Procedural Content Generation Assessment Item 1 Report

**Development Log**

I started my diorama by building on the workshop tasks, galaxies are created with randomness to provide variance between them which are placed around the diorama. Random colour values are assigned to each star, with a rule that makes stars closer to the centre more vibrant (See Figure 1). Doing this procedurally adds more variation than a traditional skybox, with noticeable differences each time.

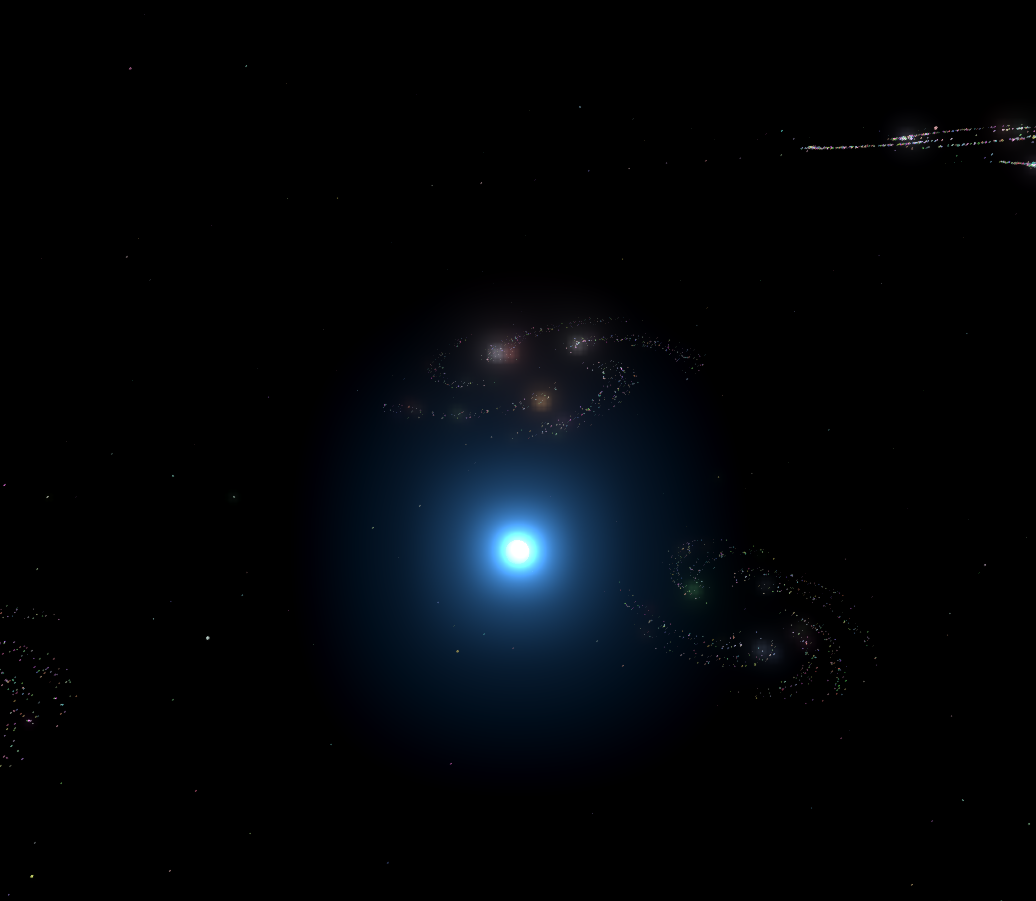


*Figure 1: Procedural distant galaxies being spawned around the diorama.*

**Narrative**

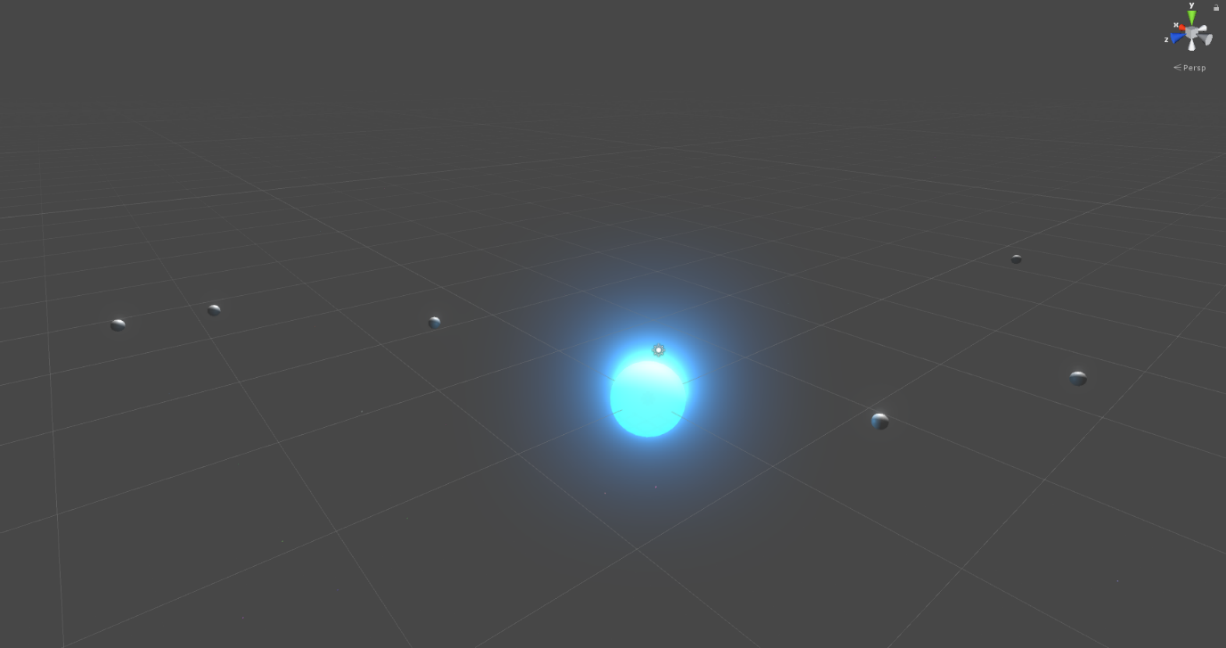
The diorama’s narrative pertains to a new mission by humanity to colonise distant temperate planets in their solar system from their home planet “Terra”. Two fleets have assembled near Terra to embark on their journey to expand to new worlds. The thematic style is realism.

The solar system’s star is generated with a random scale which determines its colour (See figure 2).

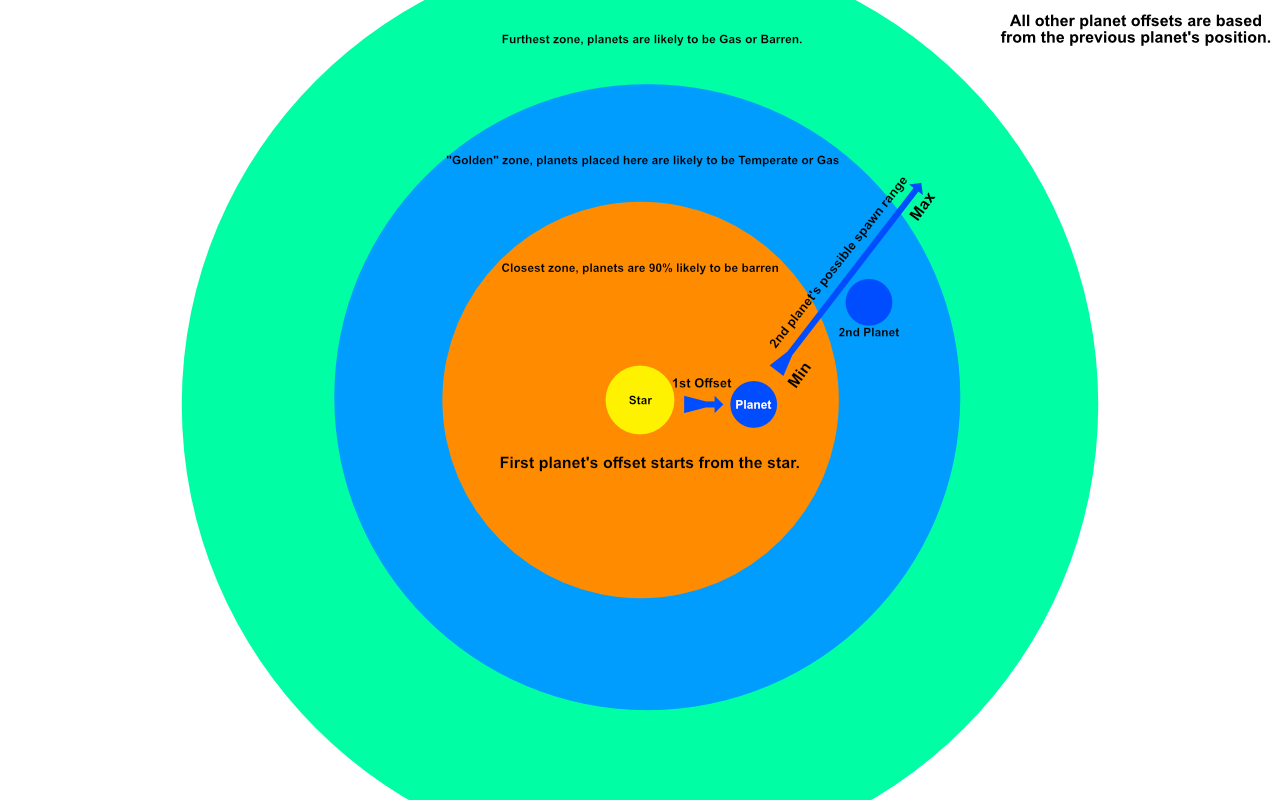


*Figure 2: A small star (with cooler colours) generated in the diorama.*

Planets are placed around the star by placing a planet based on the previous planet’s position with a randomised offset and their distance from the star determines the planet’s type (See figures 3 & 4). This system helps to create a different diorama whilst conforming to the theme each time, whereas manual placement would not offer any variance.

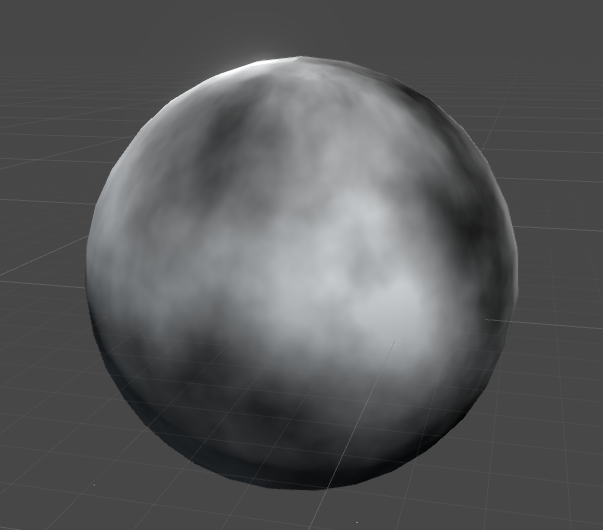


*Figure 3: Spawned planets orbiting around the star using the placement system.*

**

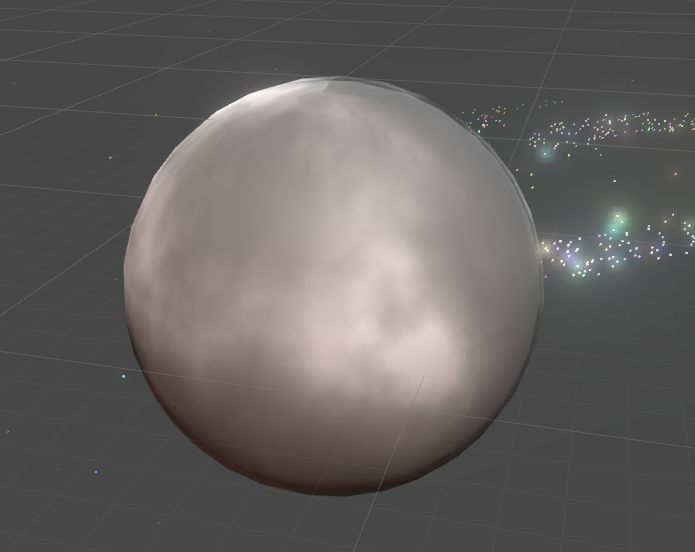
*Figure 4: Diagram of how the planet placement system works.*

Next, noise from LibNoise (Bevins, 2007; Mendez, 2010) was used to create procedural atmospheres for planets, though this texture was opaque and needed editing (See figure 5).



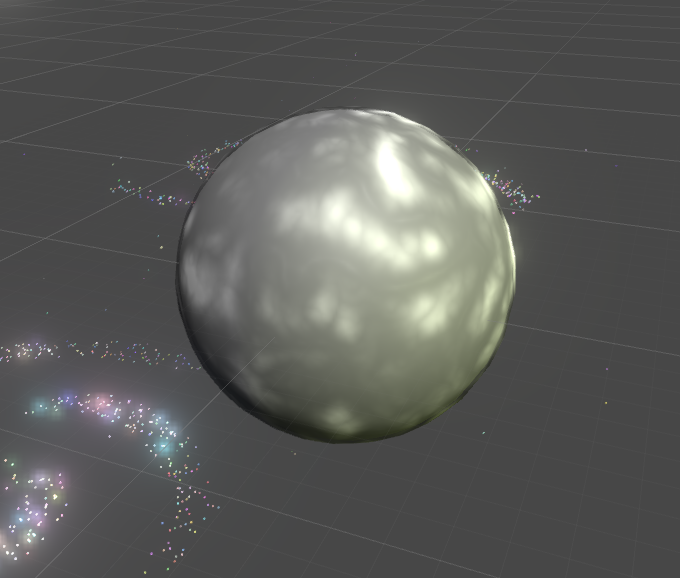
*Figure 5: A planet with a raw noise texture applied to it.*

Looping through each pixel of this generated texture and applying their grayscale values to the alpha channel solved this issue (See Figure 6).



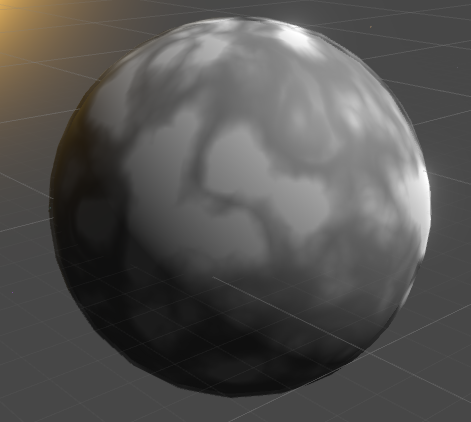
*Figure 6: A planet with the dark colour pixels phased out of the texture.*

I noticed that LibNoise contains more noise algorithms and decided to apply different ones to create a more realistic atmosphere (See Figure 7).



*Figure 7: Planet with Billow noise generated atmosphere.*

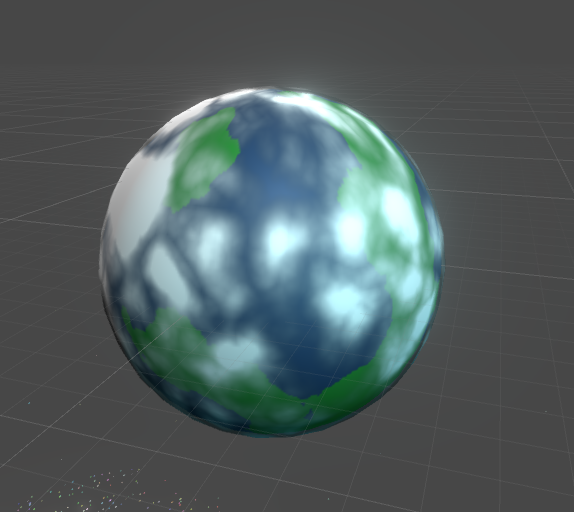
While this now looks acceptable for a planetary atmosphere, I combined the two noise generation methods together to see if it produced a more realistic result (See Figure 8).



*Figure 8: Planetary atmosphere created with two different noise types.*

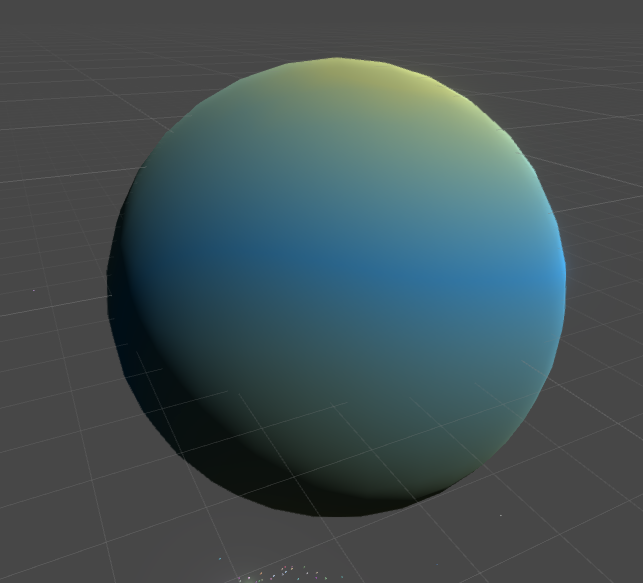
This method results in a thicker, denser looking atmosphere, both techniques can be used to create different types of atmospheres for planets.

For the planet surface, the first approach was to use noise to alter the planet’s vertices to create procedural terrain. While this could work, it would not be noticeable from increased distances and may be more expensive to generate. I instead applied a texture to the surface using noise and altered the texture’s colours based on a randomised grayscale threshold, darker values result in oceans and lighter values result in land, this produced a highly realistic result and conformed to the theme well (See Figure 9).



*Figure 9: A complete temperate planet.*

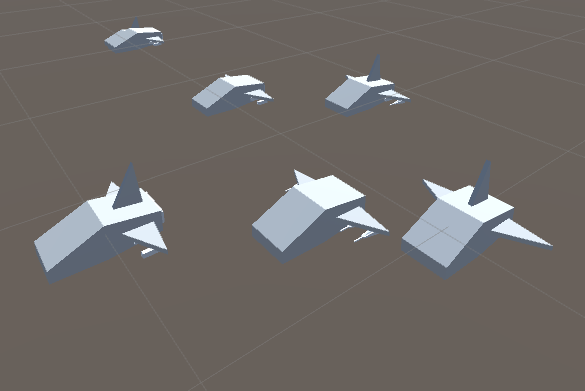
For gas and barren planets, the colours are interpolated to create a smooth gradient more akin to how they look in real life (See Figure 10).



*Figure 10: Gas giant with interpolating colours.*

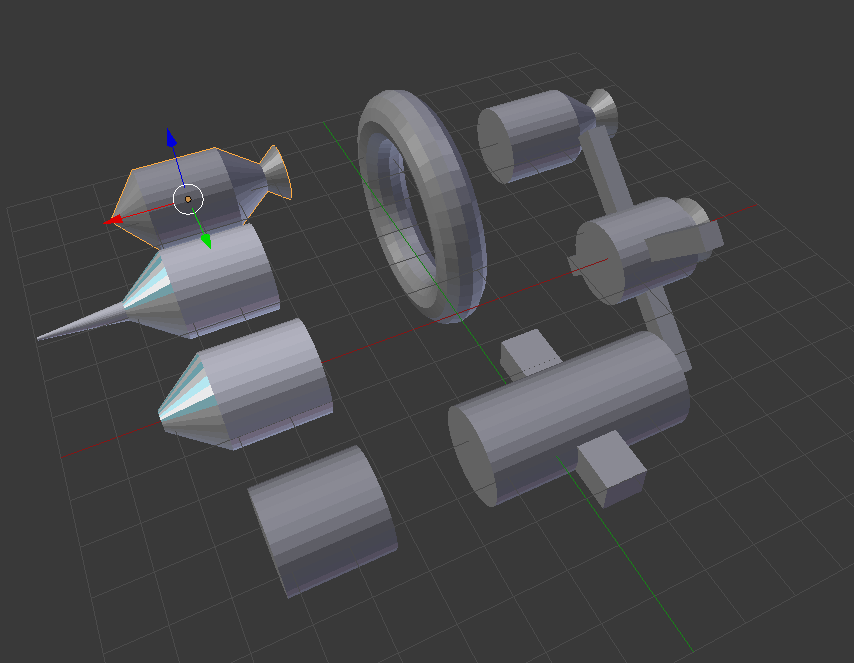
All planets are generated constructively with a random seed which allows for many diverse visual variations, providing a stochastic experience. Using coherent noise produced realistic atmospheres and surfaces quickly, this would be impossible to produce manually as fast with similar results for all planets.

Next, the guard ship’s base is instantiated from a prefab with empty game object “nodes” attached in predefined positions. Probability effects what components a ship receives, giving them variance. The simplistic nature of this system means that a generate-and-test method was unnecessary (See Figure 11).



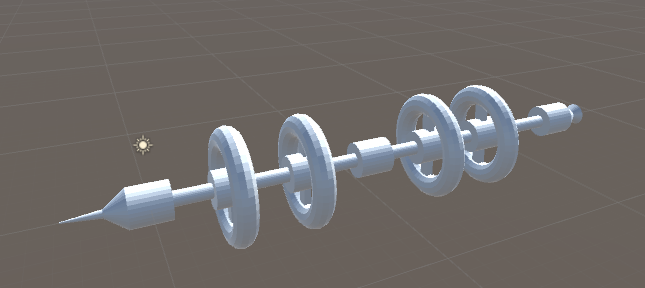
*Figure 11: Guards being created.*

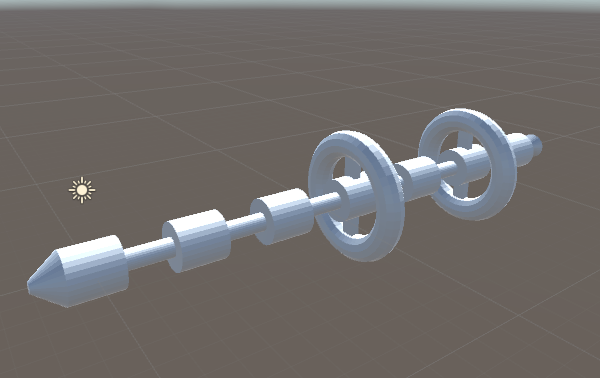
Colony ships are created using an odd/even number system that was previously created for a moonbase system. In this scenario, the odd/even system worked well, linking the different buildings together to create the moonbase sequentially, but the building’s orientations did not work properly. I realised this system could be used to generate colony ships sequentially from parts I had modelled (See Figure 12).



*Figure 12: Ship parts to be used in the colony ship generation system.*

The odd/even system rules that every odd number, a connection must be placed and every even number, a module must be placed. The module type depends on a probability variable that is randomly assigned each loop. The first number will always result in a cockpit module and the final number will always be an engine module. (See Figure 13).

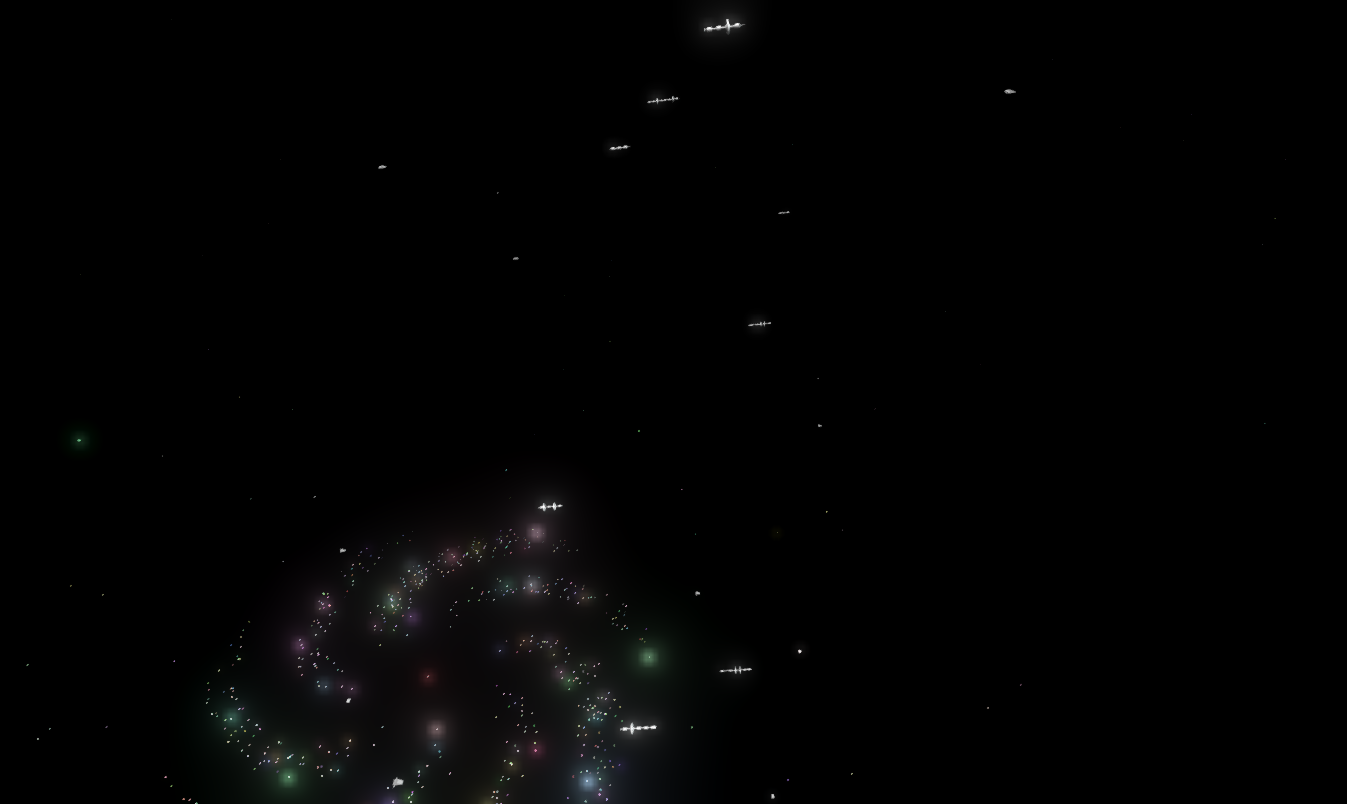




*Figure 13: Two colony ships made with the colony ship generator.*

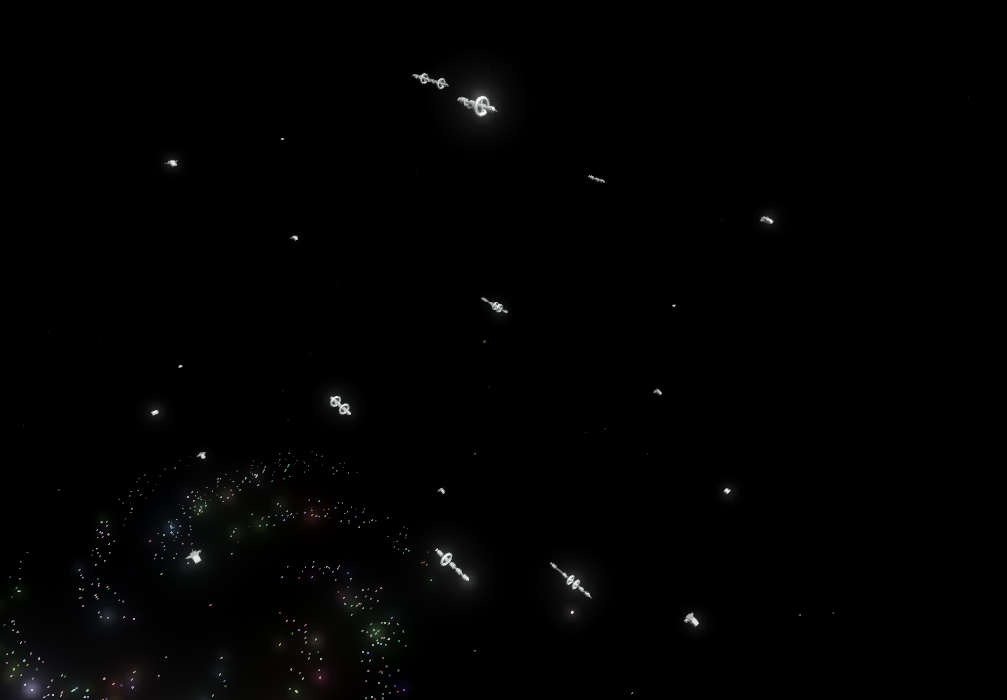
This system produces large amounts of colony ships constructively and reliably which vary in visual appearance, they can be produced quickly due to its efficiency. Modelling these manually would be a large time-consuming task but could result in more stylistic ships. These ships conform to theme’s realistic narrative and style effectively, as envisioned.

I used the odd/even system again to spawn the fleet to ensure even distribution between guards and colony ships (See figure 14).



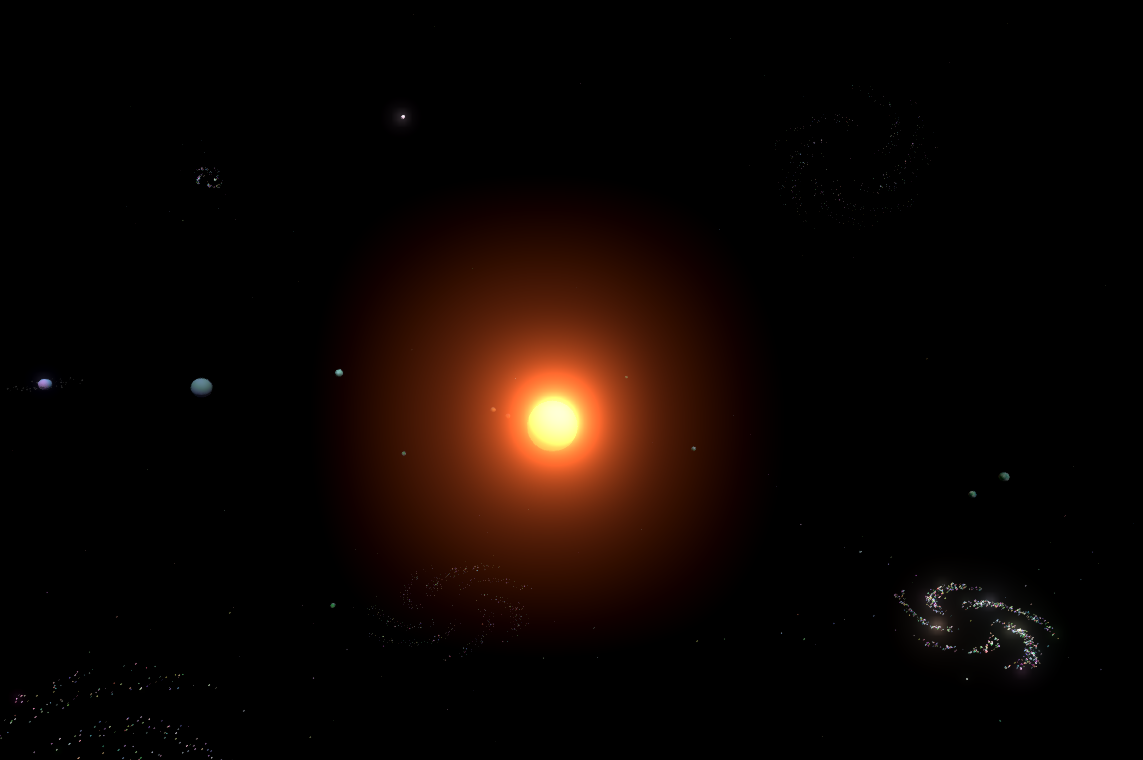
*Figure 14: The ships being spawned into the scene as a fleet.*

To move the fleet coherently, the first idea was to move the parent object towards the target, moving the ships with it. While this worked, the fleet felt static due to little movement amongst ships and wasn’t realistic. The boids algorithm was implemented to add emergence to fleet movements and to make the formation more realistic/believable (See figure 15).



*Figure 15: The fleet now with boids algorithm implemented.*

With fleets moving towards temperate planets the diorama was now complete (See figure 16).

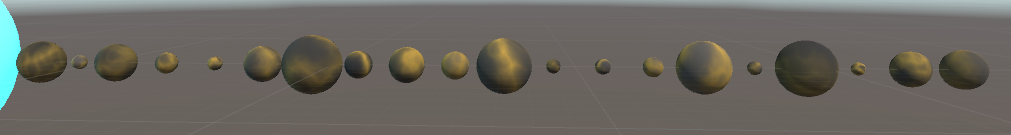
**

*Figure 16: Screenshot of the completed diorama.*

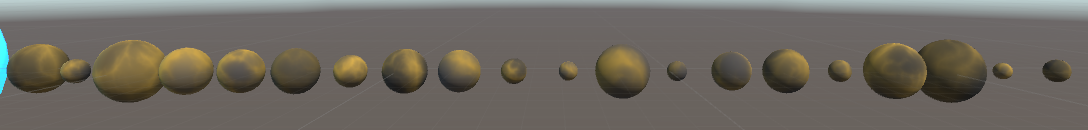
**Testing**

The approach for testing breaks down the procedural systems into their subsystems, such as the planet system’s texture generation for the atmosphere and surfaces. The independent variables of the algorithm’s parameters will be constrained to thresholds and tested in quartiles. A sample size of 20 planets will be generated in its own isolated scene to ensure these systems are working as expected at all ranges.

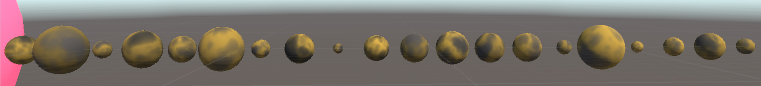
Splitting the variables into quartiles provides better coverage of the entire range by constraining the median value of the default range into different places. Using 20 planets as a sample provides ample distribution across each quartile, allowing for the program to generate the majority of outcomes to ensure they fit the theme (See figures 17, 18 & 19).



*Figure 17: The full/default range of variables being used to create Barren surfaces.*

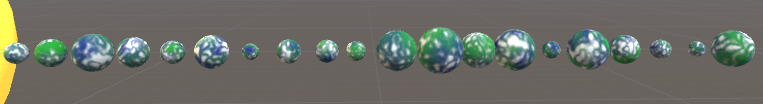
**

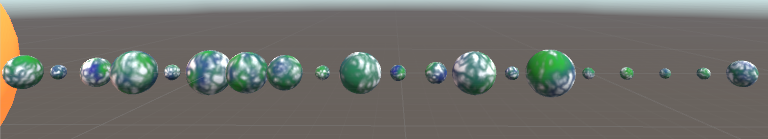
*Figure 18: Barren surfaces created with frequency range (0.4, 0.8 (default)) being constrained to the lower quartile (0.4, 0.5).*

**

*Figure 19: Barren surfaces created with frequency range being constrained to the upper quartile (0.7, 0.8).*

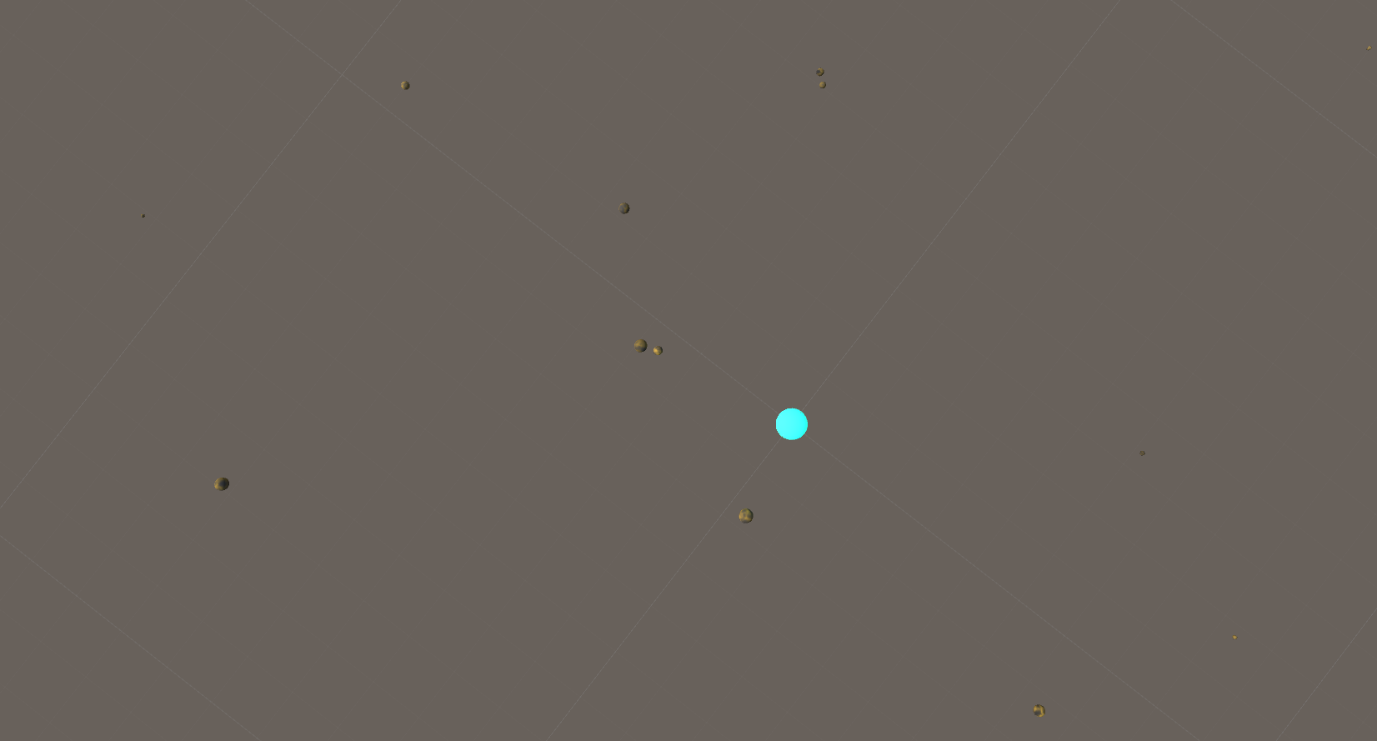
This methodology was repeated for each parameter of the barren surface subsystem to ensure each value was producing expected results, and was repeated for all planet types (See figure 20 & 21).



*Figure 20: Temperate surface and atmospheres being generated at their default/full variable ranges.*

*Figure 21: Atmosphere frequency variable being constrained to the upper quartile.*

The planet placement system was tested to ensure planets would not collide with each other and no collisions were found (See figure 22).



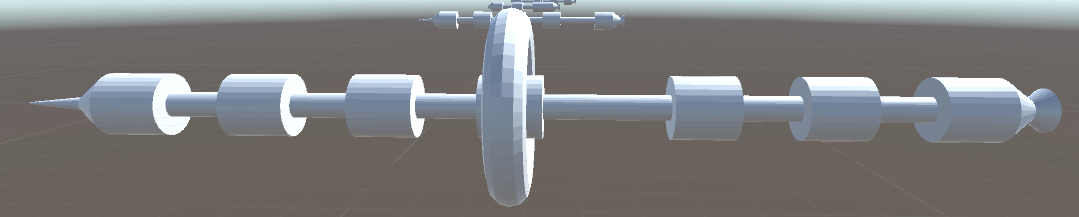
*Figure 22: Testing the lower quartile ranges of the planet offsets.*

The colony and guard ship building algorithms were tested to ensure they produced expected and realistic outputs, using the same methodology (See figure 23).



*Figure 23: Colony ships being created (top-down view) with default parameters.*

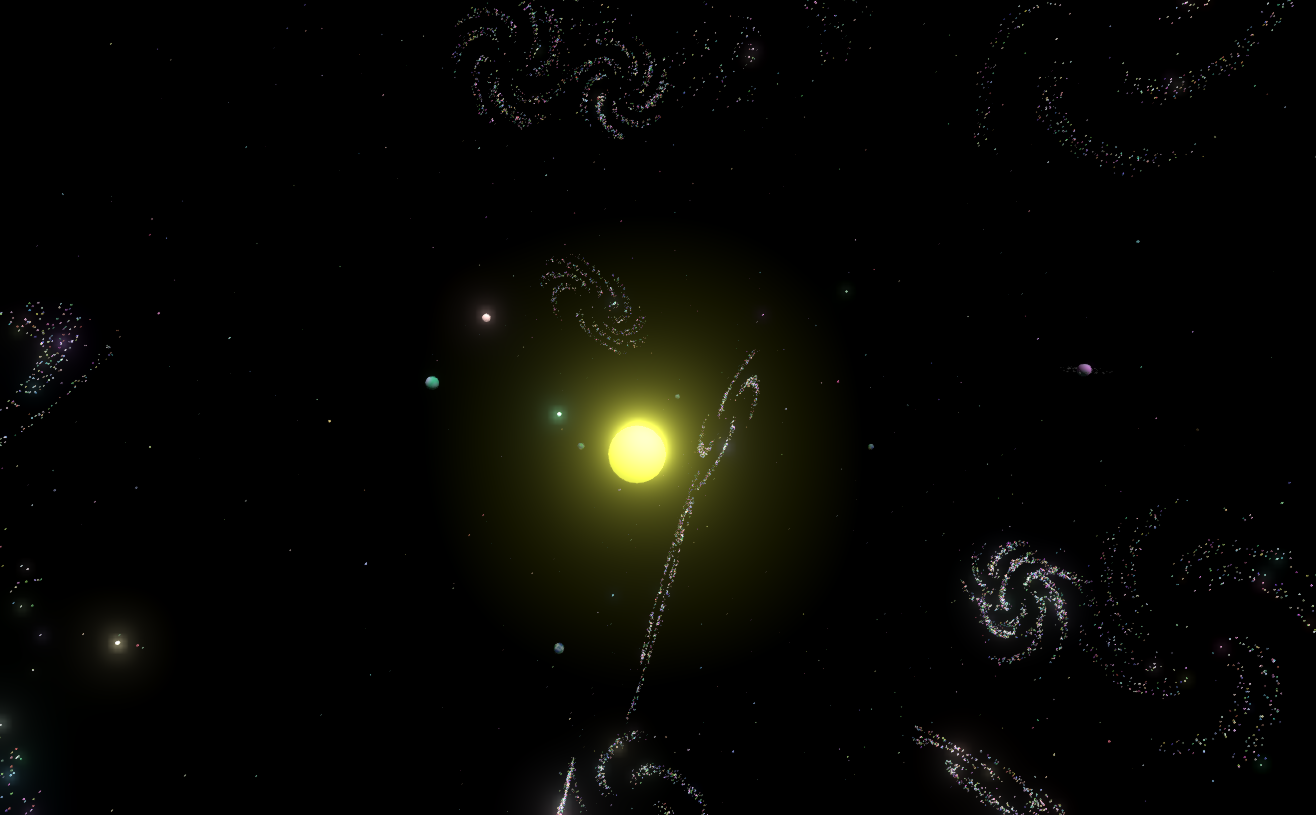
Some ships were created in ways I did not expect, some ships used two connectors after a module instead of one which was the expected behaviour (See figure 24).



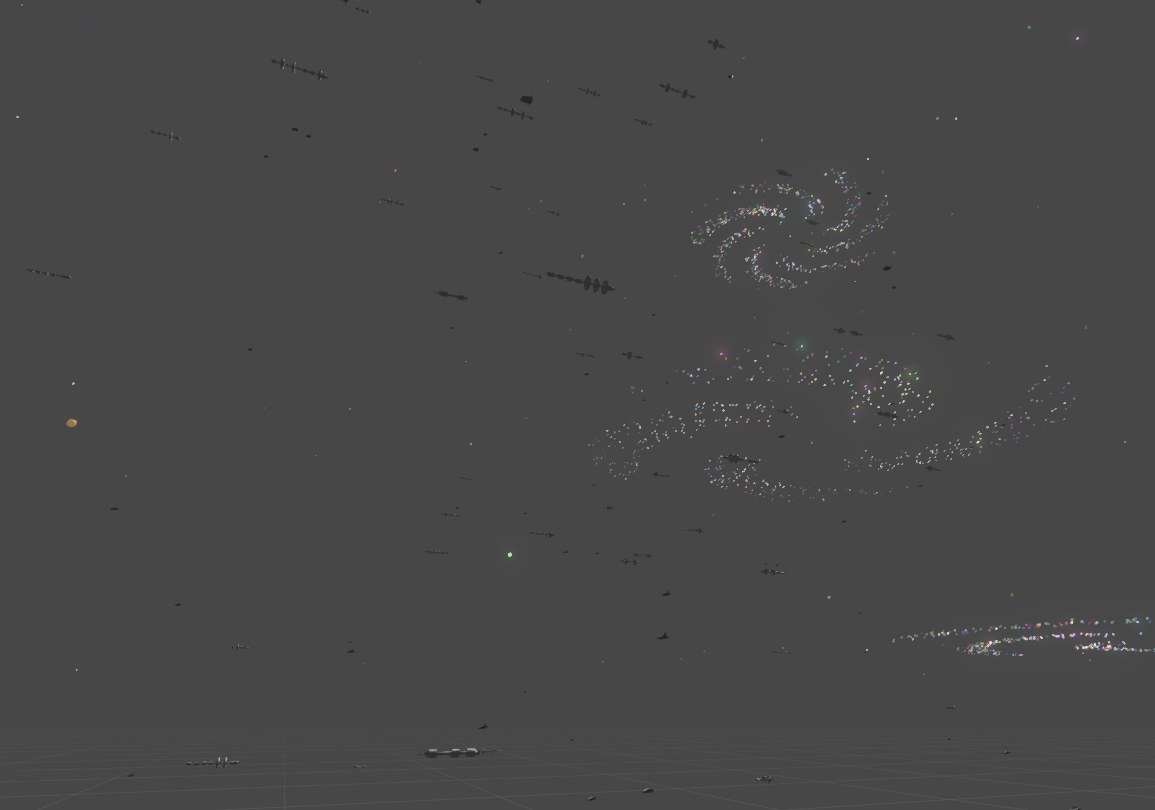
*Figure 24: A colony ship that used two connectors after the ring module.*

While unexpected, it surprisingly added more variance to the ships by making them look less uniform and breaking up their overall structure, so I left the algorithm in its current state.

I also tested the diorama holistically using the same methodology, editing planet count and fleets spawned which all produced expected results (See figure 25 & 26).



*Figure 25: Increased spawn and variance of galaxies and planets.*

**

*Figure 26: Increased fleet spawns.*

**Reflective Critique**

Overall, I am satisfied with my procedural diorama. The use of noise to create procedural textures for realistic looking planets produced realistic and surprisingly reliable results and highlighted the power of using coherent noise to solve problems. The odd/even number system produced excellent results efficiently, this system worked well for sequentially building the colony ships. The planet placement system worked well to produce a different scene each time whilst conforming to the theme.

This module has highlighted that problem-solving ability is lacking, I’ve learnt the importance of understanding different available tools and how to use them creatively to solve problems such as LibNoise. Progressing through this module helped me to look at problems for creating content in a different way and highlighted the resourcefulness of using academic research to help provide innovative solutions such as the procedural galaxy algorithm, L-Systems, and noise generation.

**References**

Bevins, J. (2003) *LibNoise* [coherent noise-generating library]. Available from <http://libnoise.sourceforge.net/> [accessed 29th April 2018].

Mendez, J.R. (2010) *LibNoise.Unity* [UnityPackage]. Available from <https://github.com/ricardojmendez/LibNoise.Unity> [accessed 29th April 2018].