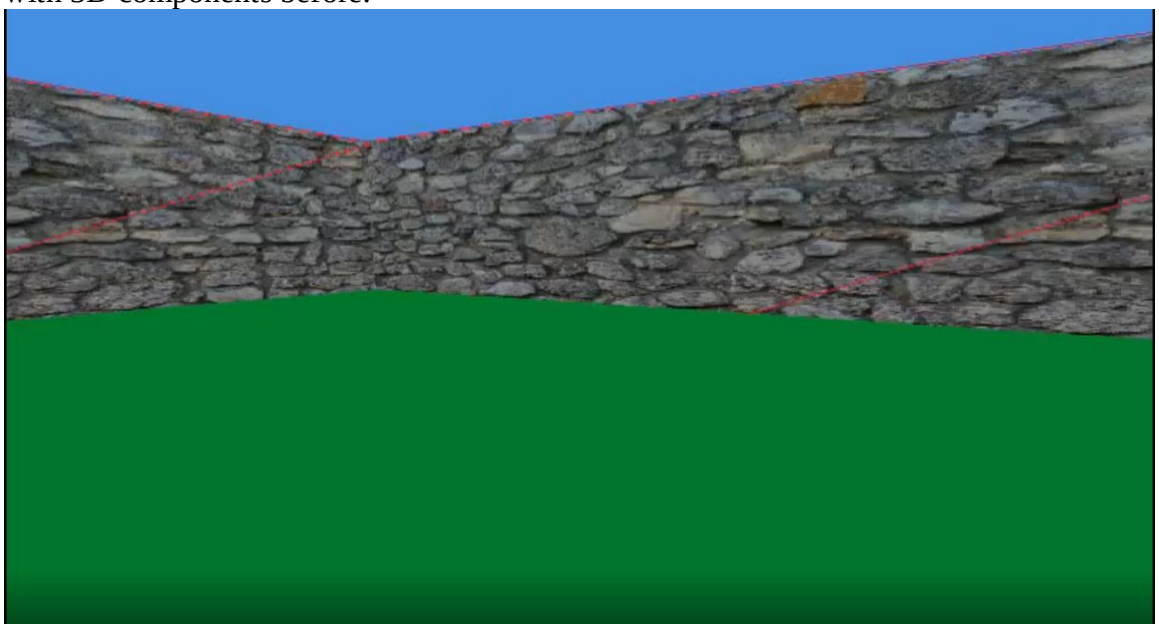
For this report will outline the design and development of my final year project. A spaced themed game built in C++ using SFML. Focused on procedural generation, player driven editing and NPC integration. The project when through some an early concept shift which I will detail. The following section will describe how my concept was developed, the technical challenges and the systems implemented to support my gameplay.

# Project Direction Introduction

Initially I had my mind set on doing some sort of survival game for my project which would include various passive systems that you would usually see in one that were constantly happening around you such as weather, resource growth (like trees and crops) and entities that lived in the world that went about their day to day. This grabbed my interest with the opportunity to design systems that would have this constant cycle and a schedule while player interacted in the world. Things like different types of animals going about their day hunting or grazing with their behaviours. The general ideal that intrigued me was this sort of simulated world.


My original inspiration being from survival games which I enjoyed playing myself along with other games that had systems I thought were interesting such as A Plagues tale which has realistic swarming mechanic. Games With player driven building took my focus as I was interested in the idea of giving the user the ability to create something that effected the game. For graphics library I went with Raylib, based on the idea that I hadn't used a graphics library

with 3D components before.



Upon doing getting some time to think on my project concept and doing some work on my project in the first couple of weeks and getting some simple things running in Raylib like a first-person controlled character and walls. I found that with my initial idea that I planned on making game features that I like playing in my own time but not one that I found very interesting to make and implement. Having come to that conclusion I went back and rethought what I wanted from my project.

I decided to go down a different direction with my project, I went back on Raylib and decided to use SFML as I thought if I wanted to do anything complicated, I wanted to spend my time solving the technical problem of the feature and not my unfamiliarity with the library I was using.

With this change of library, I also changed my goals to some degree. I still wanted to make something that was modifiable by the user like base building but not focus on the sort of NPCS systems scheduled systems as I mentioned before, instead I wanted to do something procedurally generated as this is something I had no experience with, and I thought would be a very interesting to pursue.


1.

Upon deciding on player modifiable structures and procedural generation being my focus I decided it would be space themed, and the player would build their own ship and space station with the enemy space stations beig procedurally generated. But how would I go about creating a generated space station in C++ using SFML in a way that supports NPC navigation and gameplay interactions?

# Literature review

My main point of research for this project was the dungeon generation as this was the most unfamiliar aspect of the project I delved into. While other parts I had experience with before were still difficult to implement and took time but I was able to rely on my own prior experience with these topics and / or lecture notes for things like the Decision tree. I still needed a small reminder of some topics such as A* which I brushed up on my watching a short video at (Computerphile (2014) *A* (A Star) Search Algorithm).

In researching procedural generation, I found a lot of references to what people where calling the tiny keep algorithm which was a dungeon generation algorithm implemented for a game called Tiny Keep. Upon seeing its steps to implement a procedurally generated dungeon at (*Procedural Dungeon Generation Algorithm*, Game Developer) I found myself interested in pursuing this method as it had a diverse range of techniques used in stages to create the final product of a generated dungeon. This involved a Delauney triangulation and a minimum spanning tree both of which I had no experience with.

What attracted me to this method was the how the randomness was created by the initial spawning of rooms in radius and then applying a separation force to them until they were all separated, following then a culling of rooms of a particular size which would leave you with a unique structure of rooms each time.

In this description of the generation a game engine was used and many different libraries to perform the Delauney triangulation and minimum spanning tree. These I implemented myself in C++ as it gave me the opportunity to learn even more outside my comfort zone instead of delegating it to a library. These were the main 2 topics I had to look into the most.

I did some research into the concepts of Delauney triangulations at (*Introduction to Delaunay Triangulation*, Tinfour Documentation), along with some other videos I have reference down below, to better understand the concept of what it was I was going to implement, but also what its use cases were outside of mine, which I found very interesting to find that it was used for creating meshes of non-overlapping triangle to form a set of points which gets commonly used in terrain generation, 3D modelling and mesh generation.

Along with this I also did a bit more research into how to implement a circumcircle which I understood the concept and how it played its role in the Delauney triangulation but in code I spent a lot of time trying to get it to work but couldn't. I ended up getting AI assistance for the construction of the circumcircle from 3 points.

The minimum spanning I also research at (GeeksforGeeks, *What is Minimum Spanning Tree (MST)?*,)which provided some good explanations and visualisation on how the algorithm worked along with some variations on it. I ended up going with Prims which is a greedy search algorithm. This research helped me understand the algorithm and its purpose for my project on cutting down the edges created by the triangulation so that we ended up getting hallways that could be navigated from one end of the base to the other that weren't excessively cluttering the dungeon.

These sources and techniques helped guide the design of my procedural generation but also helped improved my own skills in implementing pure ideas and not relying on working from code examples as having to implement these features from understanding the concepts really improved my own skills as a programmer.

# Evaluation and Discussion

## *Pathfinding*

The pathfinding system in this project was implemented using a grid-based version of the A* algorithm. This system is responsible for NPCs pathfinding from their current node to their goal node. Due to the bases not having any dynamically moving obstacles the path gets calculated all at once at the start of when a goal is assigned.

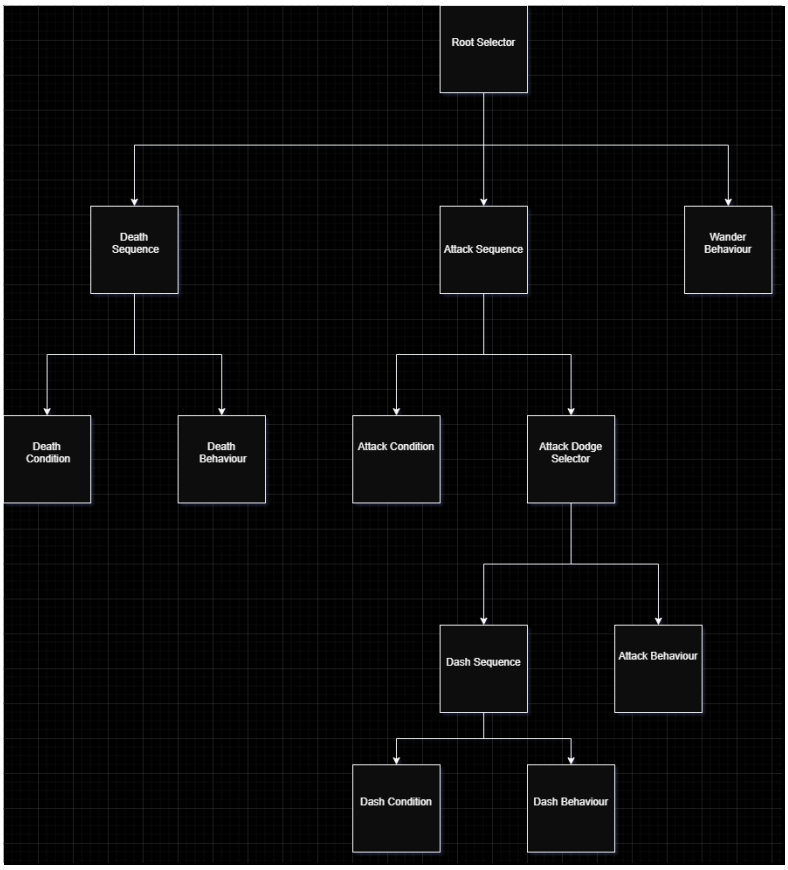
My main goal for this agent was for it was to function similarly to the Unity NavMeshAgent where a goal position gets assigned and the agent handles the movement logic independently. This design is very simple and makes traversal a simple aspect which get delegated to an agent that behaviours then don't have to manually worry about. Which I believe I successfully achieved, as in my implementation you need only set a goal for the agent after which it manages the path that needs following and updates it position over time. For the developer the only thing you need to do other than call its update function it is set your character position to the agents own position.

While it performs well, I would have liked to be improved would have been to add a hierarchal A* where each room also had a node. This design follows a sort of layered structure of nodes with the first layer being the rooms and the second being the cell-to-cell traversal. Its purpose would have been to first plot a what rooms we need to go through to reach our goal using, each room having a node this would work the same as the cell-to-cell pathfinding but instead we would be directly plotting a path from room to room directly which would make the movement more realistic and efficient. This higher level of nodes per room would have improved the NPCs pathfinding making it more efficient and realistic for movement between rooms.

## *Decision Tree*

The decision tree was used to manage NPC behaviour. Each NPC is assigned its own custom decision tree compose of selector, sequence and action nodes. This allows for NPC to decide relative to different gameplay contexts such as attacking dodging when under threat or transitioning or a death state.

The system worked well in practice. With the decision tree logic being cleanly separate from the actual behaviour implementation. IT was easy to build different trees for NPCS and add on behaviours without having to rebuild the tree or the behaviour itself. This made testing and debugging also very efficient. As everything was quite self-contained in its logic, so it was clear if there was a problem with the behaviour, decision tree or the condition logic.




Behaviours transition where responsive and clean, with NPCS correctly evaluating condition and moving between behaviours without any noticeable delay or inconsistency. Th chaining behaviours was made simple with the tree for example when adding the dash sequence to the tree I needed only create a new selector and make the dash sequence and the attack behaviour a child of the selector.

The system worked well for my project, but I would like to have organised part so it better such as the construction of the tree, having some sort of class that constructed tree with specific attributes for NPCs and handed it off to them rather than them having to put it together themselves. In general, though the system worked well for my project and made debugging and problem solving when implementing behaviours a much simpler task.

## *Animator*

The animation System plays a key role for many classes in animating their sprites and the timing of gameplay action.

Its designs focus was on modularity once again taking inspiration from unity where this could just be a component which could be given to a class that would animate its sprite and manage its textures without the user having to worry about it.

The modular design allowed for the animator to be used by all behaviours individually animating appropriated base on their current state while also being able to activate certain game event when animation ended due to the animator notifying the used one a loop of the animation was completed.

The reusability is a specific strength of this component and a goal I wanted to achieve as it was highly reused among classes like NPC, Player and Weapon to animate their respective sprites with ease.

A addition to this animator I would like to add which I touched with notifying the used of a finished animation is animation events at key frames, this would be interesting to add to the animator and give a wider variety of use among classes and give even more room for customisation.

## *Weapons*

The weapon system allows for customisation through derived classes. While the base class implements all the base functionality of managing the weapon and the bullets its fired along with the particle effects it fired off, it's up to the derived class to specify how the bullet it shot and what it looks like.

This approach made it very easy to implement other types of weapons such as the shot gun. The use of polymorphism allowed for specific to lie in the visual and how the bullet or bullets got shot. Along with this bullet class was made simple to so that it could be easily reused by all weapon types, by assigning a different speed value and sprite.

The animator also came in use here when specifying when a shot would be fired. This being on the last frame of the shooting animation, which provided a satisfyingly timed shot.

The bullets being inherited from GameObject also made their management in the grid and when they hit an obstacle or NPC very steamed lined. This process was managed by the weapon along with the particle system being activated upon a collision.

An improvement I would have liked to have made is probably some more effects for bullets in the way the responded to collision, having ricochet effects. Overall though I am happy the way it turned out with the weapons as I think there design is maintainable and extendable while also delivering satisfying gameplay feedback through the particle effects

## *NPC and Behaviours*

The NPC system is a culmination of the systems. Its brought to life with the agent, animator and decision tree components. Nearly acting as a container of for these systems to interact with each other.

The NPC transition between behaviour states base on the decision tree. Its either wandering, attacking, dashing or dying. These behaviours are transitioned through based on conditions in the behaviour tree.

These behaviours take reference to the other components of the NPC so when in control the behaviour has access to all components in a self-contained manner.

Due to the componentised nature of the NPC each part takes responsibility for specific roles leaving the behaviours as the managers of the they are used and the NPC as the container for this all to happen.

The only logic which I would have liked to extract was the decision tree as in the update as this could have been delegated to another class along with the building of tree.

## *Base Editing*

The base editing system allowed for players to customize the shape and look of their base using the tiled based editor. Through the dedicated UI, layers could place walkable or non-walkable tiles along with interactable world items directly onto the grid. These changes effect the player and NPC navigation of the base along with the overall structure of the space stating.

This system is tightly integrated using other components like the Grid, TileEditorBox, TileLibrary and WorldItemLibrary. Each cell tracked its had values which where altered through the editor such as the walkability and if there was a World Item placed on It. The UI is clean an intuitive to use with different tabs for different section.

Functionally the base building met my design goal of give the player agency over being able to modify a structure. It enables dynamic change to the visuals of the base and also how NPCs navigate it. The base is also saved to a json file using the nlohann::json library, meaning you can come back to the exact base you created. using the One area of improvement I would like to make is validation of say the structure of being closed off properly with wall or if all walkable cells are reachable from each other.

## *Ship Building / gameplay*

The ship building system worked quite differently to the base building system, instead of working of cell to be placed on the ship system worked off ship parts and connection points. Each piece had 4 connection points which when selected and dropped try find the closest connection point to it on another ship part within a distance to the part.

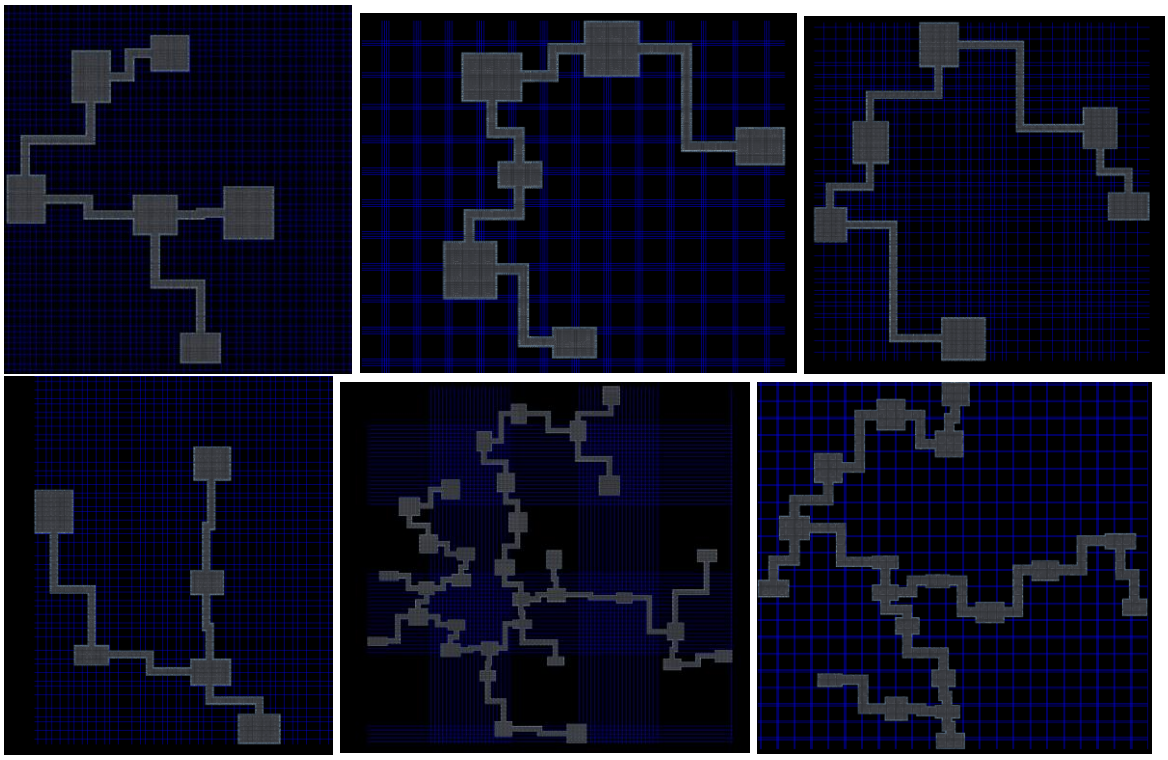
This was one of the first features and it worked out well functionally. It also has an EditorBox, ShipPartEditorBox, which uses a ShipPartlibrary to get a selection of parts which are broken up into the part type they are. So, while the UI and method of getting parts works the same as the tile base editing the connection of parts follows a different logic based off the sprites connection points which are based off the sprite texture.

Along with this the ship is flyable to different bases in which It can land at, making it so you can customize your own ship and fly to an enemy space station. This portion of gameplay also uses the MinMap to help navigate you to the enemy base. Had I had more time I would have added more sprites and effects for the flying of the Ship itself.

## *Procedural Generation*

My procedural generation of the space station was very successful. The algorithm consistently produced varied room layouts with each room properly spaced and connected through a logical network of hallways. When creating the perfect spacing though I had to make some changes as I found with the default algorithm it often placed rooms quite close together resulting structures a bit awkward looking, so I adjusted the collider of each room during the separation phase to be twice the size, so it gave it this buffering room which solved my problem and made for better layouts. The use of the Delauney triangle and minimum

spanning tree made sure all rooms are reachable while having a realistic structure that could be traversed from one side of the dungeon to the other.



A key benefit of how I designed the dungeon generator is its flexibility. Through some adjustable parameters, such as the number of rooms you want spawned, their max/ min room size and hallway dimensions make the system adaptable and easy to fine tune.

Most importantly the resulting dungeon worked well with other existing system I made for the project like the NPCS pathfinding which worked seamlessly due to it being built with my already implemented grids.

One area I would like to improve would be its visual diversity, adding another layer of randomness to when the textures where assigned would give the dungeon a fresher feeling room to room instead of the currently standard look throughout. Introducing a simple decoration system would greatly improve the feel of the generated dungeon also with placing decorations or even theming some rooms around a function.

The speed of the generation varies dependent on the number of rooms you spawn in ideally but in general it is quick enough that it can be done at runtime but requires a small second or 2 wait behind a loading screen. The final generated grid takes up some memory but due to the optimisation of only rendering what the camera seeing it doesn't affect performance in any meaningful way.

## *GameObject Driven Design*

Upon creating the grid structure, I didn't want to have to have global checks constantly as this would be in efficient so once again I though about how game engine run these sorts of

interactions and Gameobject came to mind in unity. In their system we can in game objects are unified under a monobehaviour class which allows for interactions between other monobehaviours. That's what inspired my implementation of the GameObject driven design.

This design allowed for me to unify all objects in my game under one umbrella from the perspective of the grid. Each cell in the grid holds a set of GameObjects currently occupying it. The game objects add and remove themselves from the cells set as they move around the world but what this means also that a GameObject can simply check its surrounding cells for GameObjects occupying these cells for any collision if there even is a GameObject in these cells making collision checks way more efficient in contrast to doing constant global check with ever object.

This design also heavily aided the design of the MiniMap as it allowed for it to take in a vector of Gameobjects and assign MiniMap icons based on the gameobjects tags making the process very streamline and intuitive.

This is an addition I am very happy with as it made me understand how I should design my systems from the get-go.

## *Centralised Resource Management*

To streamline the management of assets and improve maintainability I implemented a set of centralised libraries which include TileLibrary, WorlItemLibrary, ShipPartLibrary and Loader System. These were designed as singletons responsible for handling the loading, construction and management of items of the same type that are used through out my game such as tiles, textures, ship parts and fonts.

I implemented most of these systems early on and they proved to be highly effective and useful to me throughout development. They ensured that each texture and font assets was only ever loaded in one and shared by reference thought the whole game. The libraries also simplified the process of constructing object like tiles and items as the libraries instantiate these items one, categorised them and then distributed copies based on Enums and indexes.

This system also aided the saving of the map to file, because the tiles where categorised by an enum and an index, this information could easily save to a file and be loaded back in when needed.

This system of centralised construction was a real advantage when it came to consistency among objects as it prevented the duplication of logic and the possibility of the same object being constructed in a different way.

# Technical Achievements

## *Procedural Generation*

- Implemented a procedural generation system using a Delaunay triangulation and Minimum Spanning tree to create a space station of varying layouts.
- Resulting dungeon that was populated by nodes traversable for pathfinding.
- Parameters allow for flexible tuning of generation process.

## *Grid Spatial Partitioning System*

- Created a reusable gird and cell system that supports tile placing, pathfinding and collision checks against objects that are spatially relevant to each other.
- Performance optimised to only render relevant cells to the players camera and limits logic checks to neighbouring cells.

## *Pathfinding Agent*

- Designed with a modular A* based system which handles the navigation to targets independent of the NPCs behaviour
- Integrated seamlessly with the procedurally generated layouts of the dungeon.

## *Decision Tree*

- Bulit on modular and interchangeable nodes (Selector, Sequence, Condition and Action Nodes)
- Enables behaviour switching in game based on specified conditions.
- Very extensible and reusable across different NPCs.

## *Animator*

- Handles frame by frame sprite animation and management across multiple classes
- Allows for animation completion to be tied to game events like shooting or dashing, accommodating to synchronised actions.

## *Base and Ship Building systems*

- Created in game editors for the base and ship construction, allowing players to place or remove tiles and item dynamically.
- Base changes get saved to a json file which gets loaded in upon starting the game

## *Weapon and Bullet System*

- Built modular weapon system with support for derived weapon typed and behaviours.
- Implemented a responsive projectile system with particle effects, configurable speed and collision detection

## *Mini Map*

- Implemented a mini map system that displays player, NPCs, Important grid layout with corresponding Icons which tracks position in real time to inform player.

- Used Shared GameObject Tage centralise the tracking the conversion of GameObjects into MiniMapIcons and MinMapGrids.

### *Centralised Resource Libraries*

- Developed Centralized singleton libraries ( TileLibrary, WorlItemLibrary, Loader, ShipPartLibrary) for managing the distribution of tiles, world items, textures, fonts.
- Ensures assets were only loaded once and reused efficiently throughout the game, reducing memory usage and improving performance.
- Allowed for new assets to be added by simply dropping assets into designated folders with no extra code needed, improving scalability.

## Project Review

In my project I started off in quite a different with wanting to use Raylib and have my features be based around simulation systems. I think changing direction was a good choice, I am glad I got try Raylib out, but I think for the sort of features I ended up wanting to implement I would have ended up fighting with the library more than the actual features problems. SFML allowed for faster prototyping compared to Raylib due to my experience with SFML.

In my project I think there is a lot that went well. The modular components that make up the NPC for one is system I am quite happy with especially the agent and animator. The Game object design implantation also gave rise to ease of use with the grid and the mini map which I also think turned out well with its displaying of alternative visuals. I am particularly proud of the procedural generation and its resulting dungeon but along with this I am also very proud of the step-by-step visualisation of the steps that built up the dungeon from the Delauney Triangulation to Minimum Spanning tree to the generation of hallways between rooms.

There is also thing that didn't go to plan naturally, I think my base and ship building and features lacked polish when it came to it and I think this is particularly due to it being something I got functioning to a standard early on in development but failed to come back to in time before the end.

I think if I could have done things differently, I would have set a smaller scope for my features to begin with, that way I would have completed smaller features to a more polished degree and then came back to expand on their functionality. Instead, I got a lot of features done to a functional standard but felt the need to move on before polishing to get big features like the dungeon generation done.

If I was to give advice to someone starting their own project I would say to work towards getting the game loop going first and build around that, as I ended up spending time on a lot of features individually that I then had to put together, which I got a good few int but there were some I didn't manage to get in before the end.

## Conclusion

My aim for this project was to develop a modular procedurally generate space station game in C++ using SFML, with system supporting pathfinding, player editing and NPC behaviours.

I successfully implemented the dungeon generation system I researched using a Delauney Triangulation and Minimal Spanning Tree, with each stage of the generation being optionally visible. Grid Based Traversal was achieved for both the NPC using A* pathfinding algorithm and their behaviours being decided a decision tree which decide what modular behaviour they are currently in. These NPC behaviours are powered by the agent and animator which help with following out this behaviour specified actions. Along with this Enemies and Player both used the same very transferable weapon system. Base and ship building was also achieved to allow the player to customize their base, saving it to file and build a ship to traverse to enemy space stations in. All these systems were built upon a centralises resource management system which was made with efficiency in mind by centralising the construction of objects and loading of assets to one place. Systems in my project where all designed with modularity and scalability in mind for more than one use case.

The final result of my project I think succeeded with my revised goals of doing some procedural generation and player modifiable structures to a great degree, especially with dungeon generation.

This project significantly improved my skills as a programmer. It challenged me to take pure algorithmic concepts from descriptions like the Delauney triangulation and minimum spanning tree them into functional system, something that before I would definitely have found intimidating and difficult but now I would find interesting and exciting. It also helped me thing on a larger scale and to write my code in a modular and reusable way that could be used across my different classes. This project also taught me

# Future Prospects

If I were to continue developing this project I would focus on further expanding the procedural generation system, which for me was the most enjoyable and satisfying part of the project. I would design a decorator system that could add randomized visual elements to diversify the appearance of the generated rooms. This could include thins like themes, furniture and environmental props to make the generated station fell more unique and lived in.

Along with this I would add more dynamic world content like interactable objects, environmental hazard or even randomized events could be very interesting. This would give the station more depth and repeatability. Given how much I enjoyed working on this apect of the project it is something I am considering continuing working on if not on this project in a new one.

## *Acknowledgements*

I would like to thank all my supervisors - Noel O'Hara, Lei Shi, and Martin Harrigan – for their support and guidance thought the course of this project. I would like to acknowledge Noel O'Hara for his consistent feedback and keeping me on track through the year.

## *References*

Computerphile (2014) *A* (A Star) Search Algorithm - Computerphile*. [YouTube video] 26 November. Available at: https://www.youtube.com/watch?v=ySN5Wnu88nE (Accessed: 20 January 2025).

Johnson, D., *Procedural Dungeon Generation Algorithm*, Game Developer, Available at: https://www.gamedeveloper.com/programming/procedural-dungeon-generation-algorithm [Accessed March 4 2025].https://gwlucastrig.github.io/TinfourDocs/DelaunayIntro/index.html

Triggs, G.W.L., *Introduction to Delaunay Triangulation*, Tinfour Documentation, Available at: https://gwlucastrig.github.io/TinfourDocs/DelaunayIntro/index.html [Accessed 11 March 2025]

Vedantu (n.d.) *About Circumcircle of a Triangle*. Available at: https://www.vedantu.com/maths/about-circumcircle-of-a-triangle (Accessed: March 18 2025).

GeeksforGeeks, *What is Minimum Spanning Tree (MST)?*, Available at: https://www.geeksforgeeks.org/what-is-minimum-spanning-tree-mst/ [Accessed: 24 March]