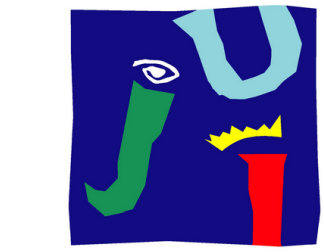


UNIVERSITAT JAUME·I

Shattered Sphere

Víctor Elvis Segrelles del Pilar Garcia

Final Degree Work
Bachelor's Degree in
Video Game Design and Development
Universitat Jaume I

June 10, 2024

Supervised by: Miguel Chover Selles.

Acknowledgments

First and foremost, I want to express my heartfelt appreciation to my mom and dad for their unwavering support throughout my entire degree journey, enabling this opportunity.

I'd also like to extend my gratitude to all the teachers who made it possible to study video games at a public institution through this degree program.

I also would like to thank Sergio Barrachina Mir and José Vte. Martí Avilés for their inspiring LaTeX template for writing the Final Degree Work report, which I have used as a starting point in writing this report.

Abstract

This document presents the project report of the Video Games Design and Development Degree Final project by Víctor Elvis Segrelles del Pilar Garcia.

This project entails the creation of an RPG Space-Sim video game, designed to apply and expand upon the knowledge acquired throughout my degree. The game emphasizes procedural generation and seamless transitions between worlds using vehicles. Additionally, it explores dialogue trees and systems leveraging Generative AI to craft voices for the dialogues. To enhance the overall experience, combat systems have been integrated into the game along with inventory systems and other RPG commodities.

Keywords: Videogame, Procedural Generation, RPG, Unity Engine, Seamless transitions, Space, Custom Gravity, Generative AI.

Contents

CHAPTER 1

INTRODUCTION

Contents

1.1 Work Motivation

The driving force behind my pursuit of this project was the desire to lay the groundwork for a video game that could be expanded upon. Over the course of four years of study, my aim was to showcase, test, and further develop the knowledge I had acquired to develop a video game. Additionally, with the recent advancements in generative AI, I was intrigued by the opportunity to integrate some of its features into the project to evaluate its applicability in a full production environment and its potential utility.

Moreover, I was motivated to undertake this project because it allowed me to concentrate exclusively on the programming aspects of video game development and to deepen my understanding of Unity. The chosen genre of the video game, a Space-Sim RPG, encompassed a wide range of technical challenges, including physics, vehicle mechanics, procedural generation, and gravitational systems. Thus, I saw this project as an ideal opportunity to immerse myself in these complexities and to master every technical aspect of game development.

1.2 Objectives

The main objective was to develop the foundation of a larger game and to test and expand my abilities related to the unity engine and video game programming while taking into account the time and the resources we had, and document it in this report, that allows us to obtain the Video Game Design and Development bachelor's degree at the Jaume I University.

Development Objectives:

- **Usage and modification of procedural content generation libraries:** To achieve immersion and variety which is crucial on this style of games. Recognizing their significance, I chose to incorporate Procedural Content Generation (PCG) libraries to aid in crafting the game world and atmosphere. This necessitated familiarizing myself with their usage and, at times, modifying them to suit the specific requirements of the game design.
- **Semi-realistic gravity:** The nature of the game required a custom gravity that would allow players to align with the center of the planet or gravity source in question. It also allowed to create ships with their own gravity fields. To develop this feature we expanded the implementation that our character controller provided.
- **Kinematic Character Controller:** To work with custom gravity we needed a kinematic controller. This controller was provided by another third party library that we had to learn to use and it was fairly complex. once we learnt the basics we implemented the custom gravity fields that the game needed.
- **Seamless transitions:** One of our main objectives was to give the player the ability to travel between planets without loading screes.
- **Large maps:** Another of our main objectives was to create relatively large maps to achieve the feel of the immensity of space. This objective proved way more challenging than anticipated due to unity's 32 bit float coordinate system but was solved for the final release.
- **Interaction with objects:** A fundamental part for many video games, interaction with objects allows us to change the world and utilize many of the game systems.
- **Vehicle piloting:** An extension of interacting with objects is the interaction with vehicles such as spaceships, we use this interaction to control and drive them as they are the method of transportation that will alow the players to travel the game world.
- **Implement a dialogue system utilizing generative Artificial Intelligence (AI):** With this objective we aimed to learn more about generative AI and test its viability of implementation.
- **Develop a functioning demo showcasing the game features:** The demo aims to show as fast as possible many of the game virtues.

1.3 Environment and Initial State

As I mentioned before, my initial interest revolved around developing a solid video game foundation that could be built upon in the future if i wanted to. I wanted to develop this project alone so that i could manage the time and also use it as a learning opportunity since one of the project personal objectives was to learn and improve my programming skills. The work would be done with the Unity game engine, and its programming would be entirely done by me, the author of the work. For the visual aspect, free and paid assets included in the Unity asset store would be used. This game remotely inherits its world and narrative design from the GDD developed in VJ 1222-Conceptual Game Design adapted to the scope of this project.

1.4 Project Access

Primary Download And Repository: Git Repository
Backup Download: OneDrive

CHAPTER 2

Planning and resources evaluation

Contents

2.1 Planning

The next lines show the tasks that have been planned ahead to build all the project. This section also includes a Gantt chart showing the same tasks in a more visual way. Some margins where considered to account for possible unpredictable issues during development. Figure 2.1

- **Research and information gathering (20h):** look for tutorials, articles, libraries or manuals for procedural generation. Look for game references to gather ideas to implement in-game. Research *Unity* navmesh and look for alternatives such as the A* pathfinding project. [11]
- **Conceptual game design (12h):** create the initial design of the game, including the story, gameplay, User Interface (UI), dialogue system [17], AI and game world. Establish the objectives based on our previously researched references and scale them based on the estimated time assigned for the project.

- **Game Core Creation (155h):** Building of the game.
  - Gameplay Core (40h): study and implement the character controller and the gravity aligned movement, develop the camera and the Canvas UI, implement the respawn system, the inventory system.
  - Procedural World creation (40h): learn to use the SGT (Space Graphics Toolkit) library [24] and create the procedural worlds adjusting them to our requirements of level design and art design.
  - Starship Systems (30h): create all starship related systems, including piloting, health and shields, turrets and fixed weapons and weapons types, damage interfaces, hangar systems, warp fields.
  - Dialogue system (15h): understand and implement the "pixelcrushers" dialogue system, generate the voices with AI, write the dialogues and setup the non-player characters (NPCs).
  - Waypoint system (10h): implement a navigational, quest and enemy marker waypoint system and select sprites to represent each waypoint, setup the waypoint manager and waypoint canvas.
  - AI space(10h): creation of a basic but functional AI for spaceships and turrets as well as a fleet AI for larger fleets.
  - AI ground (10h): research the implementation of the A* pathfinding project custom navmesh and agents, ground turrets.
- **Tutorial level building (20h):** the building of the tutorial/demo scene.
- **Asset search and creation (30h):** [13] [6] [7] [8] [21] look for appropriate assets for our game.
- **Documentation (20h):** Create the Work report demo and presentation.
- **Error Margin (30h):** Extra time reserved for unexpected complications or extra features.

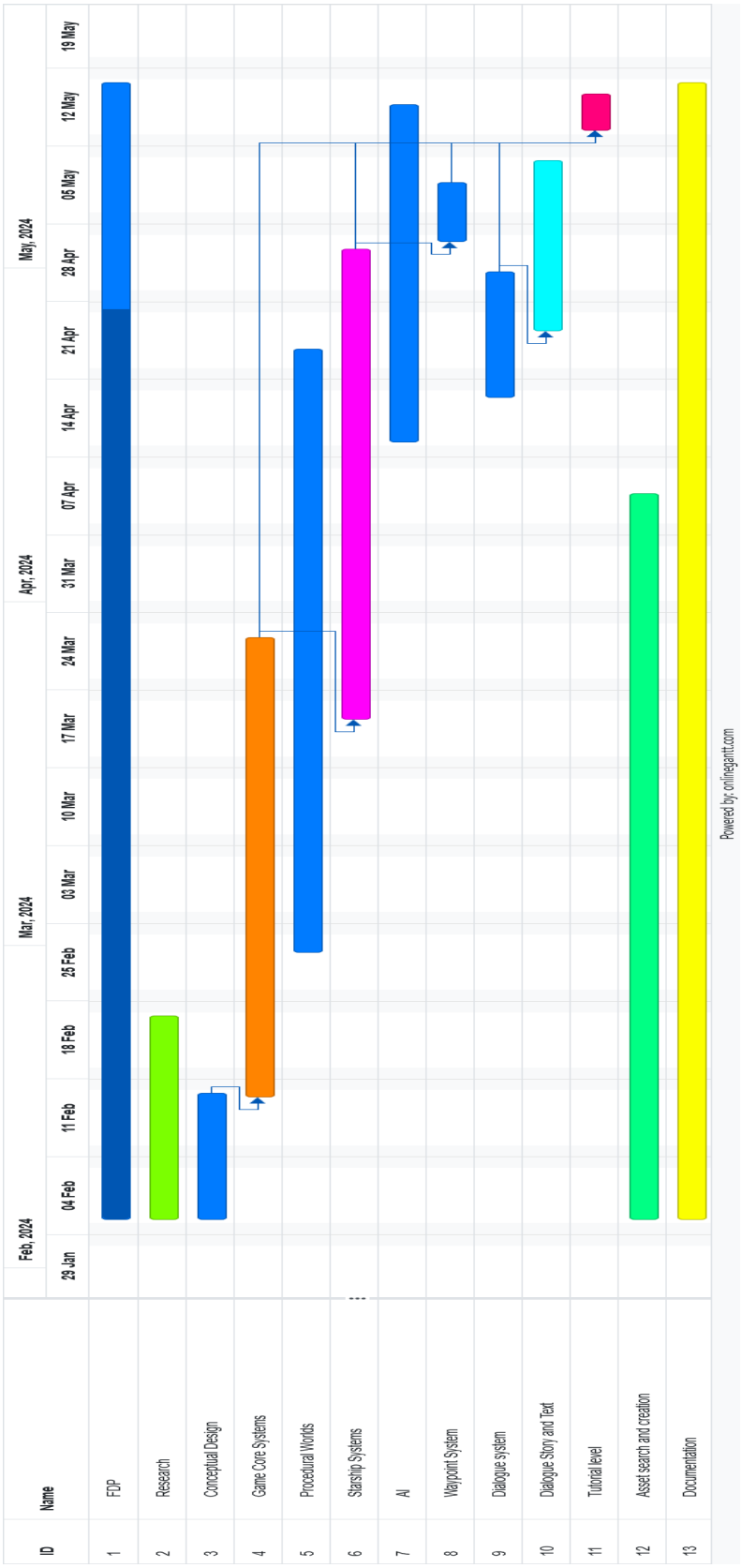


Figure 2.1: Planning Gantt chart [2]

2.2 Development

The next lines show the tasks that have been done during development to build all the project, some of which were not planned. Not all the tasks were done one by one (finishing one before starting the next), but some were made alternating between them. On this section times will be adjusted to the real development and non planned tasks will be described.
This section also includes a Gantt chart showing the same tasks in a more visual way. Figure 2.2

Research and information gathering (25h)
Conceptual game design (20h)
Game Core Creation (200h)
- Gameplay Core (50h): Implement the floating point origin.
- Procedural World creation (35h)
- Starship Systems (50h)
- Dialogue system (20h)
- Waypoint system (15h)
- AI space(15h)
- AI ground (15h)

Tutorial level building (20h)
Asset search and creation (20h): Adjust some 3d models in blender [20], adjust some 2d sprites in krita [10].
Documentation (40h): Create the *LaTeX* [16] Work Report, the presentation and compile the tutorial demo for the exposition.

2.2.1 Development conclusion related to the planning

The planning was fairly accurate, maybe a bit optimistic for most tasks except the starship systems and documentation. The starship systems experienced the difficulty of the hangar system and scene management while the documentation didn't plan for the usage of *LaTeX*, other complications such as finding out that *Unity*'s 32 bit floating positions would cause problems also caused delays on development. However the planning already accounted for such possibilities in the form of a margin of error that covered most of the delays.

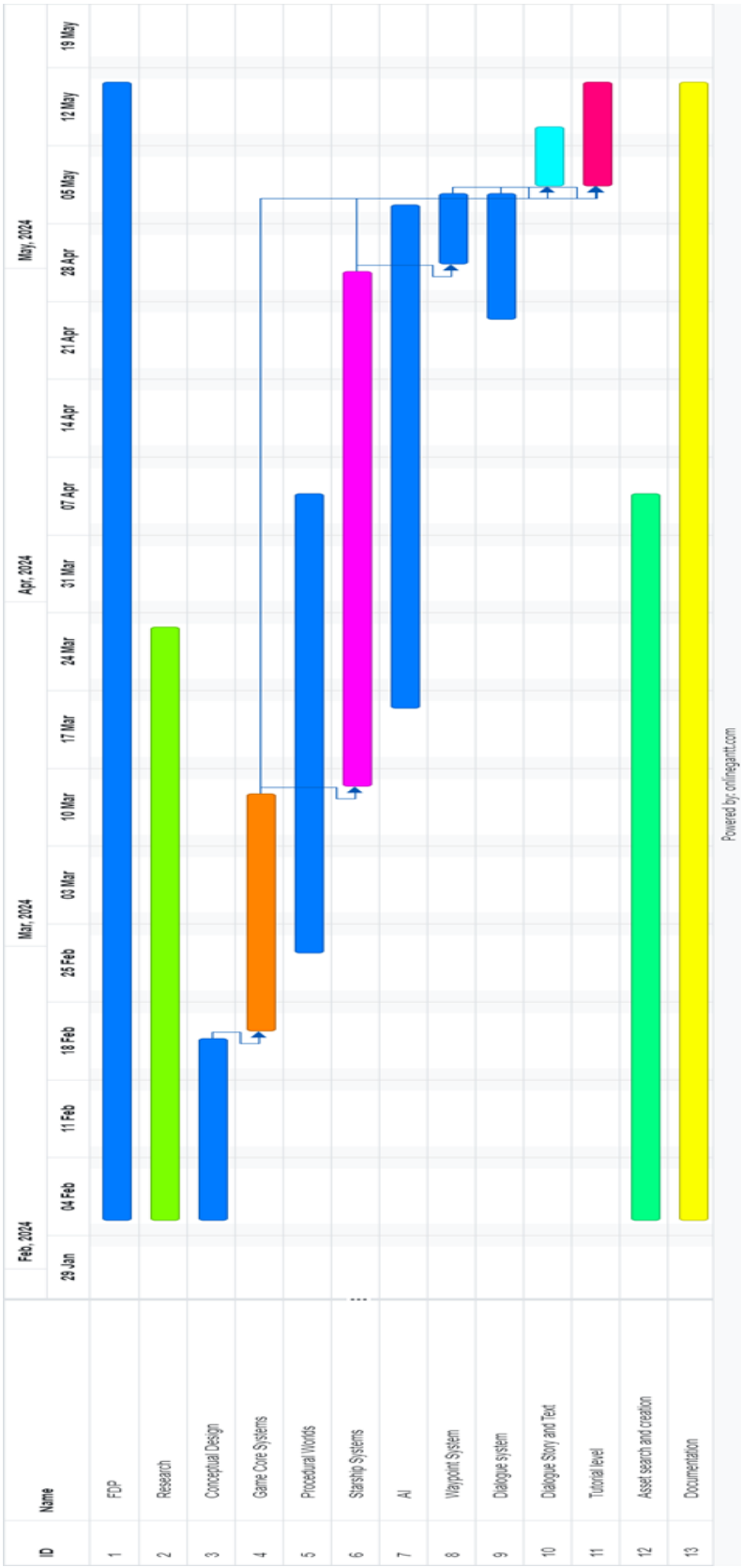


Figure 2.2: Development Gantt chart

2.3 Resource Evaluation

Analysis of the theoretical project development costs.

The number of hours dedicated to the project were 330 in total, divided on a schedule of 6h per day 5 day a week except at the final stages of the project that it switched to 6 days a week for additional bug-fixing and polishing. In regard to the economic costs of the project, the average wage per hour of a programmer working on Spain is 14.5€ [1]. This results in a labor cost of 4785€ for the project. Additionally, we have to factor in the price of the tools and graphical assets used for our project. It's important to note that the costs might not reflect bundled prices in regards to assets.

Hardware Tools: The physical tools used for the development of the project. The project was made on different machines due to mobility and availability of these, however this is not strictly necessary, and we should only factor the cost of one of the machines used.

The cost of the project, including labor, software, and hardware prorated over the three months of development, would amount to 5266€.

2.3.1 Development Systems

CPU: AMD Ryzen 9 3950x /AMD Ryzen 7 5800x3d /Intel Core i7-10870h. Cost of purchase (AMD Ryzen 7 5800x3d) 430€.
GPU: Nvidia RTX 3090 24GB (AI) /Nvidia RTX 3070 8GB. The usage of the RTX 3090 24GB allowed us to load larger AI models for voice creation, however this step alone might not warrant the investment on a more expensive GPU as the RTX 3070 was perfectly capable of loading "good enough" models. Cost of purchase (RTX 3070) 500€.
RAM: 32GB DDR4 120€

2.3.2 Software Tools

- **Unity (Free)** [22] [23]: Game engine developed by Unity Technologies to create 2D and 3D games.
- **Visual Studio(free)** [12]: It is an IDE developed by Microsoft that is compatible with almost every highly used programming language, like C sharp, C++ or .NET. It is pretty easy to use with Unity.
- **Gogs(free)** [3]: Self-hosted Git service.
- **Sourcetree(free)** [5]: Free Git GUI.
- **Krita(free)** [10]: Digital painting and Editing.
- **Blender(free)** [20]: 3D content creation suite.
- **Overleaf(free)** [16]: LaTeX editor.
- **Windows 10 pro (199$)**: Microsoft WIndows Operative System.
- **Adobe Substance 3D Collection(49.99$/month)** [4]: suite of smart creative apps and high-end content that gives artists everything they need to create 3D digital content.
- **Assets**: Some paid unity libraries and assets were used on the project, and we should account their license cost.

Chapter 3

Game Design Document

Contents

This chapter presents the original Game Design Document (GDD), this document details the original plans for the game and its objectives and specifications of design. This document details the finished product objective not just the foundations of the game that were our objective for the Final Degree Work so that latter on the project could be completed by crafting the additional planets dialogues and narrative.
The scope of this Final Degree Work was to implement all that would be required to develop this final objective but without having to spend time on carefully crafting a hundred star systems and thousands of lines of dialogues. Completing the project specified on this document would only require more time and doing repetitive tasks instead of developing aditional game systems.

3.1 One Page

Target audience: Videogame players with special focus on RPG players or sci-fi enthusiasts.

Target age: 13 +
Platform: PC
Genre: RPG
Developer: Víctor Elvis Segrelles del Pilar García

3.2 Game Description

Lore and cultural description

Roleplaying game set on a futuristic sci-fi universe where human society has far evolved beyond our current technological capabilities. The society represented in the game will be a dystopian autocratic imperial society going in line with many of the most famous sci-fi literature, it is a well-known and tested plot design choice for sci-fi and RPG games alike to base their main story on the hero's journey. To add interactivity, we'll allow the player extra freedom of choice in line with true RPG games that will alter and branch the story and the state of the world based on player's choices.

World Description

The worlds visited during the game will try to mimic a semi realistic universe, emptiness of space, stars, barren worlds and asteroids will dominate. There will be a few "habitable worlds" with vegetation, but they will be rare to find. Players will also find artificial constructs and worlds.

Technology description

The technological level of the in-game universe will allow for relatively easy access to space travel and FTL interplanetary travels along with advanced weapons and protection common to many sci-fi universes.

3.3 Narrative and Story

3.3.1 Plot Summary

In the vast expanse of the galaxy lies the Imperium, a sprawling dominion governed by Emperor Xerathius with an iron grip. Among its ranks, Commander Talya serves faithfully, executing missions with deadly precision in the shadows of the empire. When tasked with investigating a burgeoning rebellion, she finds herself ensnared in treachery as her trusted allies betray her during an attack on a rebel outpost. Outnumbered and facing imminent death, Talya's consciousness flickers on the brink of oblivion, only to awaken on a mysterious distant world. With fragmented memories and a will to survive, she embarks on a journey where players can shape her destiny. Whether seeking vengeance against her betrayers or forging a new path, the player is free to navigate Talya's story in an open world of possibilities.

3.3.2 References:

Movies: *Star Wars*, *Star Trek*, *Dune*.
Books: *Star Wars*, *Dune*, *Une aventure de John Difool*, *Warhammer 40k*.
Videogames *Star Wars*, *Star Trek*, *Mass Effect*, *Stellaris*.

3.3.3 Narrative Description

The narrative structure in open-world games is quite unique compared to more linear forms of storytelling found in traditional media like books or movies. Open-world games provide players with vast virtual landscapes to explore, often with minimal constraints on where they can go and what they can do. This freedom creates both opportunities and challenges for crafting narratives.

- **Non-linear storytelling:** Branching storylines, where player choices influence the direction and outcome of the narrative.
- **Emergent narratives:** Unexpected events and interactions occur as a result of the game's systems and mechanics. These narratives often arise organically from the player's actions, leading to memorable and unique experiences.
- **Player agency:** Allowing players to shape the narrative through their actions and decisions. This can range from simple choices in dialogue trees to major decisions that alter the course of the story. Player agency is crucial for creating a sense immersion and investment in the game world, as players feel like active participants in the narrative rather than passive observers.
- **Reactive storytelling:** The game world dynamically responds to the player's actions. This can include changes in the environment, the behavior of NPCs (non-player characters), or the availability of quests and resources.

3.4 Game Rules

3.4.1 Design decisions

- **Objectives:** The goal is to achieve one of the endings.
- **Difficulty:** The difficulty will depend on the game area and story point.
- **Variety of Objects:** There are different types of space vehicles, weapons (heavy, medium, light, swords), equipment (protections, tools, etc.), components (ship shields, star maps, etc.).
- **Variety of Decisions:** different decisions and policies that produce different results, depending on the player's playstyle they are not inherently good nor bad.

- **Complexity:** The player will have to consider the area they are in, how to deal with their rivals, either diplomatically or by force, how to proceed with their story, how to invest their resources.

3.4.2 Operational game rules

- The player controls vehicles, characters, and ships.
- The behaviour of objects is predefined.
- The behaviour of NPCs is initially predefined, however, according to our decisions as a player, we can alter it.
- The game contains a global situation system, this will modify many rules and mechanics depending on the state of the world (at war, at peace, etc.).
- The game has an internal economy that depends on the planet (prices can vary between planets), and the global situation of the game.
- Combat is lost when the amount of life decreases to 0.
- The player's goal is to complete the story with one of its endings.

3.4.3 Foundational game rules

- Scenarios are procedurally generated for secondary and main planets; additional details are handmade for main story points.
- Each area has its own gravity, planets having spherical gravitational zones while spaceships can have different shapes of gravity fields according to their interior configurations.
- Projectiles and laser weapons damage the player and their vehicles, potentially destroying them.
- The universe is divided into control zones, which could be interpreted as "countries".
- The energy of structures and vehicles is consumed according to the structure or vehicle in question.

3.5 Mechanics

3.5.1 Progression Mechanics

Progression through the game universe will depend on the player's equipment level, as each area will have a recommended level. While the main story will take us to areas that match the equipment level we should have at that point in the story nothing will prevent the player from getting into hugely disadvantageous situations.

3.5.2 Economic Mechanics

- We will have purely economic resources such as money which we can use to improve weapons, armor, ships, etc.
- We will have resources in vehicles and ships, such as shields (with limited capacity), energy, and weapon ammunition.
- Our character will have a limited amount of hit points. We will also have a limited amount of life, both in our character and in vehicles.
- Shields regenerate with time, hit points have to be healed or repaired.

3.5.3 Relational Mechanics

- The player will be able to talk to most NPCs with basic interaction.
- NPCs may give the player secondary missions and rewards.
- The player will be able to build relationships and create friendships with some NPCs from the main story or secondary missions who may join the player's cause granting bonuses or assisting in combat depending on our decisions (rebellion system).

3.5.4 Strategic World Mechanics

The game universe will be composed of different planetary systems that we can travel through, which, in turn, have multiple planets, moons, and stations that we can visit.

3.5.5 Base Mechanics

- The player will be able to move within a 3D space.
- The player will be able to take basic actions such as interacting with objects and NPCs.
- The player will be able to commerce to buy or sell equipment and vehicles.
- The player will be able to control vehicles with different movement schemes and resource management (energy, fixed turrets, etc.).

3.5.6 Rebellion Mechanic

The rebellion is the player's open world default faction, it opposes the central autocratic government by default, although it can change. The rebellion has some key attributes that last the whole game duration:

- **Resources:** money to buy equipment and items.

- **Popular opinion:** how the galaxy sees the rebellion, this will affect all the other attributes indirectly.
- **Awareness:** it determines how much effort the empire will put in fighting the rebellion, higher awareness means that the player will have more troubles freely moving through imperial controlled space such as bigger patrols or planetary blockades.
- **Allied forces:** it determines the number of forces that support the rebellion's cause, it is linked to popular opinion but having a lower opinion may not necessarily mean losing on this value, certain forces are required to hold sectors.
- **Controlled space:** systems claimed by the rebellion, giving special discounts and adding their strength to the rebellion, claiming sectors increases awareness considerably.

3.5.7 Galactic State Mechanic

The galaxy is divided into several sectors (at first controlled by the empire), this can change by the player's actions with the rebellion. The galaxy also has an overall state, it will start with the "peaceful" state, but it can deteriorate depending on the player's actions.

3.6 Game Balance

3.6.1 Intransitive Relationships

Weapons and defensive systems are effective against specific types.

- Plasma laser weapons are ideal for shredding armor and health points, but ineffective against shields.
- Ion weapons specialize in nullifying shields but are not capable of damaging health points.
- Missiles are balanced, they do not deal extra damage against anything, but they are not weak against anything either and are generally slower.

3.6.2 Transitive Relationships

Weapons and defensive systems consume resources or ammunition. This prevents dominant strategies since having a large quantity of identical weapons, specifically efficient against an enemy, depletes their energy very quickly. This is perfect for "hit and run" strategies but bad for longer battles. A balanced configuration allows for longer encounters and being neither weak or strong against any opponent.

3.6.3 Difficulty Progression

Due to the "open world" nature of the game, it is the difficulty itself that guides the player through different levels and places, as nothing prevents the player from attacking a fortified system directly, although with the initial ship, the chances of defeating even the weakest are null. The difficulty of enemies does not change, but as the player becomes stronger, it is easier to defeat them. Each sector will be locked behind its fortress as its hyper gate will be shut down until the player can enable it.

However, the difficulty curve is designed so that the player's power scaling is lower than the scaling of enemies, a difference that the player must compensate for with the skill acquired during the game. The greatest difficulty will be in the final chapter, where all game systems will have an effect.

3.7 Gameplay

The level hierarchy of the game is structured into 3 phases. In the first phase, we will play the introduction, which will lead us to the open world of the game; the second phase is about the game world, in this phase there is no specific order or hierarchy, the game leaves it up to the player decision when to enter phase 3, which would be the final phase and conclusion of the story.

In the game, there are a few common gameplay loops.

- Combat missions that challenge the player's mental and physical skills, combat challenges.
- Exploration missions where the player must find secret entrances or discover hidden relics.
- Diplomatic missions where the player must speak/convince NPCs or trade with NPCs.
- Key story missions that will alter the game outcome.

Atomic challenges we will encounter.

- Shooting enemies (ground or space): hitting or missing.
- Dodging obstacles (ground or space): avoiding barriers, jumping walls, dodging asteroids.
- Activating switches/interacting with world elements.
- Finding an NPC (ground): open world or combat.
- Controlling a turret or vehicle (ground or space): open world or combat.
- Healing ally/repairing ally (space or ground): restoring health points of an allied NPC depending on our equipment.

3.8 Level Design

3.8.1 Planets

Uninhabited planets: These planets do not have any permanent settlements although it is possible to find mining outposts or small isolated facilities. These planets are abundant and generally go unnoticed, but they lack infrastructure and may be too far from trade routes to be relevant. Figure 3.1

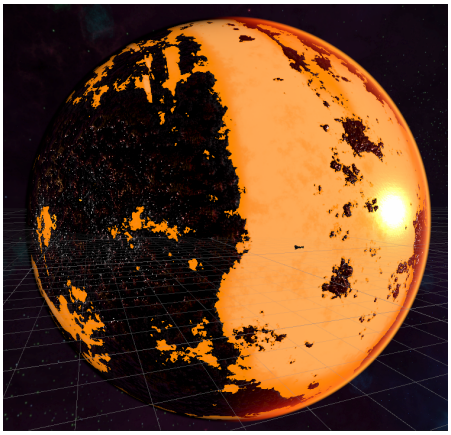


Figure 3.1: Uninhabited planets

Colonized planets: These planets have permanent settlements that usually have all kinds of basic vendor services, research facilities, NPCs with secondary missions, etc. These planets are relatively rare and are not actively monitored by the government. Figure 3.2

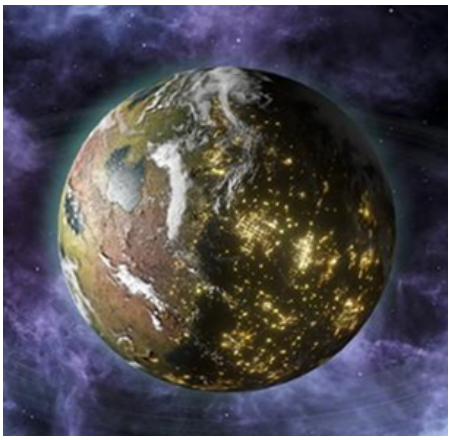


Figure 3.2: Colonized planets

Urban capital planets: These are large city planets, they contain all kinds of research facilities, refineries, vendors of all kinds, and high-quality (and expensive) products, vehicles and ships. These planets are highly monitored, and it is difficult not to attract attention. Corporations have their headquarters on them, and there are only a few in the entire galaxy. Players can obtain the highest quality of ships on this planets. However if the imperial awareness is high it might be difficult to avoid combat on this planets. 3.3

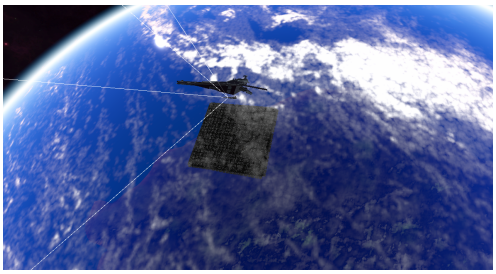


Figure 3.3: Urban planet large city

Dyson Sphere World: The imperial capital at the center of the galaxy and final point of many story paths. It can only be accessed via hypergate and the gate will be heavily defended, it contains the imperial palace and government structure. It's the final level of the game. Figure 3.4

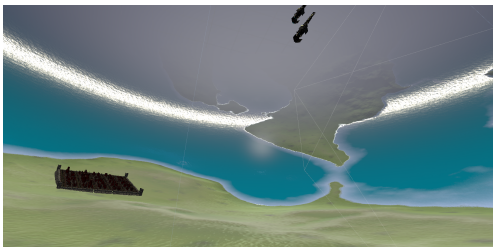


Figure 3.4: Dyson sphere reverse world

3.8.2 Moons

Just like with planets, there can be uninhabited, colonized or urban moons, which are smaller versions of planets, however, there is an exclusive type for moons.
Bastion moon: these are military fortifications designed to maintain order in different sectors, they are extremely well defended and exert control over the sector they are present preventing it from being claimed, they are present around the capital planet of each sector guarding the hypergate that communicates sectors. Figure 3.5

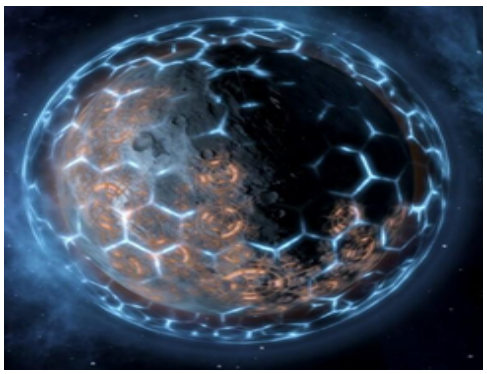


Figure 3.5: Bastion moon

3.8.3 Space Stations

Asteroid station: these are asteroid mining outposts; they only have basic services but can be used to buy basic equipment unnoticed.
Research stations: these are stations generally close to gas giants where we can buy some advanced components or normal technology.
Military stations: these are much smaller defensive installations than bastion moons, they have a considerable variety of weapons and, despite being smaller, are not poorly defended.
Trade stations: this is the perfect place to buy and sell basic resources. They are by far the most common among stations. 3.6
Hypergates: they allow long distance travel between sectors. Figure 3.7

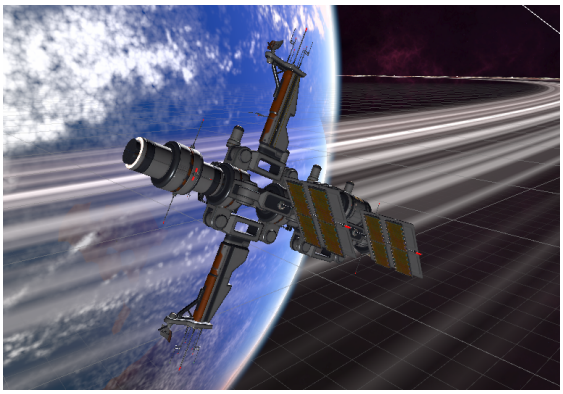


Figure 3.6: Trade station

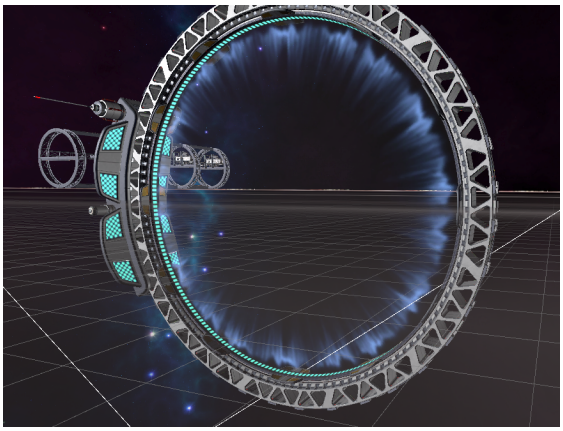


Figure 3.7: Hypergate

3.9 Graphics and Styling

The game will implement realistic graphics but will contain scaled down planets (still massive but not 1:1 planetary sized, this is both for performance and style reasons as such a massive planet will mostly be wasted space). High poly geometry and assets have been used whenever necessary to increase immersion but also limited their amount as we wanted the project to still be performant.

Spaceship and station design will be diverse to represent the multiple starship manufacturers present on the galaxy, ranging from more traditional designs to some exotic constructions that take advantage of gravity fields within the ship.

3.9.1 Lighting

The lighting on this project from the artistic standpoint is very important and necessary for the atmospheric feel. The light on a planet surface depends on the atmosphere so it scatters based on the atmosphere configuration Figure 3.8 Figure 3.9, going to orbit changes lighting to a standard space lighting where light proceeds from a single point (the star) of the system, combined with the skybox that gives some systems a characteristic tonality.

Night Sky: on the side of the planet were the star light can't reach the atmospheric scatter pretty much disappears allowing us to see the night sky and more easily see ships or other bodies in orbit. Figure 3.10

Interior Lighting: The interior light mainly depends on the building or ship design, this is intentional as each ship manufacturer would have its characteristic style.

Projectiles also emit light, this not only helps with visualization in the blackness of space but helps with the sci-fi immersion.

Shadows(art): Initially it was decided to allow for very long shadow distance, however this would bring limited visual benefits at the expense of a massive performance hit. Shadow render distance was reduced as a solution.

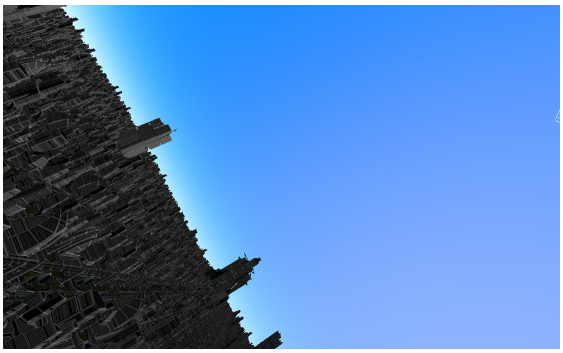


Figure 3.8: Atmospheric light scatter

Figure 3.9: Atmospheric light scatter on a molten world

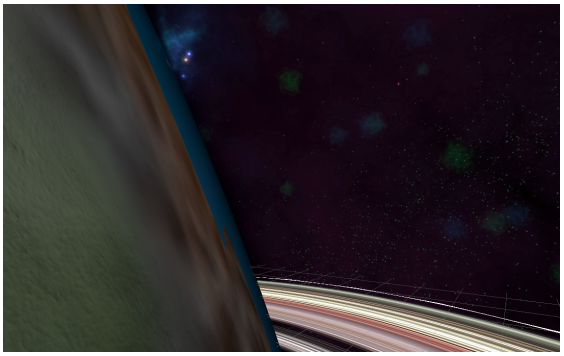


Figure 3.10: Night Sky

3.10 Music and Sound

Soundtrack: Calm instrumental music will be playing during the game; some systems will have specific tracks to increase immersion. In case of combat the music will be overridden by combat music.
Sound effects: will contain but not be limited to door sounds, engine sounds, shield and energy weapons sounds and many other immersive effects.

3.11 User Interface

The user interface will show current health, shields (if present) current weapon and waypoint markers, the interface will be different for ships and the player, having the ship interface additional things such as speed indicators and energy distribution indicators.

3.12 Tools Used

Blender: used for editing some 3D models to better fit our game and to simplify some meshes to gain performance.
Unity: The game engine used to develop this project.
Nvidia CUDA: used for generative AI dialogue voice creation.
Python: used for generative AI dialogue voice creation.
Microsoft Windows OS: used to develop the game.
Gogs git: self-hosted git service that allows us to skip the 100MB filesize limitations on github.
Atlassian Trello: used to keep track and plan ahead of the project development.
Visual Studio: the main IDE used to develop the project.

Chapter 4

Functional and Technical Specification

Contents

In this section, the different parts of the game will be explained, that is: game design, level design, core systems, gameplay mechanics, user interface, user experience, sound design and art design.

4.1 Multi-Scene Workflow

A multi scene workflow was used for the game development, this decision was made after thinking how to implement a persistent state through multiple scenes and/or very large ones.

The persistent scene: This scene contains the game manager, the player, the cameras, the UI, the dialogue system, and the spaceships. This was made so that the game remembers the position of each spaceship even if the player travels across star systems (scenes). The idea was to be able to travel back and forth while having your ships at the exact place you left them, and to be able to move ships between systems. This persistent scene also contains all data worth saving.

Star Systems

These scenes contain the scenery, environment, NPCs, pathfinding and everything related to that specific level of the game.

Menu

The menu scene and galaxy menu scene contain the main menu and in-game traveling menu.

4.2 Game Core Systems

This encases the essential game functionalities to create the world and gameplay. They are essential for all the other game components and the overall feel of the game.

4.2.1 Gravity Alignment System

Design:

The game world will override the default *Unity* Y vertical downwards gravity, this is a key requirement for the gameplay experience we intend to provide the player of a vast and open universe. With the gravity alignment the player character will be able to walk on planets or ships with gravity fields no matter their orientation. Figure 4.1

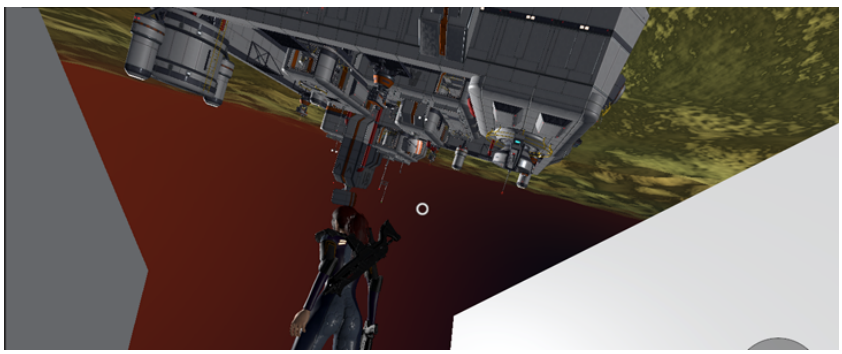


Figure 4.1: Gravity alignment

Implementation:

To implement this system, a module that was available with the Opsive UCC package was used, this package gave the player an ability (which will be explained in detail latter) to align with a direction. Using this ability, 2 shapes of gravity forms have been implemented.

- The spherical gravity zone: this will align the player towards its center and its ideal for planets and other realistic gravity sources.

- The Linear gravity zone: this will emulate gravity in a specific direction, used for space stations and starship gravity fields.

Both gravity zones work the same way. Additionally, both zones implement an intensity value that will determine their strength, this will prevent a planet gravity ripping the player from its ship should both zones overlap.

The zones consist of a large trigger where the effect takes place and there's no gravity outside of the trigger area. This was changed from a starting version that had no limit and used a more realistic gravity formula, while more accurate as gravity has no "range" in real life, it was worse both in performance and user experience.

Gravity zones also contain a specific script that changes a player state and enables or disables its gravity alignment ability.

4.2.2 Scene partitioning, Floating origin and Chunk division

Design:

Star systems are huge, and to give the player the sense of a massive empty space objects should be positioned very far away from each other while being massive enough to feel like planets or moons.

This size requirement led to a problem with *Unity*'s transform limitations. *Unity* transform values are stored in floats that are 32 bits, this means that the game loses precision the further form the origin the player goes, past 100.000 units this system starts to fail and the game jitters while physics start to break.

In a space game, 100,000 units is almost nothing. Assuming 1 unit equals 1 meter, this translates to around 100 km on each axis, which is insufficient for the vastness of space and won't allow us to deliver the desired player experience.

A workaround reducing the game scale was first imagined, but soon discarded as reducing the scale would do nothing as floating precision will still be lost, and on a reduced scale it will become noticeable sooner removing any potential gain for staying at lower coordinates.

The chunk floating origin:

The floating origin solution was implemented after discarding other ideas, this solution divides the scene into multiple chunks, and once the player crosses to another chunk the scene origin gets moved to it, this way the 100.000 units are never exceeded but the player gets the feel of vastness. Figure 4.2

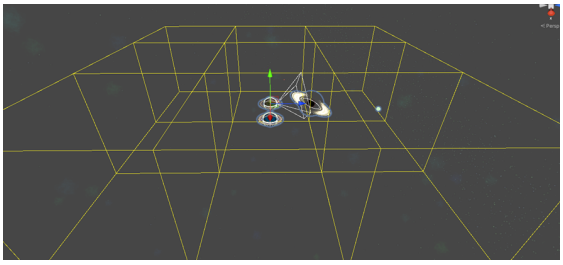


Figure 4.2: Chunk Division

Chunk floating origin or floating origin: the difference between this system and a pure floating origin is that only the world gets moved around the player a few times (on chunk change) as opposed to constantly.

Implementation:

To implement a floating origin, scripts were added to keep track of chunk coordinates. The elements of the map are divided into various chunks containing their coordinates, while the player and ships keep track of their current chunk positions. When the player crosses a chunk border, their chunk position is updated and the origin of coordinates is moved, effectively shifting the entire scene around the player.

With this floating origin chunk system, the maximum number of chunks can be multiplied by the usable coordinate range (100,000 units), resulting in a total of roughly 2*10$\hat{1}$1 km in each direction of usable space. This range is significantly more than needed.

To update and move the chunks around the player, each chunk listens to an event to reposition itself relative to the player's current chunk when the player crosses a chunk border. This entire world teleportation process is transparent to the player.

Limitations:

A floating origin system however has some limitations in use, the use static objects is impossible as they can't be moved on execution.

4.2.3 Layer-based Camera Clipping Planes

Design:

The camera clipping planes are adjusted to provide an easy form of culling, each layer has a camera distance, most of the layers have a set distance of around 4-5000 while the "planets" layer and "space massive" layers have a much larger distance so planets and stars can be seen from afar without having to render the buildings on them.

This was used in conjunction with Level of Details (LODs) to improve performance and to provide a fast way to cull objects that didn't implement a LOD group. This solution is inferior to adding LODs to every single object, however its orders of magnitude faster to develop and performant enough.

Implementation:

To implement this, cameras have a custom script that sets the layer cull distance according to a configuration, each camera has its own configuration so that the player and spaceship camera can be configured individually.

4.2.4 Layer Mask Manager

Implementation:

This simple script allows to easily configure the collision and render layers of different objects it's attached to.

4.2.5 Ship Manager

Implementation:

Since it was designed to keep ships on the persistent scene and to remember their position this script handles that, the position of the ships, the actual player ship (last one entered or currently driving ship).

The ship manager contains all the ships and their current scenes, and handles disabling or enabling them accordingly to the current loaded star system, it also handles traveling between the star systems with the ships and any docked ships (important so that large ships with fighters don't lose the fighters when traveling between systems).

Ships add themselves to this manager and it can hold a large amount of ships without problem as well as any amount of docked ships on other ships.

4.2.6 Scene Manager

Design:

To simplify game organization and separate the game functionality over the levels the game uses a multi scene workflow, this workflow means that loading scenes and unloading them is a bit different.

Implementation:

This scene manager loads the star systems, to do so it additively loads the desired system that it takes as a parameter and unloads the current loaded one. Once the loading and unloading is complete the new star system is set as the active scene so that its skybox and other factors load properly. The persistent scene is never the active scene.

4.2.7 Event System

Design:

The custom event system allows scripts to subscribe to generated events and trigger these events from anywhere else in the game, allowing for more flexibility and possible performance optimizations. One of such examples is the keyboard events script that monitors keypresses and sends events accordingly, this is done on a single script update cycle and it's much more efficient and readable than checking for a keypress on a lot of GameObjects and different scripts.

Implementation:

There's a custom Event system singleton with 3 main public functions and a dictionary to hold all the events. The 3 functions allow scripts to subscribe, unsubscribe and trigger the event by its dictionary key.

The functions also allow to specify a callback function that will be called if the event is triggered, it can be different for each script subscribed to an event.

4.3 Game World Creation

This section will explain in detail how the game world and scenes are created.

4.3.1 Planetary Body Creation

This subsection explains the creation of planets and all the components of its modular design. Planet creation is completely procedural and can be done in 2 different ways. Some components are required while others optional.

The planets are created using the SGT (Space Graphics Toolkit) library and adding some modifications whenever necessary.

Terrain Creation

This is a necessary step for planet creation, requiring terrain to determine the shape of the surface. Two methods have been implemented to achieve this.

- SGT Terrain (contains necessary components): A more modular approach, it's harder to use but allows greater customization. Figure 4.3 Figure 4.4
  - SGT Terrain Planet: It creates the base shape of the planet and controls the procedural LODs, it can be configured with different values:
    - Radius: the radius of the planet.
    - Smallest triangle size, detail and resolution: how detailed the surface should be at each LOD.
    - Observer: the entity used to calculate LOD distances.
    - Baked Detail Tiling: bakes texture details into the planet procedural mesh.
    - Areas: this is a configurable preset that can divide the planet into different areas (like grass, snow, desert).
  - SGT Terrain Planet material: contains the base material that will be used for the planet creation:
    - Material: the material/texture used.
    - Water(optional): handles the oceans, and lakes, it requires other components to function.
  - SGT Terrain shared material (optional): this script improves the visibility and quality of the atmosphere.
  - SGT Terrain heightmap (optional): this script provides the planet surface with a heightmap with a height texture and a displacement configuration, it allows for the creation of mountains and valleys on the planets.

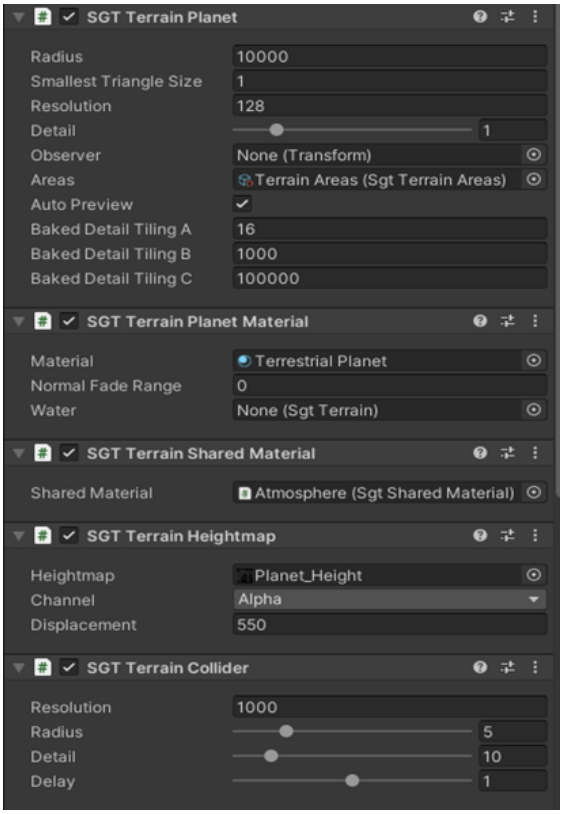


Figure 4.3: SGT Terrain Script View

– SGT Terrain collider (optional): creates the colliders for the procedurally generated mesh around the observer configured on the SGT Terrain script.
  * Resolution: in how many chunks should the planet surface be divided.
  * Radius: how many chunks of planet terrain should have their colliders generated around the player, setting this too low can lead to the player falling through the ground, and setting it too high can lead to unnecessary collider creation.
  * Detail: how precise should the colliders be in regards to the planet terrain mesh.

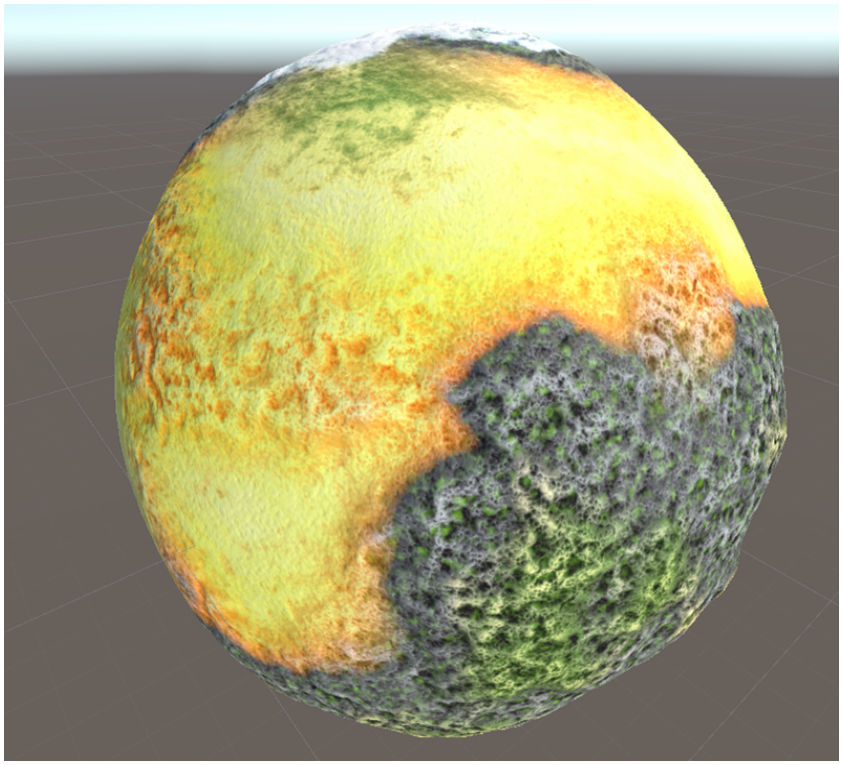


Figure 4.4: SGT Terrain Planet

– SGT Terrain prefab spawner: custom script that spawns specified prefabs on the planet surface, it can be used to procedurally spawn any construct on the planet's surface, this script can be added multiple times to. For example, spawn trees on "grass areas" and palm trees on "beach areas". Figure 4.5

  * Area: the area this script should work on, it uses the areas defined by SGT terrain planet, this can be used to prevent trees from spawning on a desert or to create biomes with different plants or buildings.
  * Threshold: this softens the area borders preventing trees and other prefabs from spawning right at the area edges in full amount.
  * Limit: how many prefabs should spawn on the chunk.
  * Resolution: in how many chunks should the planet surface be divided.
  * Radius: how many prefab chunks should the generated around the observer.
  * Rotate: controls the rotation of the prefabs, allowing for full random (best suited for rocks and other props), surface normal aligned (can be used for smaller plants and bush like assets) and planet center, this is best suited for most buildings and trees.

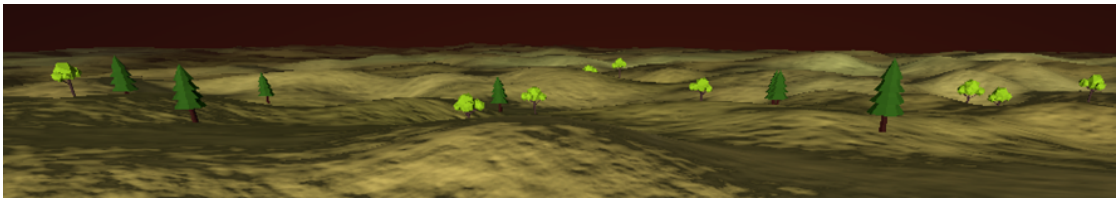


Figure 4.5: Planet Prefabs

    * Shared material: If a building is large enough to extend beyond the atmosphere (such as a skyhook), the atmosphere should not be present inside the building. Setting a reference to the atmosphere here prevents this from happening.
    * Prefabs Array: This array contains the prefabs to be spawned at random, allowing for an arbitrary number of prefabs to be set.
- SGT Terrain feature: like SGT Terrain heightmap but it only applies to a configurable place on the surface, ideal for creating landmark mountains or valleys manually.
- SGT Terrain simplex: adds simplex noise to the terrain.

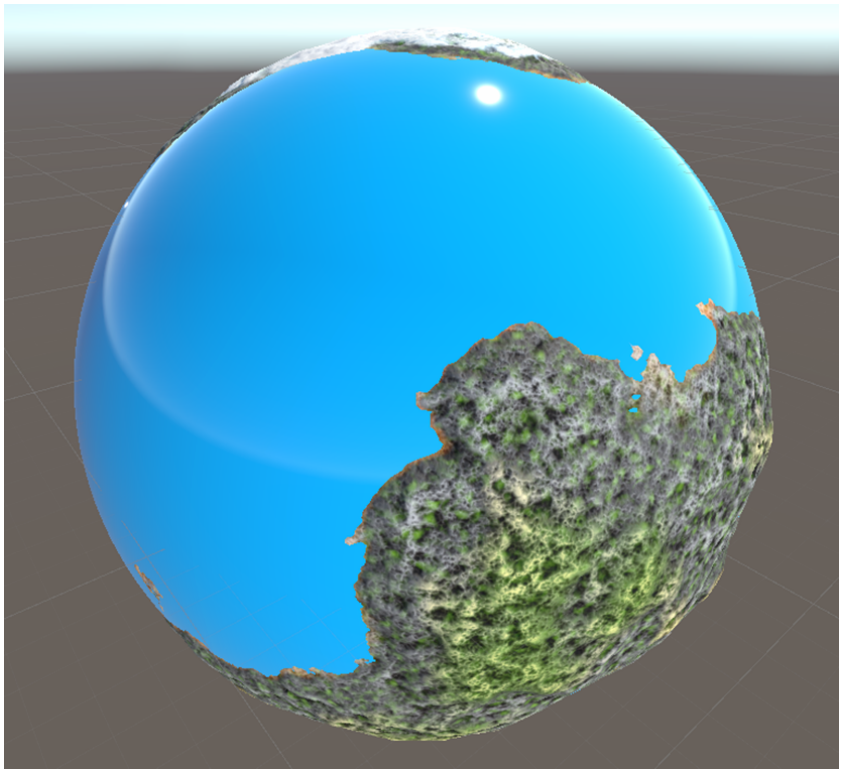


Figure 4.6: SGT Terrain Ocean

– SGT Terrain ocean: like SGT Terrain planet on most parameters, but controls the creation of oceans. It requires a ground reference.Figure 4.6
– SGT Terrain ocean Material: Sets the ocean material. This material can have configurations to allow the creation of shorelines and beaches. It can also include a normal map to animate waves on the ocean but it affects performance negatively.

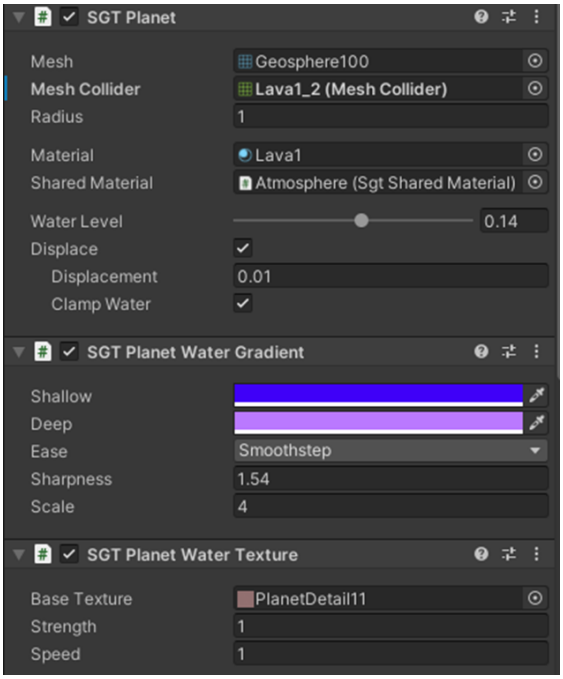


Figure 4.7: SGT Planet Scripts View

- SGT Planet: a more compact and quicker version of SGT Terrain planet. It uses a base mesh (geosphere) to then bake texture details into it. It also controls the water level. Figure 4.7
  - SGT Planet water texture and detail: these components customize the SGT Planet based water (not the SGT Terrain based water).

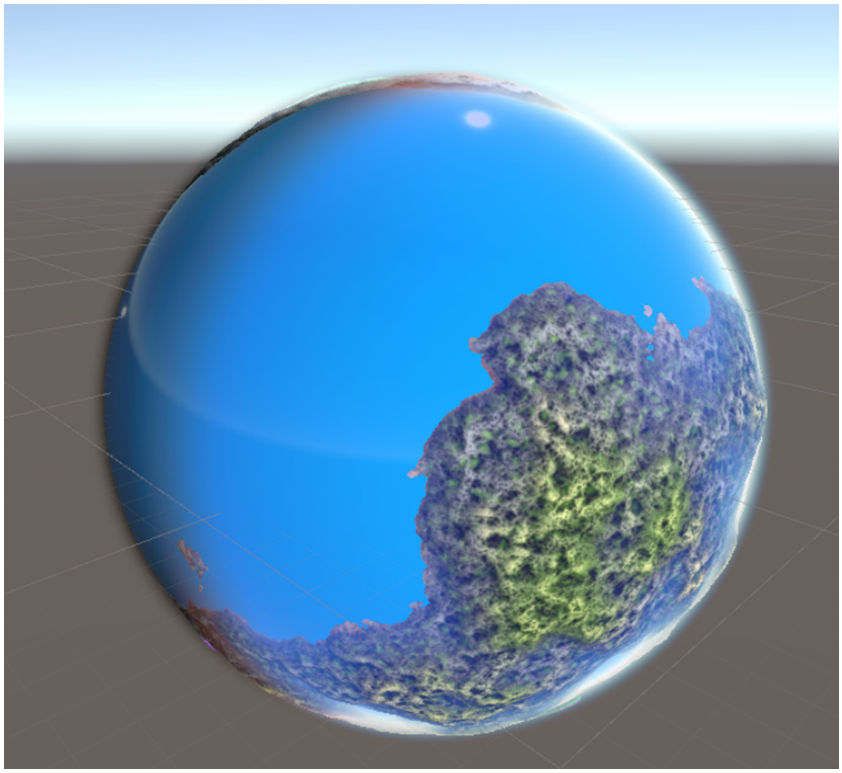


Figure 4.8: SGT Atmosphere

Atmosphere Creation

This section creates the atmospheric effect for planets, its day/night sky and atmospheric light scattering among other things that can be configurable from within the scripts.

- SGT Atmosphere: Creates the atmosphere effect around a planet surface, it requires a base mesh. Figure 4.8 Figure 4.9
  - Source Material: contains a special material/shader that visualizes the atmosphere.
  - Height: how far above the surface the atmosphere should extend.
  - Fog, sky, middle: these parameters control the properties of the sky visualization from within the atmosphere.
  - Night: enables a night sky and allows its configuration.
- SGT Atmosphere DepthTex, SGT Atmosphere LightingTex, SGT Atmosphere ScatteringTex: These scripts allow for the configuration of textures (including depth scattering and lighting) to procedurally generate them for the atmosphere and apply them to the SGT Atmosphere material.

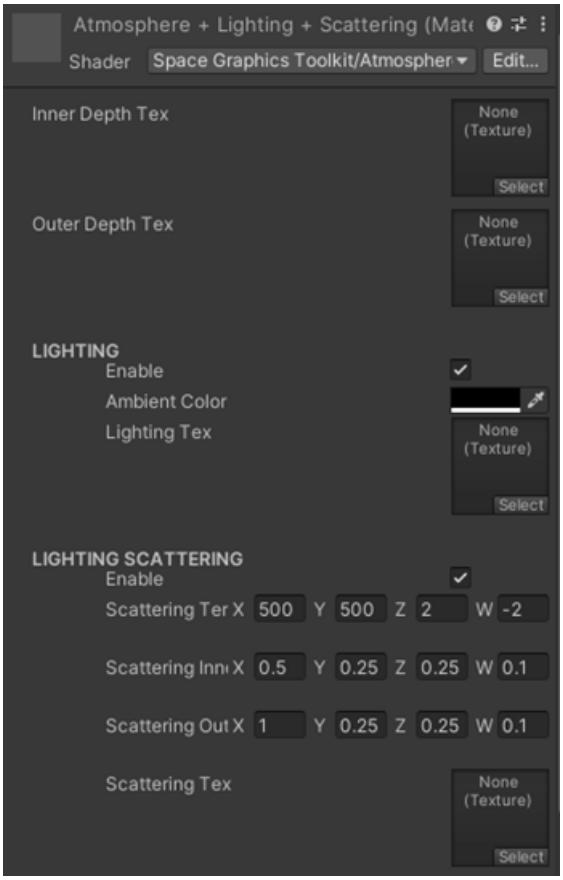


Figure 4.9: Atmosphere Configurable Shader Material

- SGT Aurora, SGT Aurora MainTex: allows the creation of aurora effects on the planets and fully configure them.

Clouds Creation

This section creates clouds on the planet atmosphere, it can also create storms with lighting.

- SGT Cloudsphere: Creates the clouds around the planet using a shader material. Figure 4.10
  - Source material: contains a special material/shader that visualizes the clouds.
  - Radius: the height of the clouds, it should be above the terrain radius but less than the atmosphere radius, while it won't cause problems to go above and below these limits it will look bad.
  - Color: cloud color.

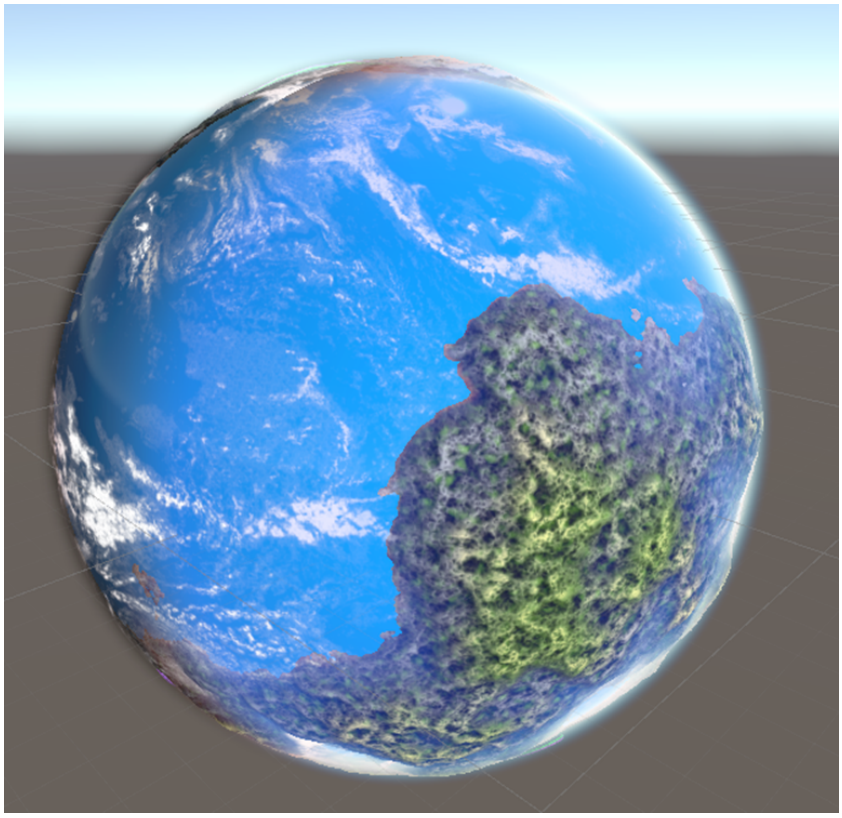


Figure 4.10: SGT Clouds

- SGT Cloudsphere DepthTex, SGT Cloudsphere LightingTex, SGT Cloudsphere NearTex: these scripts configure the procedurally generated SGT Cloudsphere shader.

- SGT Lighting Spawner: randomly spawns lighting on the atmosphere simulating a storm. It can be configured with the lighting life time, the color, the sprites, the frequency and the maximum amount of lighting at one given time.

Ring Creation

This section creates a ring around a specified object, usually a planet although it can be used to create any form of procedural ring.

- SGT ring: creates the ring and allows adjustments to the properties such as the inner and outer radiuses and the texture of it. Figure 4.11

- SGT ring MainTex, SGT ring Lighting: like with the other elements it controls the texture and lighting properties of the ring.

- SGT shadow ring: casts a fake real-time shadow on the ring simulating the planet shadow, this is way more efficient than calculating the real-time illumination using a planet as the occluder.

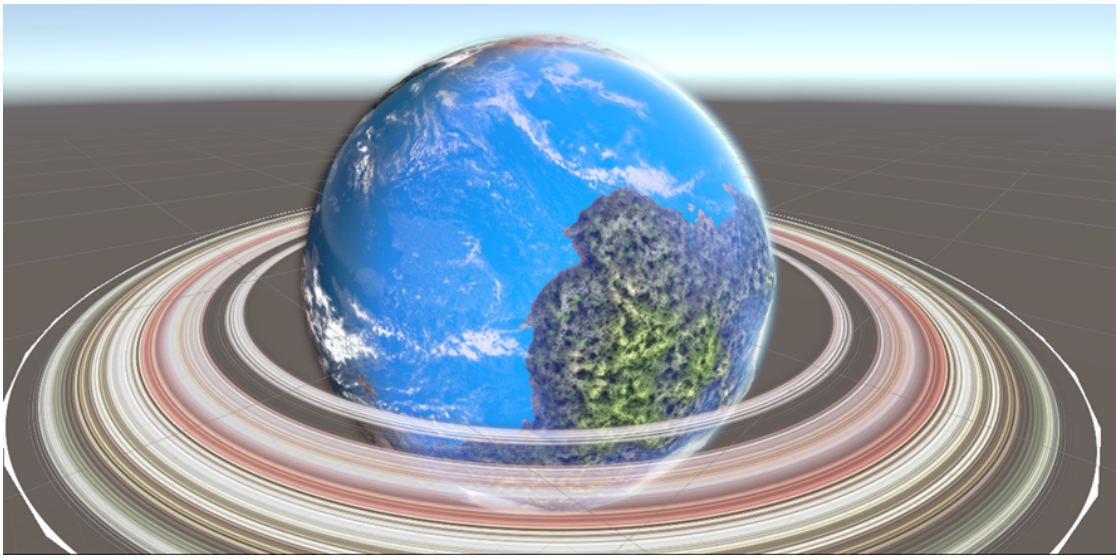


Figure 4.11: SGT Ring

4.3.2 Non-planetary Objects

- Anomalies: Space anomalies can be made with a variety of options from the SGT library. There can be nebulas, debris, solar flares and more. They add to the visual experience of the game.

- Singularity: A singularity is a point in space with a lens effect that bends light around it, it can be configured to emulate a black hole-like effect using very high lensing effects.

- Stars: the light sources of each star system scene, they are composed of 2 directional lights that rotate towards the player position, one of the lights is rotated 180 degrees and has lower intensity to improve lighting.

  - Star billboard: It gives the star a visual representation from far away. Simulating the stars using quads always faced towards the camera.

  - Star prominence: when fairly close the player can see the star surface, in a way it works like a planet surface but animated, random “flares” coming out of the surface can be randomly spawned for a more realistic representation of the star.

- Moons: they can be built like planets with smaller radius, there's also the option to have them rotate around their planet at a certain speed.
- Belts: asteroid belts can be implemented in 2 ways depending on their scope.
  - Planetary asteroid belts should be implemented using asteroid sprites rotated towards the camera, these sprites can cast shadows and can be rotating around the planet, they can also implement a simplified collider.
  - System wide asteroid belts can be implemented like planetary belts, but shadows and belt rotation should be handled with caution even if the optimized asteroids-based sprites can handle a massive amount having a system wide ring with enough density to look like a proper asteroid belt (like the ones on the solar system) can take hundreds of millions of asteroids.
  - It is also possible to combine 2 asteroid belts one with very simple sprites while the other has a few sparse gameObject large asteroids that can be used as asteroid outposts or just for visuals.

4.3.3 Artificial Objects

- Procedural cities: based on the *Mega sci-fi city* library [18] script, large procedural cities can be created using custom game objects. These cities can be placed on planet surfaces as large objects oriented towards the planet center via the SGT terrain prefab spawner. Custom areas can be defined using the SGT areas to focus the procedural city spawn along specific areas, such as coastlines.
- Reverse World/Dyson world: this is a special system, it's the capital of the empire and a key point of the main game narrative. It's a large artificial planet with a star inside it, the planet surface is reversed inwards. Figure 4.12

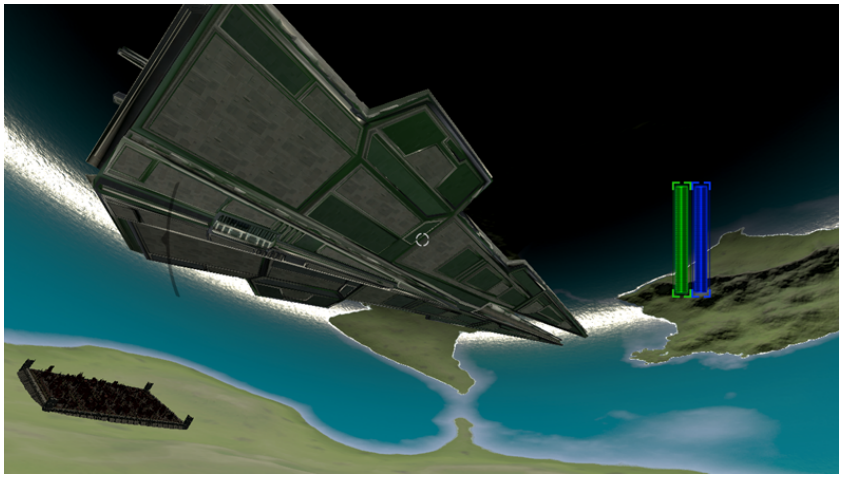


Figure 4.12: Reverse World and Procedural City

To create this planet the SGT library was modified using additional scripts that override formulas and added new options such as using negative radiuses, heightmaps working with negative radiuses and mesh collider creation with inward normals. This approach was preferred over directly modifying the library. Reversing gravity and influence also required improving the gravity scripts to allow for negative strength and if negative to reverse the gravity vector away from the center.

- Space stations and mission key locations: these locations have been manually crafted to better fit our exact requirements for the missions. They range from small space outposts, to large military bastions and planet surface outposts.

4.4 Player Character

To create the player character, the Ultimate Character Controller (UCC) [15] was used as a baseline. Building on this foundation, several functions provided by the UCC were extended, and new ones were created based on the UCC Ability system. Additional functionalities were also added to the character from scratch. The character creation process, modifications, and additions to the UCC, along with other non-UCC extra scripts, will be detailed in this section.

- **The Ultimate Character Controller (UCC):** this package handles the character movement (linear), animation, rigging, and kinematic movement. It also provides an interface for Health, Respawning and Abilities that will be modified to adapt the project needs.
- **The UCC Abilities:** The ability system provided by the package was initially confusing and hard to understand due to the vast number of options, many of which are not necessary for our purposes. However, using this ability system, abilities can be enabled or disabled via functions designed for that purpose. The UCC includes example abilities that serve as a guide for building custom variants.

  The abilities can vary a lot depending on their purpose but they generally contain a way to enable or disable them and a way to trigger them (be automatically, by an event, triggers, keypresses, pretty much anything). They also contain a state to set when the ability is in use and an optional state requirement to work.
- **The gravity alignment Ability:** This ability is based on the surface alignment ability and inherits from it. Instead of aligning to the surface normal, it aligns to the gravity field vector described in the previous section on game systems. To trigger this ability, the gravity field's trigger zone is used to change the character's state to a custom state called "align to gravity." This ability remains active as long as the "align to gravity" state is active. Figure 4.13

Figure 4.13: Gravity Alignment Ability

- **The attack ability** was a sample provided by the UCC that satisfied our needs.
- **The drive ability** is based off the interact ability. It allows the player to “drive” a vehicle, for that purpose the vehicle must implement an interface so the ability knows it’s a driveable vehicle. The difference with the interact is that when the interaction key is pressed it will put the player on “the driver seat” and transfer control to the vehicle (this prevents the player character from moving while driving the vehicle). Like most abilities this ability also sets a state on the player “driving”. This ability unlike the gravity alignment is always active as the player character never loses the option to drive. An interesting option of this approach is that it will be very easy to implement other abilities based on this one like “advanced driving” that allow the player to drive other vehicles.
- **The respawn ability:** UCC treats respawning as an ability. This was already implemented by UCC but it required some adjustments to some of the auxiliary scripts for it.

  The ability by default respawns the player at the start location or a random spawn point designated for "our team" (intended for competitive shooters). This was modified to allow the player to set specific spawn points through interaction with "medbays." This interaction sets the spawn point to the specific medbay. However, this had its own complications since medbays could be on starships or stations that could be destroyed. To prevent an error a stack of spawnpoints was saved for the player in case some got destroyed and the player killed. Should all the stack waypoints be invalid there will always be an indestructible spawnpoint at the bottom of the stack, the default spawnpoint.
- **The character chunk position:** this relatively short script tracks the player current chunk on the scene, this is important for the floating origin, as the world should be moved based on this value, this script sends the event that will end moving the world when the character exceeds the chunk boundaries.
- **Collision layers mask:** the character implements a custom collision mask so some collision layers are ignored (energy shields to stop projectiles but not players or ships).

- **Fall damage ability:** This ability deals damage on fall distance and was implemented already, this had some issues as the Y distance no longer meant much with custom gravity, the impact velocity was used instead. Using the velocity at first had some serious issues as moving too fast with the ships was killing the player, it was switched to relative velocity to its parent.
- **Character weapons:** the character can pick up different weapons and use them, it's also possible to pick up ammo for the weapons. The character can also have multiple weapons equipped and change them during gameplay. The damage is handled by the projectile itself, same for the hit decals on surfaces they impact.
- **The UCC states:** The ability system relies heavily on the character states, either by using them as conditions or setting states such as driving or gravity alignment.
- **The UCC attributes:** They contain the character key variables, such as health and shield, it also contains more advanced options to interact with these attributes.

4.4.1 Character UI

The character UI is the HUD the player will see when the player isn't driving any vehicle or in some menus. Figure 4.14

- It consists of an UI displaying the health and shield bars on the top left. (displaying the character attributes of health and shield).
- It displays the current weapon or weapons equipped (default fists if no weapon is currently equipped).
- It contains the crosshairs.
- It has a red vignette effect when the player gets hit.

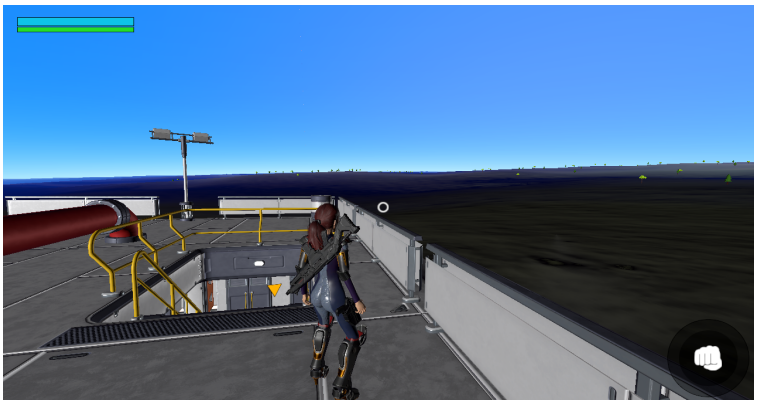


Figure 4.14: Character UI

4.5 Ships

As a game set in a sci-fi environment, the usage of spaceships will be common throughout the game. This section will provide detailed explanations of how each ship component works and the reasons behind their implementation.

- **IDriveable interface:** For the player to be able to pilot the ship, implementation of the IDriveable interface is necessary. This allows the character's "drive" ability to recognize the ship. The interface is implemented on the base ship controller.

- **Ship piloting:** Driving a ship with full control requires 6 degrees of freedom (up/down, forward/back, left/right, yawn, pitch, roll). The ship controller handles the user input and adjust the ship velocity accordingly.

  It's worth noting that the forward/back direction input is handled a bit differently in the form of thruster strength that can be changed with the mouse wheel but stays set instead of having to press the buttons constantly, this improves long travels in-game.

- **Warp:** Once the ship exceeds a certain threshold of speed it enters warp mode, here the increments of speed are larger and the weapons get disabled unless stated otherwise (some special ships might be able to fire on warp), a special effect is also used to represent a warp tunnel.

- **Ship gravity:** Given that ships can be pretty much in any rotation most ships contain artificial gravity given by the "directional gravity component" that the "align to gravity" ability of the character will use to stay on the ship ground. The ship's gravity influence tends to be stronger than planet gravity so that it won't rip the player away from the ship. This might not be the case for singularities.

- **Ship chunk:** Monitors the current chunk the ship's at for the floating origin system, it also handles not moving the ship on a chunk change if the player is on it.

- **Ship isPiloted/isWithin AABB:** Checks for the players presence on a ship, the is piloted variable is of key importance for multitude of scrips while the AABB checks if the player is within the ship bounding box (but not necessarily piloting it). This is useful when checking for the player presence if the ship gets destroyed with the player onboard or to align the player to the gravity of the ship.

- **Weapons:** Some ships will include weapons. They will be explained in detail on the next section.

- **Subsystem targeting:** It is possible to damage specific parts of ships such as engines or turrets.

- **Ship interior features:** Except small fighters, ships offer interiors, this not only helps with the immersion but also serves some purposes. Larger ships can have medbays where the player can heal and set spawn points within the ship, they can also have hangars for smaller ships to dock at, and other rooms like engine or shield rooms to allow for modular upgrades.

- **The hangar system:** A ship with hangars will allow the player to store smaller ships and move them all by moving the carrier ship. The player can use smaller ships to explore ahead without risking the larger, usually harder to get, ship.

- **The modular component system:** Some ships allow for component swap on their respective rooms, for example the player can change the shield generator on the shield room for a better one (as long as it's compatible, it's not possible to put a massive core on a small ship).

- **The ship camera:** Each ship has a camera configuration profile that will adjust the ship camera for the specific ship. This controls the camera distance based on the ship size.

- **Scene change:** The only way to change scene is from within a ship as scene change implies doing interstellar travel, on a scene change the player brings the ship piloted and any docked ship along to the new scene.

4.5.1 Ship UI

The ship UI displays the relevant ship information, it can be different on some ships either information wise or design wise to represent different ship manufacturers Figure 4.15, it consists of.

- Health and shield bars: they display the current ship's hull integrity and shield integrity.

- Thruster speed bar: displays the current thruster speed in relation to its maximum.

- Crosshairs: the manual aiming point for turrets.

- Power distribution (optional): how the ship's power is distributed (engines, shields, weapons).

- Subsystem health (optional): the status of systems such as engines.

- 2D representation of the ship(optional).

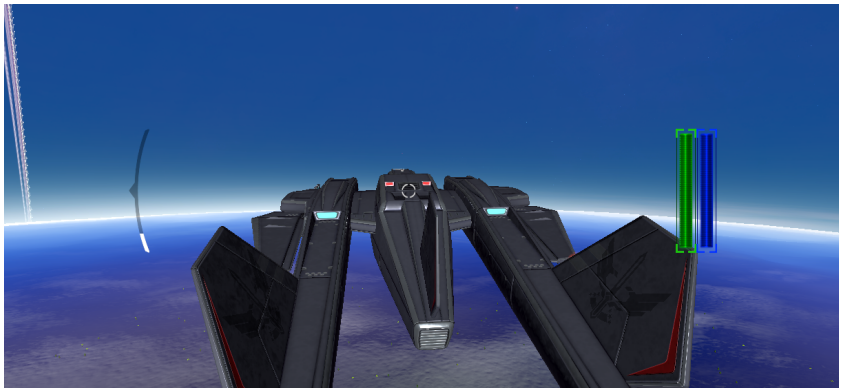


Figure 4.15: Ship UI

4.6 Weapon Systems

There are 2 main classifications for weapons, personal weapons and ship weapons.

- **Ship weapons:**
  - Based on mounting type:
    - Fixed mount weapons: Fixed weapons will shoot towards the direction they are facing, this tends to be common on smaller ships where the player will have to rotate the ship to aim the weapons. Figure 4.16

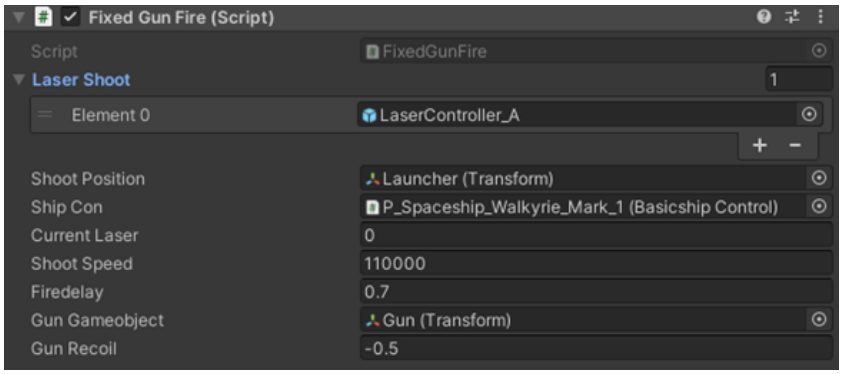


Figure 4.16: Fixed Weapon Script

* Turret mount weapons: Turrets allow for more flexibility as the player can aim them independently from the ship's direction. Turrets have a limited firing arc depending on the ship's geometry and turret location. Turrets can be set to autofire, on this mode they will automatically aim to the selected target and fire if it's within firing arc. Figure 4.17

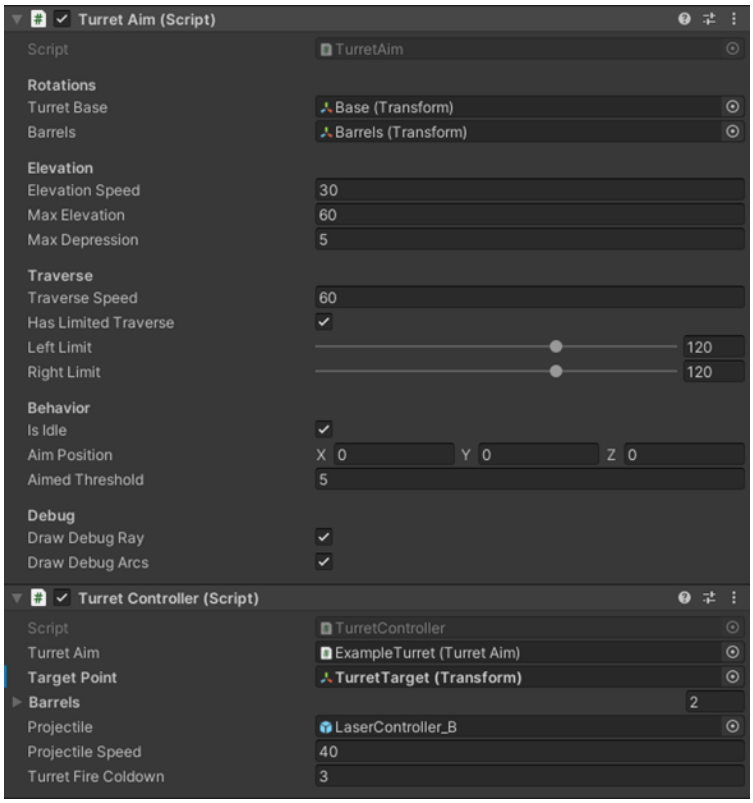


Figure 4.17: Turret Weapon Scripts

– Based on weapon type:
  * Laser cannons: The most common type, they can be mounted on turrets or fixed slots, they deal normal damage and travel very fast. They use energy.
  * Ion cannons: Like lasers but they disable shield regeneration. They use energy.
  * Missiles: They track the objectives if one was selected when fired, missiles travel slower and can be destroyed as they are physical objects. They deal a moderate amount of damage and use ammunition.
  * Proton bombs: This system can only be mounted on fixed weapons, they are very slow and have a slow rate of fire, proton bombs deal massive damage and use ammunition of their type.

- **Ship weapons projectile scrips**

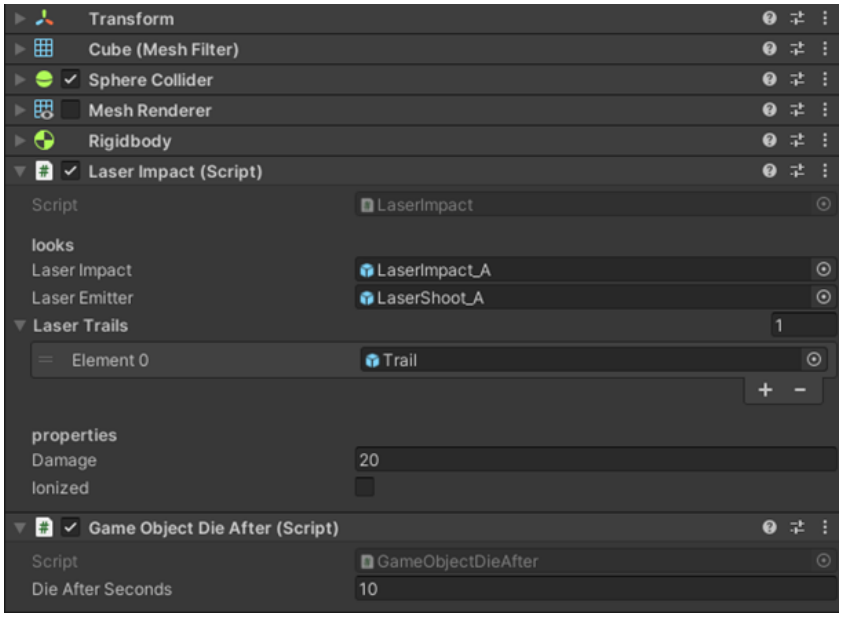


Figure 4.18: Type A Laser bolt

The laser Impact script controls the laser damage and its visuals like the trail and impact effect. Missiles beams and bombs have a similar script attached. The die after script prevents lasers that do not hit anything from staying indefinitely in game.

- **Character weapons:** There are 2 kinds of character weapons, one-handed and two-handed. The player can equip 2 one-handed weapons on each hand
  - A weapons asset pack that already had the weapon models and animations for reload as well as the recoil system and was provided by the UCC Demo was used.
  - The projectiles work in a very similar way to the ship projectiles just on a smaller scale of visuals. Personal weapons cannot effectively damage ships (except grenade launchers or other explosive weapons).

4.7 AI

The game implements a simple AI system mainly focused on player combat.

4.7.1 Ground AI

For ground combat the A* pathfinding with a custom navmesh provided by the *A* pathfinding project* was used, the benefit of this navmesh was its adaptability to other surfaces such as spheres or non-vertical aligned space stations as opposed to *Unity* built in navmesh that was too limited for the project. The ground AI implementation on the project will need further work and refinment for a full release or a demo as only the basic fondations have been implemented (navmesh and turrets).

Very basic agents that would attack the player on proximity were implemented.

Non agent based enemies like turrets will shoot on sight towards the player.

4.7.2 Space AI

A simple behaviour for the AI ships was implemented. They will engage opposing team ships by distance and some will try to flee when too damaged. If their target ships flee too far they will revert back to a patrolling stance or idle stance. The movement of the ships will depend on their size, while a fighter will try to approach and shoot a larger ship will stay at range firing with its turrets.

4.7.3 Fleet AI

Capital ships contain a more advanced AI, they can coordinate smaller ships to focus fire on specific targets or recall all allied ships to defend them.

4.7.4 Flagship Influence and AI

Flagship class ships have influence spheres that decay based on the distance to the ship. AI ships affected by the influence zones might alter their behaviour based on the influence score. For example, if a ship wants to retreat because it's damaged but the influence of the zone it's at is vastly on its favor it will stay fighting, the opposite can happen if a ship is undamaged, but the zone influence is too negative, it may retreat towards one of his allied flagships. The idea behind the influence zones is to keep battlegroups around their respective flagships. If the flagship is destroyed ships will fall into disarray and ignore influence zones unless another friendly flagship is close.

4.7.5 Collision avoidance

Ships will monitor their forward arc by emitting 5 ray casts with limited distance forming a forward cone, if one of the rays hit a specific set of layers the ship will change its state to evading for a short period and then get back to whatever it was doing. During evading

state the ship will alter its trajectory as many times as necessary until all 5 raycasts are clear. This is done to avoid ships crashing into each other or into asteroids/stations.

4.7.6 Turret AI

Turrets have their own AI based on their configuration:

- Primary turrets: They will fire at the target the ship AI they are part of wants.
- Secondary turrets: They will fire at a target of their choice, they are usually smaller and have shorter ranges. An example of this would be a flagship focusing another flagship with its main large turrets while the secondary turrets target fighters that are very close in a form of anti-fighter defense.
- Point defense: These turrets will only target missiles trying to intercept them.

4.8 Dialogue System

The game uses a dialogue system with options for interactions with NPCs, the dialogue system uses the *Pixel Crushers Dialogue System* library. This library allows for the creation of dialogue trees with actors. It also allows the use of LUA scripting for more advanced features.

- **The dialogue database**: This is the main element of the system, it stores all the dialogue trees and actor references.
  - Actors: Can be added to dialogues, an actor is a conversation agent that will have assigned dialogue tree nodes. At least 2 actors are required, but more can participate, the player actor that will have access to the UI selections during dialogue and at least one NPC actor that will converse with the player.
  - Conversations: These are the dialogue trees; each conversation has its start point and end point and can be composed by as many nodes as we want. Figure 4.19

    Connectivity between the nodes will establish the dialogue flow.
  - Nodes: The nodes contain the dialogue text itself, but they can also call in scripts and events to, for example, run their respective audio track so that voice can be added to the dialogue.
  - Variables: Variables from within the dialogue database can limit some answers behind a quest or other events like another conversation.
- **Dialogue system events:** This script can be added to actors so when certain conversation events happen, such as starting or ending a conversation, certain scripts can be invoked. This is especially useful to toggle the UI and set dialogue cameras if there's any.

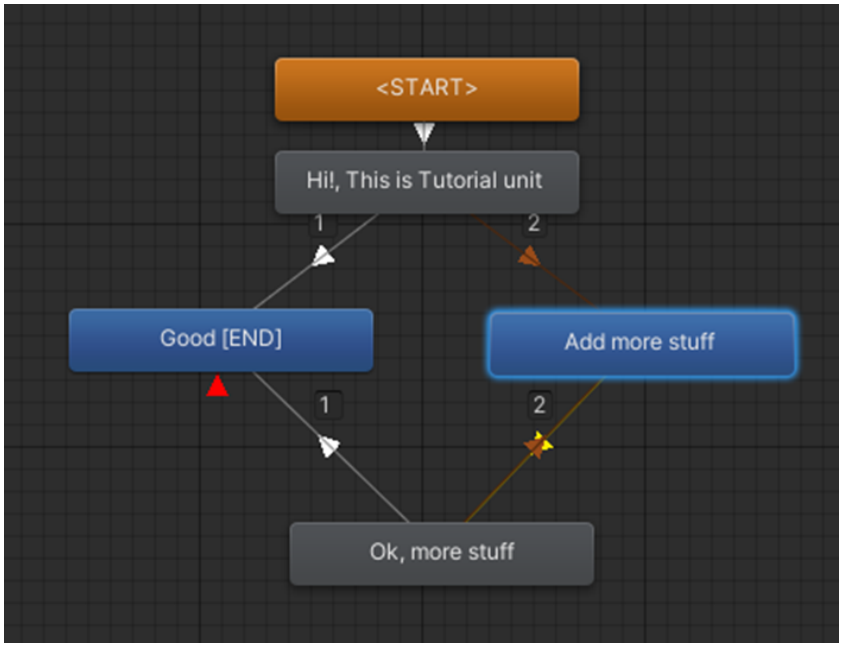


Figure 4.19: Dialogue Conversation Tree

- **Dialogue Actor:** this script contains the actor information and sets a relation between the database actor and the *Unity* game object.
- **Selector and usable:** These scripts allow the player to be able to select and start conversations with capable entities (the ones with the usable script). The usable script can be configured to only be usable if certain conditions are met. These scrips also configure the default start conversation keybind and the maximum distance to interact with the dialogue conversator.

4.8.1 Dialogue UI

The main dialogue UI consists of a simple UI with 2 main areas, the top side is where the NPCs answers while at the bottom side the player gets a dialogue wheel with multiple choices that can be clicked on. Figure 4.21. Interactable NPCs will have a small UI box with their name and “right click to speak” when the player looks at them. Figure 4.20

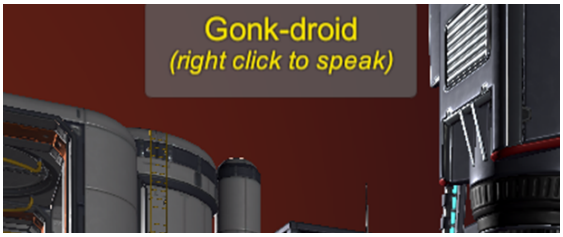


Figure 4.20: Dialogue Interactable Pop-up

Figure 4.21: Main Dialogue UI

4.8.2 Dialogue Camera

During conversation start, the active camera is switched to the dialogue camera. This camera allows for focusing on NPCs as they speak or providing a static overview of the conversation.

4.8.3 Dialogue Generative AI Voice

To create the voices of our NPCs, generative AI was used based on the RVC (Retrieval-based Voice Conversion) system Figure 4.22 [19] [14].This Python-based system enables the training of voice models, which can then be used to either modify existing voices or

generate new ones from text-based sources. Given the scope of the project, training models from scratch was deemed impractical, Therefore, pretrained models were downloaded for use with the NPCs.

To address the issue of uneven results when purely creating voices from text sources, a hybrid approach was used. Initially, an in-house voice was used to imbue the AI with the desired inflection and emotion for the dialogue. This recorded voice was then combined with text using another voice model to achieve a more satisfactory outcome.

It was observed that while this system is still inferior to employing professional voice actors, the results obtained with the AI can be suitable for non-critical conversations in various games. By reserving professional voice acting for main story conversations, a strike between quality and cost is achieved.

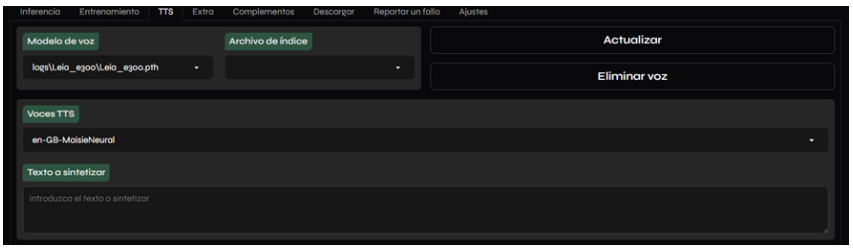


Figure 4.22: RVC text-to-Speech Web UI

4.9 Inventory System

The game implements a simple inventory, the player can pickup items like weapons, ammo, health packs or quest items. These items are added automatically to the player inventory. The inventory also has 2 “equipped” items, one for each hand where weapons can be equipped. Most weapons require both hands to be used but some single hand weapons like smaller pistols can be dual-wielded. The UI displays the current equipped weapons and their ammunition.

Other items that can be included in our inventory are ship components; however, due to their size, only one such component can be carried at a time.

To cycle between inventory weapons, the mouse scroll wheel can be utilized. Additionally, pressing "t" can unequip our weapon to prevent misfires.

The player can obtain weapons or ship components trough the game world, as quest rewards or by purchasing them on some stations using “credits”, the game currency. Figure 4.23

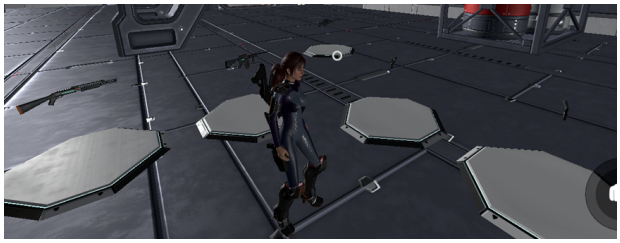


Figure 4.23: Weapon/Item Platforms for Shops or Pickups

4.10 Waypoint System

Navigation in an open-world space-based game is crucial. To achieve this, a waypoint system is utilized to indicate the position and distance of relevant locations.

- **The waypoint indicator:** This is the main class, it handles the displaying of the waypoint, it can display the waypoint in multiple options (sprites, name, prefabs) and it can keep track of items out of camera view. All waypoints subscribe themselves to the waypoint manager. Figure 4.24

Figure 4.24: Waypoint Indicator Configuration

- **The waypoint manager:** The waypoint manager is a singleton class that handles some general parameters of all waypoint indicators, it can toggle them on and off and change the cameras they are rendered at. Figure 4.25

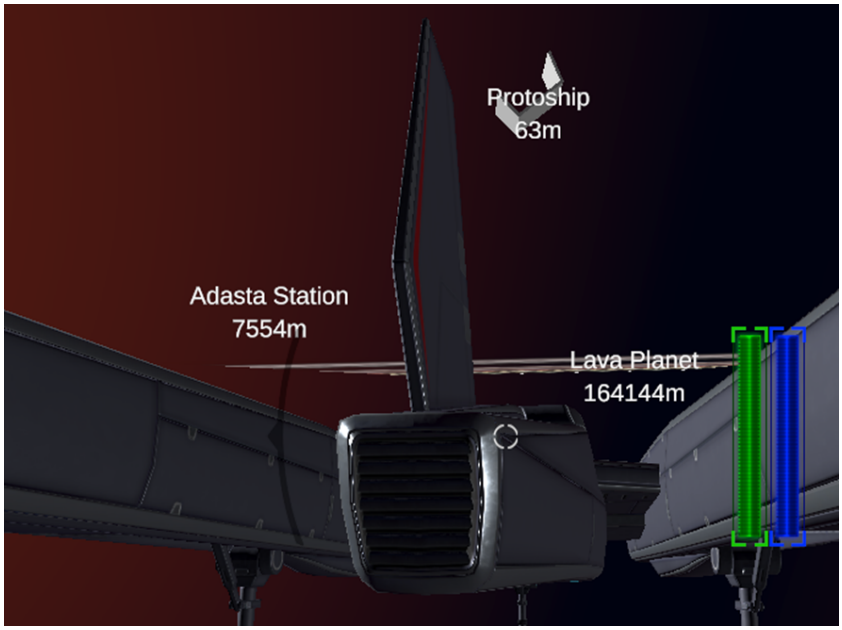


Figure 4.25: Waypoints Visible From Ship View

- **The waypoint canvas:** All waypoints render on a canvas that point to their position, this canvas contains them.
- **Hierarchical waypoints:** Waypoint indicators allow for the setting of maximum and minimum distances for waypoint visibility, which is especially useful to avoid visual clutter. Using this system, a waypoint can indicate a planet's location, and only display minor waypoints to small outposts on the planet's surface when in close proximity. This prevents navigation clutter that would occur if waypoints were rendered with unlimited distance.
- **Ship waypoints:** Finding planets and locations is as important as locating the ship after disembarking. The ship waypoint ensures awareness of the ship's location, using a different style than navigation waypoints and disabling once the ship is piloted.
- **Enemy/friendly markers:** The waypoint system is also used in combat to keep track of the enemy(red) and friendly(green) ships. Combat waypoints are tracked continuously, even when off-screen, to provide the position at all times, including when they are behind the player. Figure 4.26 Figure 4.27
- **Quest objective:** This waypoint will mark the current quest objective with a golden hexagon if it is within the star system. Its visible range will be unlimited.

Figure 4.26: Off-Screen Hostile Waypoint

Figure 4.27: Hostile Waypoint

4.11 Galaxy Menu

The galaxy menu displays the current state of the galaxy and allows travel to available star systems.

- **Control zones:** The galaxy is divided into multiple zones (sectors) containing a small number of systems, by default all zones except one are controlled by the empire (red), the contested zone will be marked in yellow, completing quests will change zones to "liberated" (blue) and unlock access to previously locked zones by "contesting" them.

- **System markers (travel click):** Star systems will be indicated on the map, selecting them will make the player travel to that system if it's unlocked.

- **Awareness:** Capturing sectors will raise awareness, this will increase the number of enemies the player faces on the following sectors and its represented by a slider at the top of the Menu UI.

- **Popular Opinion:** Decisions will alter this value, it is represented next to the Awareness.

- **Navigation and movement on menu:** The galaxy menu represents a view of the galaxy, the player can move the map and zoom in-out to more accurately select systems that are close.

4.12 Sound Design

The sound design of the project aims to increase immersion trough the game world. For this purpose, a wide variety of ambient music and sound effects were used:

4.12.1 Music

This section will explain the music choices.

- **Ambient calm music:** This will be the default music, it will be based on the star system the player is at and its main purpose is to provide a calm and accompanying feeling to the player without being too distracting. It's the background music.
- **Planet specific music:** Some rare planets will have their own ambient music, this music will try to evoke a feeling for the planet, be it mysterious, calm, etc.
- **Battle music:** If the player engages in combat music will change to a more action filled music.

4.12.2 Sound Effects

Sound effects are really important for the game feel on this project.

- **Ambient effects:** One of the main sound effects enhancing immersion includes the sounds of the ship, such as buzzing engines and beeping computers.
- **Bullets, missiles, lasers:** Each weapon has a different sound according to their type.
- **Reload sounds:** Some weapons have a reload sound.
- **Hit sounds:** Getting hit by lasers or bullets will produce different impact sounds, while getting hit by explosive weapons such as missiles will produce explosions.
- **Walk Surface:** Walking over different surfaces will produce different sounds (ground, metal floor..).
- **Warp Field:** Going at warp speed will have its characteristic humming sound. Entering and exiting from warp mode will also produce a sound effect.

4.12.3 Voice Effects

These effects are voice based and can come in 2 forms.

- **Dialogues:** Dialogues will use the AI generated voice explained previously on this document.
- ***"barks"***: These will be short voiced lines said in the background to add immersion and indirectly inform the player of things, for example an allied ship going down will yell *"critical damage sustained"* or something similar.

4.13 Additional Visual Effects

Previously, visual effects related to world creation, such as atmospheres and lens distortion for black holes, were explained. This section will cover non-world related visuals.

4.13.1 Weapon Related Effects

- **Projectile Lasers:** In a sci-fi game, laser visuals are among the most important elements. Visible, projectile-like laser shots were chosen, featuring a glowing head and a trail behind them. These laser projectiles can emit light onto nearby objects. A laser controller script manages additional visual effects, such as trails and impact visuals. Figure 4.28

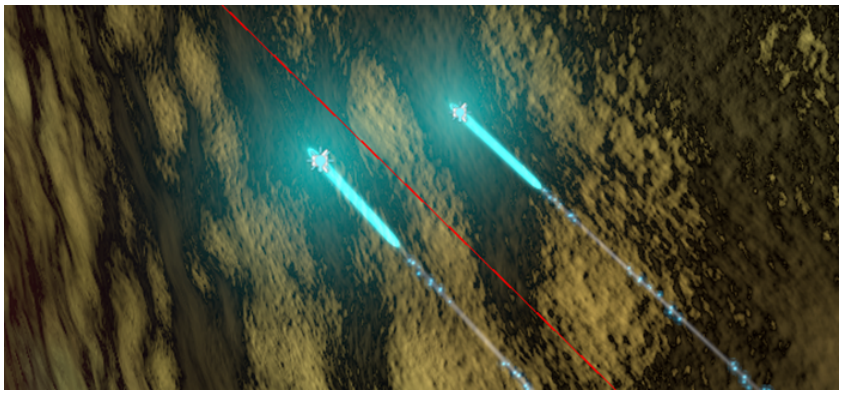


Figure 4.28: Laser-Plasma Projectiles

- **Beam Lasers:** More akin to real lasers, they travel almost instantly and produce a continuous beam from the firing point to the final destination for a fixed duration in time.
- **Proton bombs:** They are represented as slow moving big glowing orbs.
- **Explosions:** Missiles and bombs produce explosion effects; ship destruction also produce explosions of bigger scale depending on the exploding ship.

- **Plasma explosions:** Proton bombs produce very large plasma explosions to represent their devastating nature.
- **Hit decals:** Thanks to the demo weapons on UCC it was relatively simple to get their hit decals working. some weapons leave decals on the impacted surface. Figure 4.29

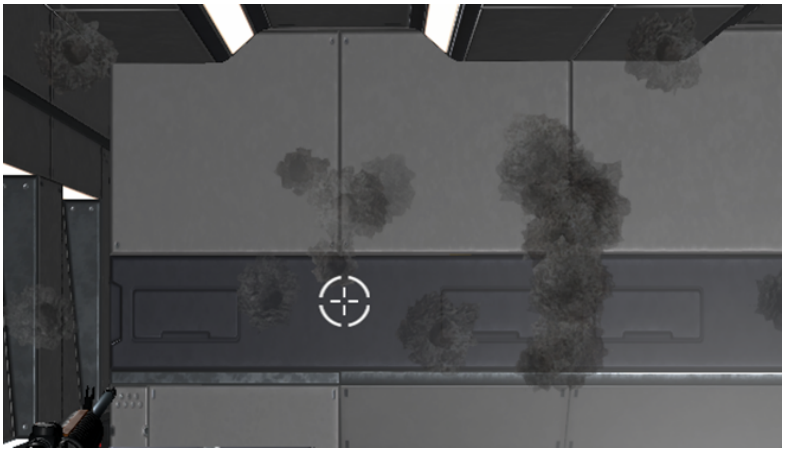


Figure 4.29: Hit Decals

4.13.2 Non-Weapon Effects

Non-weapon effects: There are more effects in the game such as computer panels blinking or water tanks provided by external assets that add to the visual quality of the game.

4.13.3 Warp Tunnel

The warp tunnel effect encases the ship while at high speeds. Figure 4.30

Figure 4.30: Warp Tunnel

4.14 Level Requirements

A level is a star system on this project. Levels share some common characteristics between them such as:

- **Egress point:** The entry point or level load-in position.
- **Space partition chunks:** Required for the floating origin.
- **Skybox:** Adds visual quality.

These components are present on all star systems. Other components will depend on the type of level the player enters.

- **Standard star system:** The most common type of system. They contain planets moons asteroids and stations along one star that acts as the main environmental light source.
- **Sector capital star system:** They contain the inter-sector hypergate additionally to what a standard system can contain. It's also more likely to find military stations, bastion moons and urban planets on this type of level.
- **Deep space anomaly:** They contain black holes, neutron stars and other exotic space anomalies, they can contain some space stations.

- **Dyson Reverse World:** Contains the reverse world, the galaxy centre hypergate and a star in the centre as a light source.

4.14.1 Tutorial Level

This section will give additional detail to the design of the whole tutorial level. The objective of this tutorial quest is to showcase as many of the game systems as possible while providing an engaging experience. The tutorial is longer than the demo as it also acts as the story introduction.

- **The start:** At first the player will appear on an imperial city planet with some NPCs to dialogue with. This will showcase the dialogue system and planet gravity.
- **The first quest:** It will consist of a basic fetch quest on the planet surface. This will showcase the simplest kind of quest.
- **The first ship:** On the first quest the player will be given a very simple ship and a quest to the local capital city. This will showcase the basics of spaceship piloting and waypoint navigation.
- **The Alpha Oupost:** On the Alpha Outpost the player will complete the second quest and be asked to solve a problem on another planet using an improved ship. This will showcase hangar systems and capital ships.
- **The first warp travel:** With the improved ship the player will travel to another planet within the system to deliver the message. This will showcase warp travel and different planets.
- **The first battle:** This will feature the first hostile ship and the first combat. With the information returned to the quest giver the player will be asked to travel to the sector capital system back again.

  This is where the demo scope ends.
- **The emergency:** Upon returning the player will be immediately asked to report to another star system. This will feature the first interstellar travel.
- **The Uprising:** As soon as the player enters the new star system a large battle will start featuring large fleet combat.
- **The orbital defensive station:** The settlement defense station has been disabled and the player will be required to join ground combat on the station to capture it. This will feature ground combat.
- **The betrayal:** After capturing the station and returning to the ship. The player ship and character will be attacked and killed by all friendly ships to cover up the secret of a machine uprising. This gives introduction to the main story start from a medical vault without memories of previous events for the player character.

4.15 Conclusion

This technical specification outlines the essential components and systems that define the gameplay experience. From procedural generation of planets and worlds to navigation and waypoint systems ensuring seamless exploration to the dynamic visual and sound effects that enhance immersion, each element is meticulously designed to create an engaging and cohesive world. Combat mechanics, galaxy traversal, and distinctive laser visuals contribute to the overall depth and realism. This comprehensive approach ensures that players are provided with an immersive and technically robust gaming experience.

Chapter 5

Results

Our primary goal was to lay the groundwork for a space-sim RPG game, and we've made significant strides in developing and testing the game systems. We successfully implemented procedural generation for planets, spawning buildings, vegetation, and various space phenomena such as stars, quasars, asteroid belts, and rings, resulting in visually appealing environments. Additionally, we achieved seamless transitions between different gravity orientations and planets, a crucial feature for a space simulation game.

Our efforts also led to the creation of a ship control and flight system, along with a turret system that includes rotation and elevation limits. We developed a dialogue system with multiple choices and added voice capabilities generated using generative AI to enhance immersion. While time constraints limited the depth of our inventory system, we ensured it covered the basics. To overcome Unity's limited navmesh capabilities, we implemented a custom navmesh using the A* Pathfinding project.

Furthermore, we established a foundational AI for spaceships, providing a basis for further refinement. A notable accomplishment was the creation of a procedurally generated Dyson Sphere world featuring custom cities and outposts.

Ultimately, our primary achievement was delivering the essential elements to immerse players in an expansive open-space RPG experience, fulfilling our overarching objective.

In addition to these accomplishments:

- We developed multiple core scripts (e.g., ship control, event manager, turrets, ship health, hangar system, AI, projectiles, gravity, additive scene control and switching) ranging from 200 to 600 lines of code each.
- We created over 40 support scripts.
- We conducted extensive adjustments and configurations of planets using the SGT library to create the game world.

- We fine-tuned and configured cities using the Mega Sci-Fi City library.
- We edited and adapted numerous scripts provided by external assets.
- We created multiple dialogue trees and configured numerous NPCs for dialogues.
- We implemented multiple cameras with different functionalities and interfaces for various situations.
- We set up multiple waypoints for different types of objectives.
- We created a tutorial demo.
- We tested various models and configurations of generative AI voice systems and experimented with various AI programs.
- We implemented a floating origin system to address limitations.

These combined efforts have provided a robust foundation for a captivating space-sim RPG, ensuring players a rich and immersive experience in an expansive universe. Figure 5.1

Figure 5.1: Sector Capital Orbit

5.0.1 Issues occurred during development

- Underestimated character controller complexity: the initial planification account for a character controller, however it was severely underestimated how complex would it be to get it to properly work with all the features that we wanted.
- Underestimated procedural generation of the Dyson World, it wasn't "just invert the normal" by far. It took more time and extra work on the collider mesh to work.
- Unity float32 limits of 100k units. It required to redesign the whole level scheme until a solution was found, to set a floating chunk origin. This system adds complexity but at least we can use coordinates further than 100k units.
- Gravity conflicts and game bugs, they slowed down development but this kind of problem was planned for in advance.
- Development of Ground AI was rushed due to time constraints due to the extra time spent on the navmesh solution research.

CHAPTER 6

Conclusions and Future Work

Contents

In this chapter, the conclusions of the work, as well as its future extensions are shown.

6.1 Conclusions

I was able to set the bases of a larger project and create the core systems. It is by far the largest and most complex project I have made during my degree. It required not only knowledge in programming and unity but in game design, narrative design, conceptual design, 3D modelling and materials, sound design. All of them taught during the degree to a certain extent, using that as basis it was feasible to improve upon and expand. I mainly leant about unity features during this project, but also about some of the limitations and how to overcome some of them. I also learned about optimization techniques, improved event usage, shader and shader graph basics. Studying the procedural library and modifying it also taught me some of the techniques to generate procedural content such as Perlin and simplex noises. In conclusion creating this project tested what I've learnt during the degree and expanded upon it. It also become clear than to fully develop this game into a final product with all its features the time investment would be far larger than the scope of this Final Degree Project.

6.2 Future work

The game I've accomplished is largely what I envisioned when I first conceived it, with only minor adjustments along the way. However, during development, I found myself brainstorming additional ideas. While these concepts were promising, they proved too ambitious to implement within our timeframe. If I were to continue this project, I'd explore improving even further the procedural generation to include materials and a crafting system. I would also improve the inventory system further and if possible add a way for players to build their own outpost gathering materials from barren planets. While the game already offers multitude of mechanics, introducing these options could enhance replay value and provide players with fresh motivation.

Additionally, I'd consider expanding the number of NPCs available for interaction. I value the depth that conversations with in-game characters bring to the experience, as their diverse perspectives and experiences breathe life into the game world. Increasing the pool of NPCs to converse with could enrich the player's immersion and deepen their engagement with the game.

Due to the open world nature of the game, adding more side quest, extra ships, planets, story or entire side story arcs or game finales would be feasible with enough time and resources.

Improving optimization and visual fidelity. While I'm satisfied with the current presentation there's always room for improvement, especially if we plan on expanding the project further.

Bibliography

[1] Average hourly wage. https://es.talent.com/salary?job=programador. 2024.

[2] Gantt chart maker. https://www.onlinegantt.com/. 2024.

[3] Gogs self hosted git server. https://gogs.io/. 2024.

[4] Adobe. Adobe creative suite. https://www.adobe.com/es/creativecloud.html. 2024.

[5] Atlassian. Sourcetree. https://www.sourcetreeapp.com/. 2024.

[6] Creepy Cat. Sci-fi kit vol 1. https://assetstore.unity.com/packages/3d/environments/sci-fi/3d-scifi-kit-vol-1-39202. 2024.

[7] Creepy Cat. Sci-fi kit vol 2. https://assetstore.unity.com/packages/3d/environments/sci-fi/3d-scifi-kit-vol-2-56794. 2024.

[8] Creepy Cat. Sci-fi kit vol 3. https://assetstore.unity.com/packages/3d/environments/sci-fi/3d-scifi-kit-vol-3-121447. 2024.

[9] Inc Docker. Docker. https://www.docker.com/. 2024.

[10] Krita Foundation. Krita. https://krita.org/es/download/. 2024.

[11] Aron Granberg. A* pathfinding project. https://arongranberg.com/astar/features. 2024.

[12] Microsoft. Visual studio. https://visualstudio.microsoft.com/es/downloads/. 2024.

[13] MSGDI. Character asset. https://assetstore.unity.com/packages/3d/characters/humanoids/sci-fi/free-test-character-asuna-205897. 2024.

[14] OpenAI. Chat gpt. https://chat.openai.com/. 2024.

[15] Opsive. Ultimate character controller. https://opsive.com/support/documentation/ultimate-character-controller/. 2024.

[16] Overleaf. Latex editor. https://www.overleaf.com/. 2024.

[17] Pixelcrushers. Dialogue system. https://www.pixelcrushers.com/dialogue-system/. 2024.

[18] Daniel Kole Productions. Procedural city. https://assetstore.unity.com/packages/3d/environments/sci-fi/mega-sci-fi-city-pack-18898. 2024.

[19] RVC Project. Tts ai. https://github.com/RVC-Project/Retrieval-based-Voice-Conversion-WebUI. 2024.

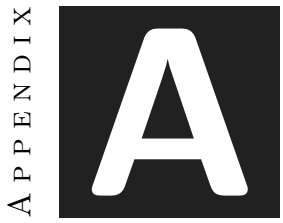
[20] Ton Roosendaal. Blender. https://www.blender.org/download/. 2024.

[21] Unity Technologies. Unity asset store. https://assetstore.unity.com/. 2024.

[22] Unity Technologies. Unity engine. https://unity.com/es/products/unity-engine. 2024.

[23] Unity Technologies. Unity engine documentation. https://docs.unity3d.com/. 2024.

[24] Carlos Wilkes. Space graphics toolkit. https://carloswilkes.com/Documentation/SpaceGraphicsToolkit. 2024.



Other considerations

This appendix is included to comment some aspects not considered in the rest of the project and indirectly related.

A.1 Git Server

Due to the limitations of GitHub with 100+ MB files I had to setup a personal Git server using a *gogs* docker container. This allowed me to work from multiple computers and keep a version control always available.

A.1.1 Docker Server [9]

For being able to utilize my own git server i had to dedicate a computer to a 24/7 server with internet connectivity.

A.1.2 Dynamic IP and Domain names

Due to the nature of residential ISP IP addresses it was impossible to guarantee a permanent non-changing ipv4 address. To circumvent this issue I used a dynamic DNS service by cloudflare and connected my registered domain to it. This ensures that the git server will stay accessible from the internet given proper credentials.