



UNIVERSITY OF PIRAEUS

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGIES
DEPARTMENT OF INFORMATICS

Thesis

Thesis Title:	(English) Making of a 3D platformer action-adventure game titled: "Path of the Stars"
Τίτλος Διατριβής:	(Greek) Δημιουργία ενός τρισδιάστατου παιχνιδιού πλατφόρμας δράσης-περιπέτειας με τίτλο: "Ο Δρόμος των Άστρων"
Student's name-surname:	Pantazis Dimosthenis
Father's name	EVANGELOS
Student's ID No:	Π22232
Supervisor	Efthimios Alepis, Professor

Delivery Date September 2026 / Σεπτέμβριος 2026

Copyright ©

The copying, storage, and distribution of this work, in whole or in part, for commercial purposes is prohibited. Reprinting, storage, and distribution for non-commercial, educational, or research purposes is permitted, provided that the source is cited, and this message is retained. The views and conclusions contained in this document express solely the author's perspective and do not represent the official positions of the University of Piraeus. As the author of this work, I declare that this work is not a product of plagiarism and does not contain material from unreferenced sources.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν αποκλειστικά τον συγγραφέα και δεν αντιπροσωπεύουν τις επίσημες θέσεις του Πανεπιστημίου Πειραιώς.

Ως συγγραφέας της παρούσας εργασίας δηλώνω πως η παρούσα εργασία δεν αποτελεί προϊόν λογοκλοπής και δεν περιέχει υλικό από μη αναφερόμενες πηγές.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Efthimios Alepis and his assistant, Margaritis Kamitsios for their guidance, feedback and support throughout the development of this thesis. Their extensive knowledge and experience were vital in the completion of the project.

I would also like to thank my family and friends, both old and new, for their support and company throughout these past four years.

Abstract

This Thesis Project is a 3D Video Game built using the Unity 3D Engine, a free tool used for the creation and publication of all kinds of video games. The genre I chose to create is a 3D platformer action-adventure game with elements inspired by popular titles such as Celeste, Super Mario Galaxy and Angels Essence. In this game, the player ventures among the stars, alongside their creator, The Witch of The Stars, towards new frontiers in search of a destination that's described in manuscripts of old. To reach it, the player must maneuver their way around obstacles, floating asteroids and bottomless pits with the help of their specialized boots that allows them to invert their own gravity. They must also find the three Astral Keys to open a number of doors sealed by those who wish to hide the location mentioned in the manuscripts from the world. During the journey, the player can find items to help them as well as talk to their creator for guidance on their next step.

Keywords: Unity, 3D, Platformer, Space, Gravity, Video Game, Action-Adventure

Περίληψη

Αυτή η πτυχιακή εργασία είναι ένα τρισδιάστατο βιντεοπαιχνίδι που κατασκευάστηκε με τη χρήση της μηχανής Unity 3D Engine, ένα δωρεάν εργαλείο που χρησιμοποιείται για τη δημιουργία και δημοσίευση όλων των ειδών βιντεοπαιχνιδιών. Το είδος παιχνιδιού που αποφάσισα να δημιουργήσω είναι ένα 3D platformer action-adventure βιντεοπαιχνίδι εμπνευσμένο από δημοφιλείς τίτλους όπως Celeste, Super Mario Galaxy και Angels Essence. Σε αυτό το βιντεοπαιχνίδι, ο παίκτης ταξιδεύει ανάμεσα στα άστρα, μαζί με τον δημιουργό του, τη Μάγισσα των Άστρων, σε νέους ορίζοντες στην έρευνα τους για να ανακαλύψουν την τοποθεσία ενός τόπου που περιγράφεται σε αρχαίες περγαμηνές. Για να το φτάσουν, ο παίκτης πρέπει να περάσει από εμπόδια, αστεροειδείς και απύθμενες τρύπες με τη βοήθεια ειδικών παπουτσιών που επιτρέπουν στον παίκτη να αλλάζει τη δικιά του βαρύτητα. Πρέπει επίσης να βρει τα τρία Astral Keys για να ανοίξει ένα αριθμό πυλών που σφραγίστηκαν από εκείνους που επιθυμούν να κρύψουν από τον υπόλοιπο κόσμο την τοποθεσία που αναφέρεται στις περγαμηνές. Κατά τη διάρκεια της περιπέτειας, ο παίκτης μπορεί να βρει μια πληθώρα αντικειμένων για να τον βοηθήσουν, καθώς και να συζητήσει με τον δημιουργό του για καθοδήγηση στη διάρκεια του βιντεοπαιχνιδιού.

Λέξεις-κλειδιά: Unity, 3D, Platformer, Space, Gravity, Video Game, Action-Adventure

Table of Contents

	Acknowledgements.....	II
	Περίληψη.....	III
	Abstract.....	III
1	Introduction	1
2	Background Information.....	2
2.1	How to run the Video Game.....	2
2.2	Similar Video Games and Inspirations	3
3	Game Design	4
3.1	Story	4
3.2	Title screen.....	5
3.3	Main game	5
3.3.1	Section 1.....	6
3.3.2	Section 2.....	8
4	Game Assets.....	12
4.1	Items.....	12
4.1.1	Stars	12
4.1.2	Health Recovery.....	13
4.1.3	Speed-Up.....	13
4.2	Portals.....	14
4.3	Asteroids.....	14
4.4	Moving Platform	15
4.5	Astral Key.....	16
4.6	Gate.....	17
4.7	Light Pillar	17
4.8	NPC and Dialogue.....	18
4.9	Player Character	20
5	User Interface (UI) and Camera.....	22
5.1	User Interface.....	22
5.1.1	Pause Menu.....	23
5.2	Camera	24
5.3	Game Over Sequence.....	25
6	Scripts.....	26
6.1	Move1.....	26
6.2	PlayerGravity.....	28
6.3	Manage.....	29
6.4	Checkpoint.....	31
6.5	Respawn	32
6.6	ObjectMovement	34
6.7	Gate.....	36
6.8	Asteroid.....	38
6.9	Dialogue scripts.....	39
6.9.1	Dialogue.....	39
6.9.2	StartDialogue	40
6.9.3	DialogueManager.....	41
6.10	FollowPlayer.....	44
6.11	Health_gain.....	45
6.12	Speed_up.....	46
7	Conclusions and Future Development	47
8	Resources and References	48

Table of Figures

Figure 2.1: File structure of the Project	2
Figure 2.2: Example of how alternate gravity is explored in Super Mario Galaxy.	3
Figure 3.1: Initial Title Screen.	5
Figure 3.2: Section 1_1.	6
Figure 3.3: Section 1_2.	6
Figure 3.4: The inverted gravity path of Section 1_3.	7
Figure 3.5: The standard gravity path of Section 1_3.	7
Figure 3.6: End of section 1_3 and introduction of the first Astral Key and gate.	8
Figure 3.7: From left to right: Sections 2_2, 2_1 and small segment of 2_3.	9
Figure 3.8: Start of Section 2 once both left and right paths have been explored.	9
Figure 3.9: Section 2_1.	10
Figure 3.10: Section 2_2.	10
Figure 3.11: Section 2_3_1.	11
Figure 3.12: Section 2_3_2.	11
Figure 3.13: Section 2_3_3.	11
Figure 4.1: Visual representation of collectible stars.	12
Figure 4.2: Visual representation of the Health Recovery Object.	13
Figure 4.3: Visual representation of the Speed-Up Object.	13
Figure 4.4: Visual representation of a Portal Object.	14
Figure 4.5: Visual representation of an asteroid Object.	15
Figure 4.6: UML diagram displaying the interaction between an asteroid and the Player.	15
Figure 4.7: Visual representation of two moving platforms. The one on the left moves up and down, whilst the right one moves from left to right.	16
Figure 4.8: Visual representation of an Astral Key.	16
Figure 4.9: Visual representation of a Gate.	17
Figure 4.10: Visual representation of a Light Pillar. Regardless of where within the light pillar the player interacts with. Their gravity will be flipped.	18
Figure 4.11: Visual representation of NPC and dialogue prompt.	18
Figure 4.12: Example line of dialogue.	19
Figure 4.13: UML diagram displaying the interaction between an NPC and the Player.	19
Figure 4.14: Small sample of NPC lines.	20
Figure 4.15: Visual representation of the player character and tutorial prompt.	21
Figure 5.1: Visual representation of the Projects User Interface.	22
Figure 5.2: Visual representation of the Projects Pause Menu.	23
Figure 5.3: Confirmation Menu when the Player wishes to exit to the Main Title screen.	24
Figure 5.4: Visual representation of the Game Over screen.	25
Figure 6.1: move1 script.	27
Figure 6.2: Player character animator.	27
Figure 6.3: PlayerGravity script.	28
Figure 6.4: First part of Manage script.	29
Figure 6.5: Second part of Manage script.	30
Figure 6.6: Checkpoint script.	31
Figure 6.7: First part of Respawn.	33
Figure 6.8: Second part of Respawn.	34
Figure 6.9: ObjectMovement script.	35
Figure 6.10: Gate script.	36
Figure 6.11: Prompt for the player to interact with the Gate.	37
Figure 6.12: Displayed message if the Player is missing Astral Keys needed.	37
Figure 6.13: Asteroid script.	38
Figure 6.14: Instance of Player getting damaged by an asteroid.	39
Figure 6.15: Dialogue script.	40
Figure 6.16: StartDialogue script.	41

Figure 6.17: Part one of DialogueManager.42

Figure 6.18: Part two of DialogueManager.43

Figure 6.19: FollowPlayer script.44

Figure 6.20: Health_gain script.45

Figure 6.21: Speed-up script.46

1 Introduction

This Thesis Project is a 3D Platformer Action-Adventure video game built with Unity6.0 Engine. Most of the assets used in this Project are either simple-shaped objects I created with the Engines creating tools or, as of the time of writing this, free to download assets from the official Unity Asset Store. All of the scripts used in the Project have been created by me. My goal for this video game is to utilize the concept of gravity and explore the possibilities of how a Game Developer could experiment with such a fundamental element and create an enjoyable experience. I sincerely hope that this Project inspires other future Game Developers to branch out and experiment with concepts or techniques that would be considered unorthodox.

In the upcoming chapters I'll cover in depth the design of the Video Game and the Game Assets that are within. I will also analyze a number of scripts that are considered the backbone of this Project and how Objects may interact with each other to add layers of interaction. In further detail, the chapters contain the following:

- Chapter 2: Information the reader is recommended to have and background information regarding the Video Games that inspired the making of the Project.
- Chapter 3: The design philosophy used in terms of level layout and basic storytelling context of the events leading up to the beginning of the Video Game.
- Chapter 4: Explanation of the Projects game assets and how each interacts with the Player.
- Chapter 5: Discusses the Projects User Interface, its cameras functions and various visual elements that may appear on the Players screen.
- Chapter 6: A detailed analysis of important code scripts made for the Project.
- Chapter 7: Concludes explaining if the goals of this Thesis were met and how it could improve in the future.
- Chapter 8: Various Resources and References used in the making of this paper.

2 Background Information

This Thesis aims to be as comprehensible as possible to readers who have little experience or knowledge in Game Development that are knowledgeable in Computer Science. The Project was created using Unity Engine version 6000.3.9f1 and the Universal 3D Render Pipeline.

2.1 How to run the Video Game

Inside the unzipped folder you can find a number of subfolders and files that are crucial for the Project to operate. To launch the Video Game, you have to open the “Thesis.exe”. It is the executable file of this project and by opening it you can experience the Video Game. Additionally, it is recommended to play it at 1920 x 1080 screen resolution for the best experience.

	D3D12	9/18/2026 11:42 PM	File folder	
	MonoBleedingEdge	9/18/2026 11:42 PM	File folder	
	Thesis_BurstDebugInformation_DoNotShip	9/18/2026 11:43 PM	File folder	
	Thesis_Data	9/18/2026 11:43 PM	File folder	
	Thesis.exe	9/18/2026 11:42 PM	Application	652 KB
	UnityCrashHandler64.exe	9/18/2026 11:42 PM	Application	1,572 KB
	UnityPlayer.dll	9/18/2026 11:42 PM	Application extens...	34,958 KB

Figure 2.1: File structure of the Project.

2.2 Similar Video Games and Inspirations

One of the main video game titles that inspired this Project, Super Mario Galaxy, is a 3D Platformer released in 2007 whose central mechanic is gravitational flow. In that game, the Player moves around a number of celestial bodies, each with their own gravitational field in a series of levels with the goal of each level to collect a “Star”.

This Project attempts to capture the sense of gravitational fields that Super Mario Galaxy established, giving the Player a sense of freedom by expanding their limitations on a 3-dimensional space and allowing them to tackle problems in a way other 3D Platformers do not.



Figure 2.2: Example of how alternate gravity is explored in Super Mario Galaxy.

3 Game Design

The game is broken into two scenes. One scene acts as the title screen and the other contains the level in which the player has full control of the character.

3.1 Story

Long ago, there once existed a race praised for its intellect and expertise in alchemy. Their civilization thrived under their continuous technological advancements and were known throughout all the cosmos, expanding their sphere of influence across the stars. However, at some point in time, this civilization vanished alongside most records of their existence. No one learnt what truly happened to them or what catastrophe had befallen them. Some speculate these ancients delved into forbidden knowledge and destroyed themselves in the process or faced annihilation from an experiment that had gone wrong. Whatever the cause was, the damage had already been done.

Over the course of thousands of years, the world would come to forget the existence of this once great civilization, often believing it was fiction or legend from the past. Countless stories have been told through generations of a long-lost nation holding treasures beyond imagination lie hidden amongst the great expanse of the universe waiting for a worthy adventurer to arrive and claim it. Whether that treasure was in the form of gold or knowledge was left up for interpretation, but many individuals grew up with these myths and even more had attempted to locate it. Yet in the end, no one was able to find the location of that nation. Some however, were close and recorded their progress on various expedition records and manuscripts.

After an untold amount of time, a young witch managed to get hold of on some of the manuscripts written by past adventurers. She has no name, but she is known as "The Witch of The Stars". She's a curious soul, with an interest in alchemy but most importantly, the art of creation. During her life, she has attempted numerous times to create life out of various items and forms of matter. Most of her attempts either resulted in failure, or the subject in question failed to meet desired expectations. That is until recently, she managed to make small breakthrough and create "Boxboy"! Boxboy is a sculpture made of cardboard and other miscellaneous items given life by the young witch. With her new creation by her side, she sets her sights on finding the ancient civilization in hopes of a breakthrough in her research. Before departing, she made modifications to Boxboy's boots, allowing him to change his gravity at will. It would be a vital piece of equipment for the upcoming journey through space.

Traveling to the region pointed by the manuscripts was no easy task as the section of the universe she would explore would be considered highly deadly by many, as three suns reside closely near each other, raising concern for an imminent supernova should they collide and as a result, few dared to approach. However, it was necessary for the young witch to take the risk, as a supernova would destroy everything in this part of space, including potential clues for the civilization's true location.

Unbeknownst to the party of our two adventurers, the region of space they were in wasn't void of activity. Someone in the past had set up security measures to keep strangers away from ever reaching the truth, perhaps out of fear of what was to come, or to test if the future inheritors would be worthy, celestial seals had been place all around stopping anyone in their tracks from progressing. The only way to open the sealed gates was by using "Astral Keys". The Astral Keys possess interesting properties. They are not shaped like any ordinary keys, and neither were completely physical in nature, they had a form akin to a sphere, with a visible core on the inside.

And so, we arrive at the present. The young witch and her creation, Boxboy must traverse across the ancient ruins, dodging floating asteroids and overcoming the trials that lay before them. They begin their quest to find the Astral Keys and uncovering the secrets of the once great civilization.

3.2 Title screen

When the player launches the game, they will first encounter the initial Title Screen. They are presented with two options:

1. Start, which loads the second scene and starts the game.
2. Exits, which closes the application.



Figure 3.1: Initial Title Screen.

3.3 Main game

This scene is broken into two primary sections, with each having smaller subsections. The first section acts as the tutorial for the video game, teaching the player the basics such as the input controls or how certain mechanics work. The latter aims to challenge the player based on the knowledge and experience they had with the first section.

3.3.1 Section 1

Section 1 is the tutorial section of the game. In this section, the player learns the basic controls and mechanics of the game. It is divided into three subsections, each attempting to teach the player a different part of the game.

- 1_1: Aims to teach the player the basic movement controls and checkpoint system.

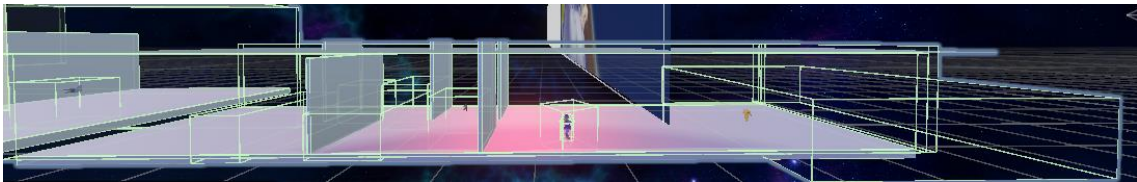


Figure 3.2: Section 1_1.

- 1_2: Introduces the gravity flip mechanic in two distinct forms. Firstly, it is shown as a pillar of light that when the player enters it, their gravitational rotation will be flipped by 180 degrees, allowing them to overcome obstacles that are impossible to pass while in specific orientation. Afterwards, the game teaches the player that they can change their gravity on demand but with the downside that once initiated, it has a short cooldown.

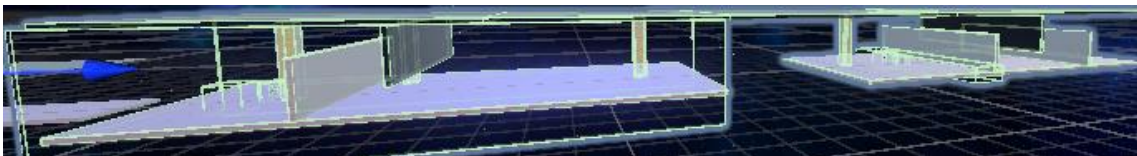


Figure 3.3: Section 1_2.

- 1_3: This is the final subsection of Section 1. It introduces the various items and how they function, as well as floating asteroids that act as a dangerous obstacle. In this section, the player is shown two different paths that they can take, which ultimately end on the same path. They can choose between the gravity inverted path or revert their gravity to normal and go along the second, slightly harder path, which contains more rewards. At the end of 1_3 the player will come across their first Astral Key, alongside a gate which they can interact with and open.

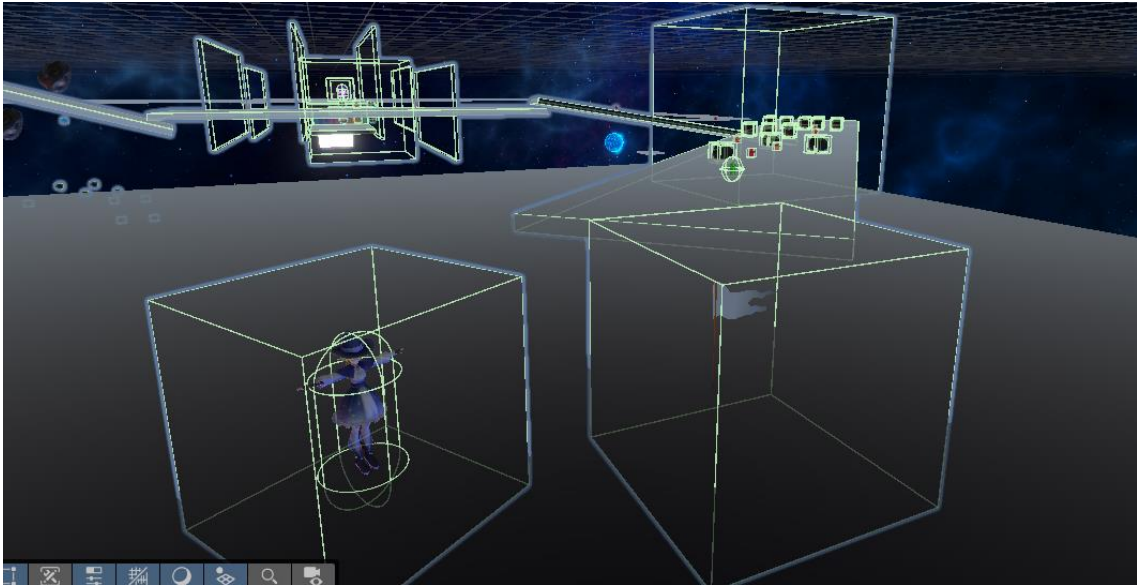


Figure 3.4: The inverted gravity path of Section 1_3.

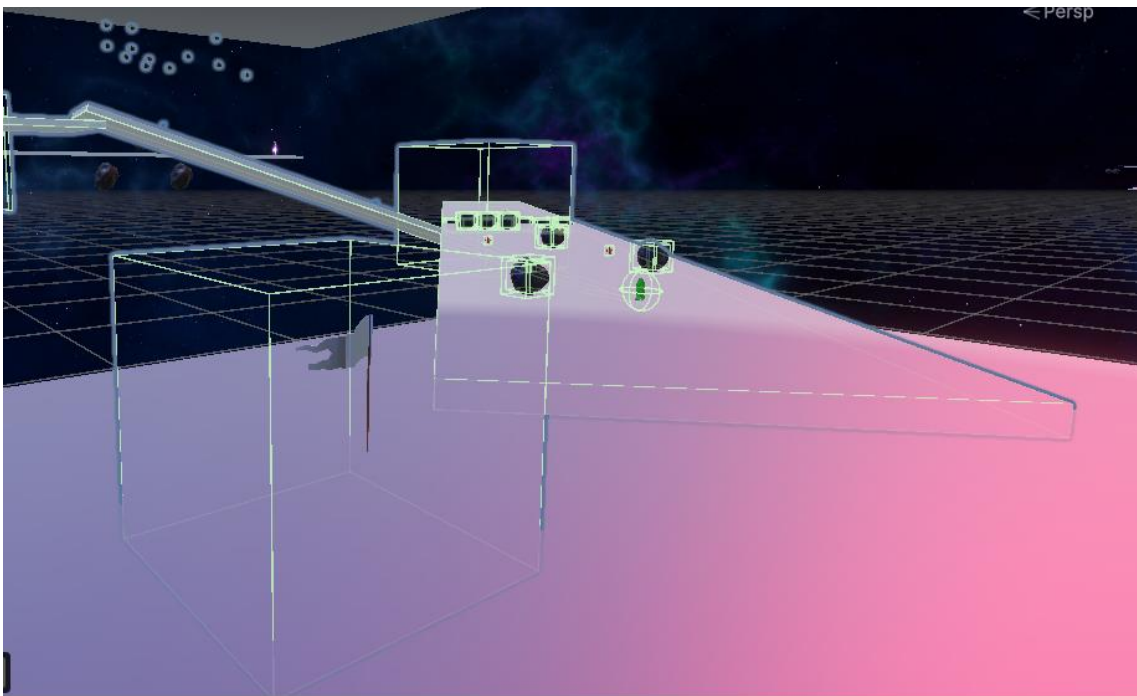


Figure 3.5: The standard gravity path of Section 1_3.

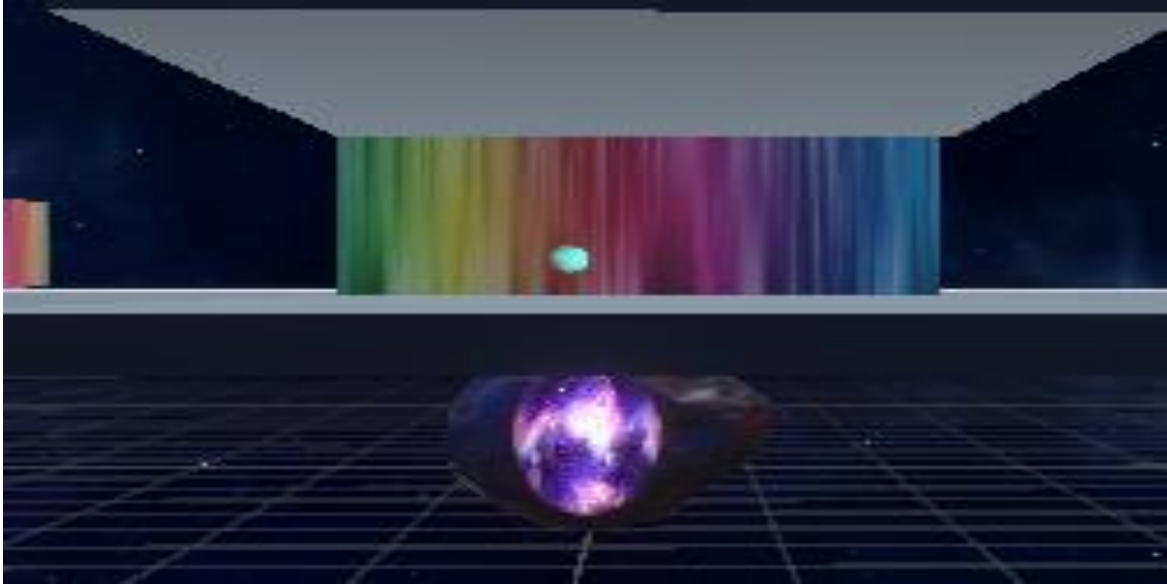


Figure 3.6: End of section 1_3 and introduction of the first Astral Key and gate.

3.3.2 Section 2

Section 2 marks the halfway point of the game and delves deeper into certain game mechanics. It contains three smaller subsections which one of them being the end of the game. After the player opens the first gate on subsection 1_3, they are presented with three diverging paths. The left and right paths contain a number of challenges that the player must overcome and at the end of each route, there is an Astral Key which is required to open the final gate on 2_3 and reach the end of the game. Additionally, there exist various portals that will teleport the player at beginning of section 2 while making the area they just explored inaccessible to prevent the player from accidentally backtracking to an area they've already been to.

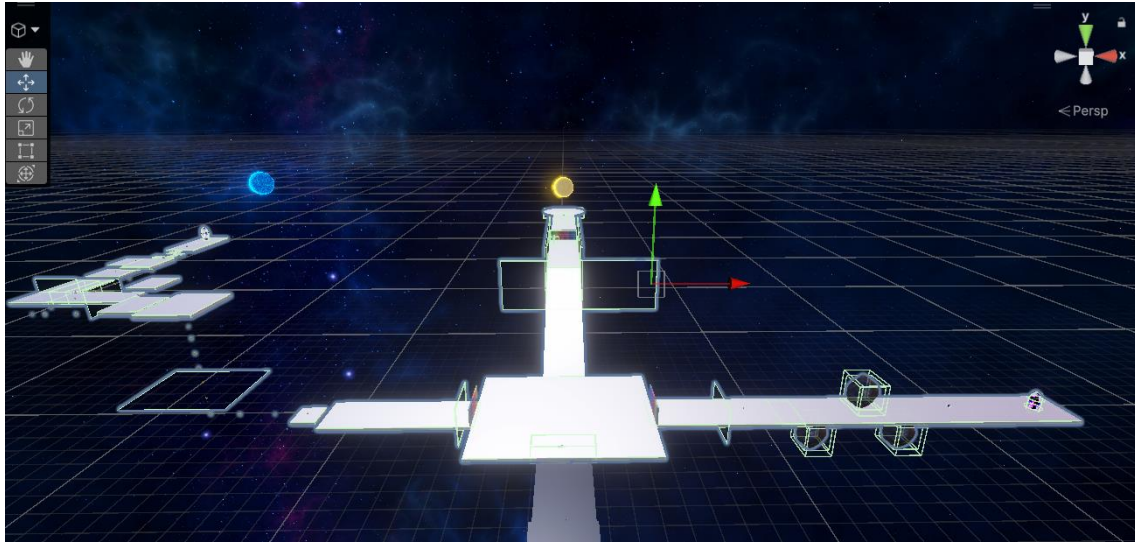


Figure 3.7: From left to right: Sections 2_2, 2_1 and small segment of 2_3.

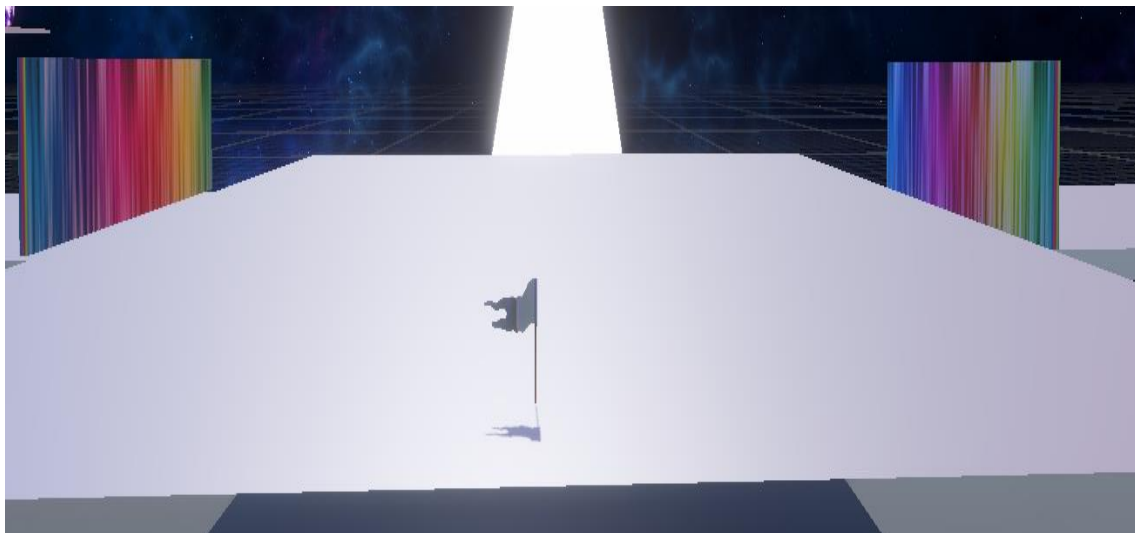


Figure 3.8: Start of Section 2 once both left and right paths have been explored.

- 2_1: subsection 2_1 consists of the central path of Section 2. It contains the final gate that can only be opened if the player has collected all three Astral Keys. Afterwards, the path leads to a small area and informs the player that they have reached the end of the game.

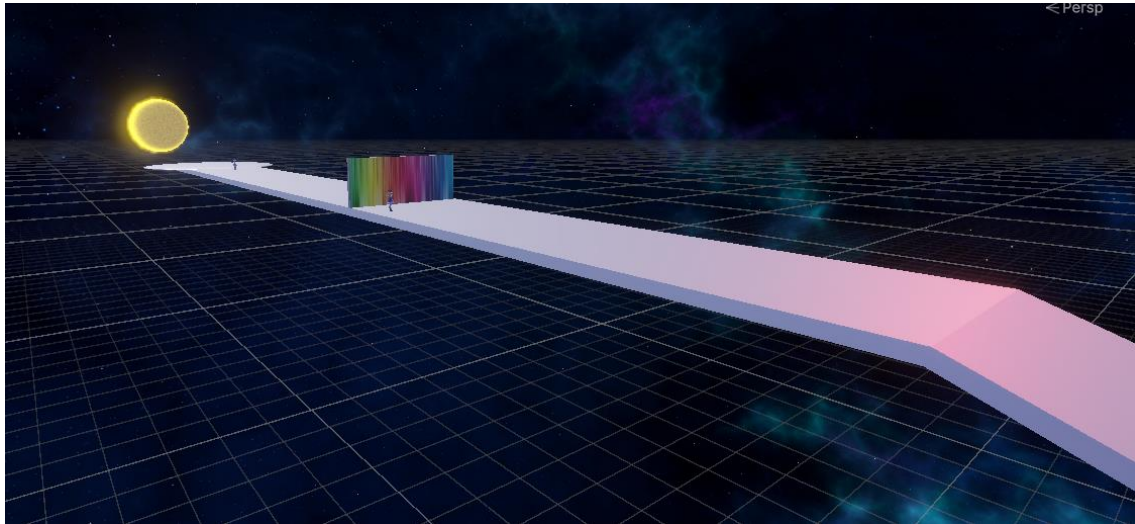


Figure 3.9: Section 2_1.

- 2_2: This section focuses on platforming and traversing with moving platforms. The player must move from platform to platform, occasionally having to change their gravitational rotation to reach certain platforms and reach the other side. At the end of the subsection, the player will be rewarded with a series of items, including an Astral key. In order to return back, the player can utilize a portal that will teleport them to the beginning of Section 2.

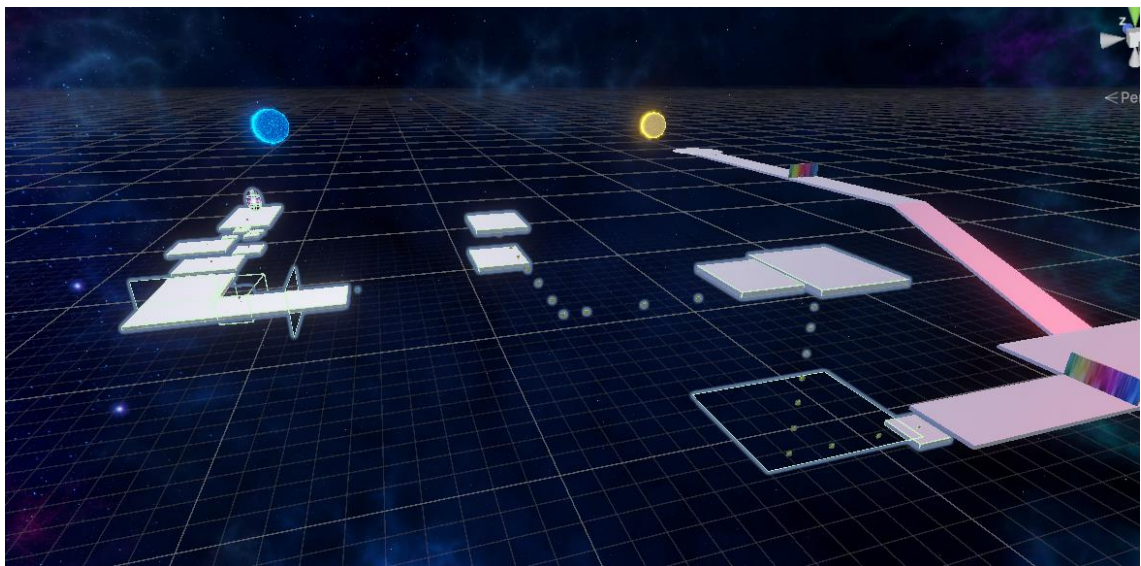


Figure 3.10: Section 2_2.

- 2_3: Subsection 2_3 emphasizes on avoiding floating asteroids as they move in different patterns. Additionally, 2_3 is broken down into 3 smaller sections, with each being a sort of challenge room. 2_3_1 starts off by avoiding asteroids moving up and down with a long pause after each time they move. 2_3_2 contains asteroids continuously moving left and right and 2_3_3 has a long straight with giant asteroid moving back and forth on a fast pace. The goal for the final section of 2_3 is for the player to move from one safe position to another all the while timing themselves to avoid the giant asteroids. At the end of the subsection, the player will be rewarded with a series of items, including an Astral key. In order to return back, the player can utilize a portal that will teleport them to the beginning of Section 2.

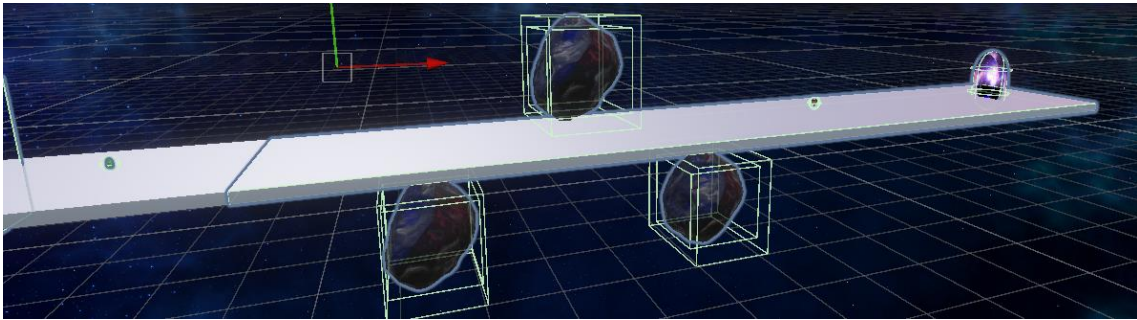


Figure 3.11: Section 2_3_1.

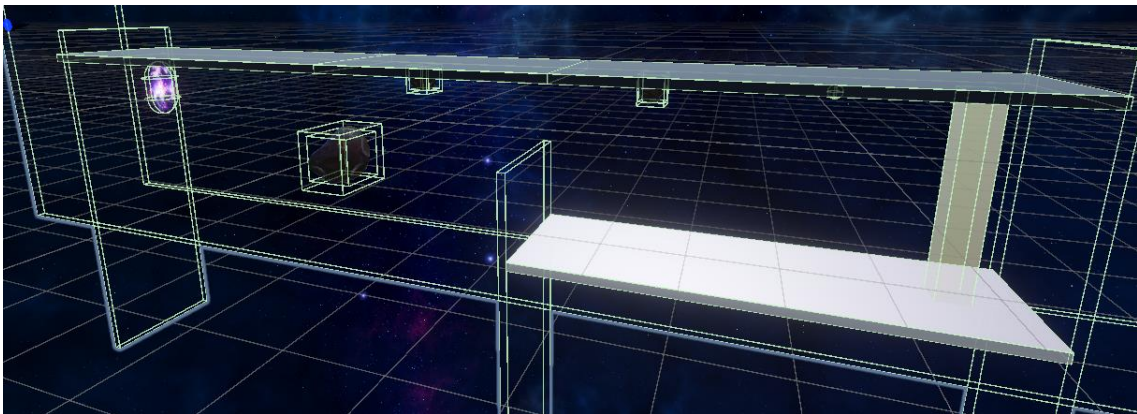


Figure 3.12: Section 2_3_2.

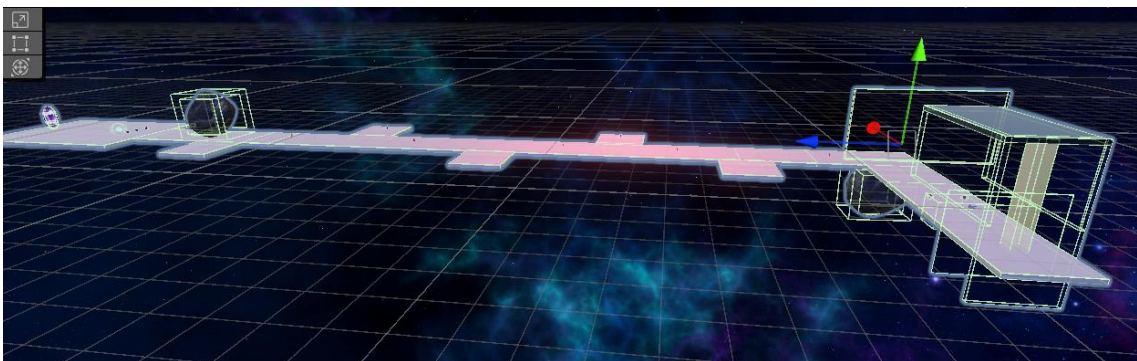


Figure 3.13: Section 2_3_3.

4 Game Assets

Throughout the course of the game the player can interact with numerous Game Objects in a variety of forms. From items to obstacles to even a Non-Playable Character (NPC) to converse with. All will be discussed in this chapter.

4.1 Items

Items are a universal aspect of any platformer video game. They can exist in multiple forms and serve a variety of roles when it comes to gameplay progression. Here I will analyze each item and their use case in the Project.

4.1.1 Stars

Scattered throughout the game are a large number of collectible stars of different colours. There are four variations of stars, each have unique value amount. From least to most valuable: Yellow, Red, Blue and Brown having values of 1, 2, 5 and 100 respectively. Once the player gathers a total of 100, they will be rewarded with an Additional Life (1-UP).

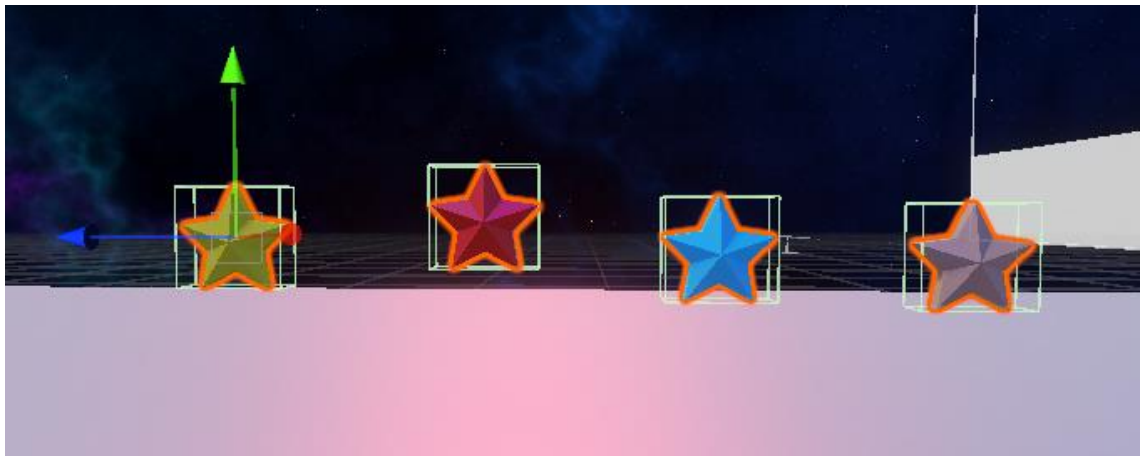


Figure 4.1: Visual representation of collectible stars.

4.1.2 Health Recovery

Since the game revolves around the use of a health bar system it is important to have methods for the player to regain health outside of dying and respawning. Each health recovery item restores 25% of the players health bar.

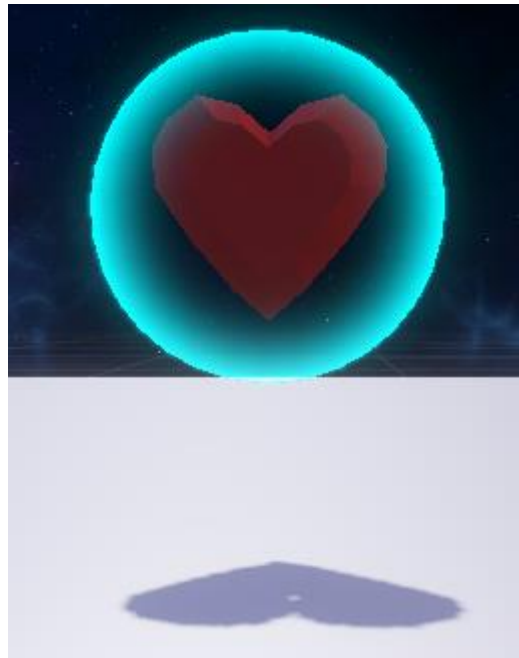


Figure 4.2: Visual representation of the Health Recovery Object

4.1.3 Speed-Up

On certain segments of the video game, to give the player an easier time with dodging asteroids, speed-up items have been placed. They temporarily increase the players' running speed for 10 seconds. While the speed-up is active, a slider will appear on the player's User Interface indicating the amount they have left before their speed returns to its default value.



Figure 4.3: Visual representation of the Speed-Up Object.

4.2 Portals

Due to technical restrictions and how the camera was created for this Project i had to create a way for the player to move across large distances or backtrack without forcing them to use limited visibility. Thus, we thought of the idea of implementing portals. Portals act as a one-way form of teleporting the player from one area to another instantly. It smoothed out the level design process and was also used in singling out sections as “one-room challenges”.

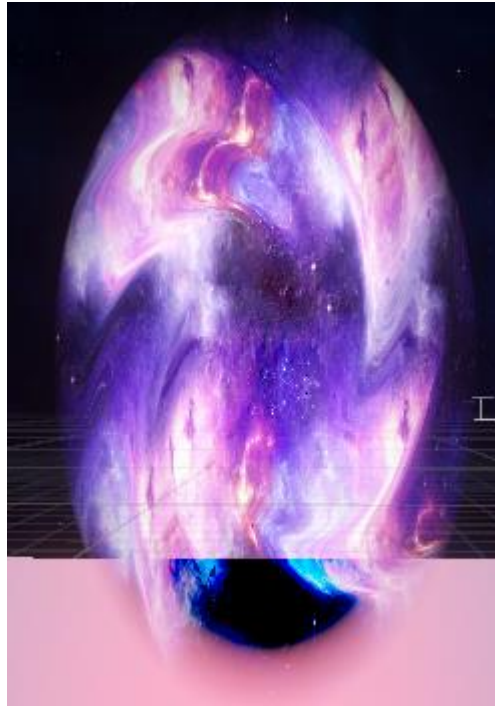


Figure 4.4: Visual representation of a Portal Object.

4.3 Asteroids

One of the mandatory requirements of this Project was the creation of objects that act as a form of a trap or environmental hazard for the player to navigate around. Since the theming of this video game was space, I decided the most fitting choice would be asteroids. In order to give variety as to how an asteroid can move, I've created a C# script that allows the developer in the Unity Editor to choose how far an object will move on the scene, on all three directional axis or alternatively, give it gravity and treat it like a rolling obstacle on a downhill surface (see reference X). Furthermore, I developed it to also control the amount of time this movement would take and how long it would take to perform the movement again once it finished the cycle. If the player comes into contact with an asteroid, they will take a certain amount of damage, which is indicated by the green health bar being replaced by red bar the more damage the player takes (see reference x).



Figure 4.5: Visual representation of an asteroid Object.

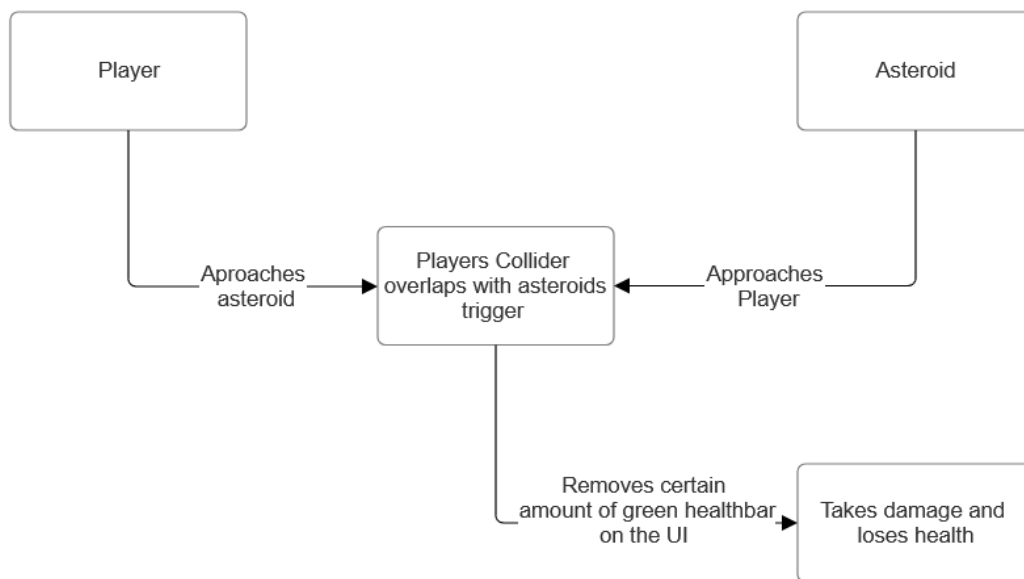


Figure 4.6: UML diagram displaying the interaction between an asteroid and the Player.

4.4 Moving Platform

A great way to add essence to any Platformer video game is the addition of moving platforms. Moving platforms have the potential to add verticality to a video game or a safe way for the player to move across great distances. Moreover, they can be an excellent way of challenging the player via moving from platform-to-platform challenges (see 1.2.2). In regard to execution, the script used to make the platforms move is similar to the one used for moving the asteroids (see reference Y).

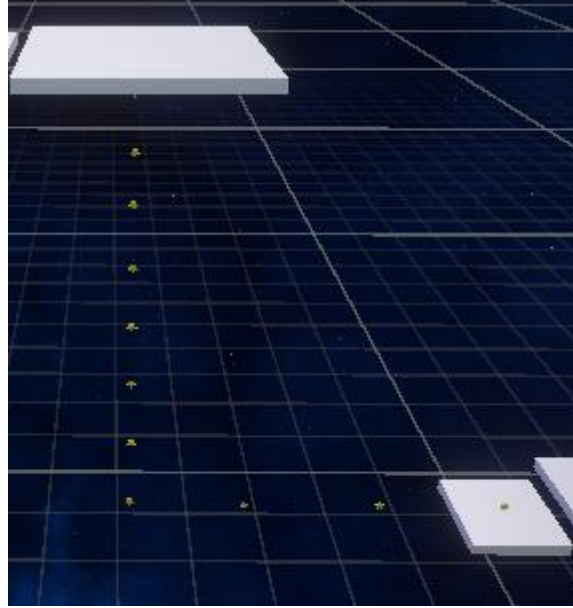


Figure 4.7: Visual representation of two moving platforms. The one on the left moves up and down, whilst the right one moves from left to right.

4.5 Astral Key

The Astral Keys are the main method behind the players progression of this video game. As the player acquires more Astral Keys they'll be able to open more Gates (see 2.6). Some gates require different amount of Astral Keys to open them and so the player is incentivized to search for all the Astral Keys in the Project (there is a total of 3 Astral Keys).

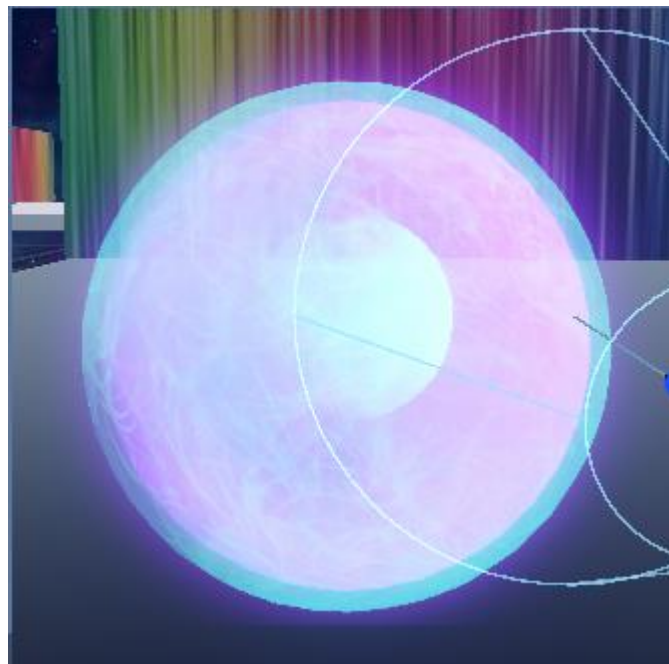


Figure 4.8: Visual representation of an Astral Key.

4.6 Gate

It's vital for a 3D Platformer to have ways of limiting the players' progression, whether it's for pacing reasons or introducing new mechanics before advancing, it's important to have a safeguard for such cases. In this Project, I tackled the problem by creating and placing Gates around the video games level. In order to open a Gate, the player must have acquired a certain number of Astral Keys (see 2.5). Once they have collected the necessary amount of Astral Keys, the Gate will open, allowing them to continue to the next area.

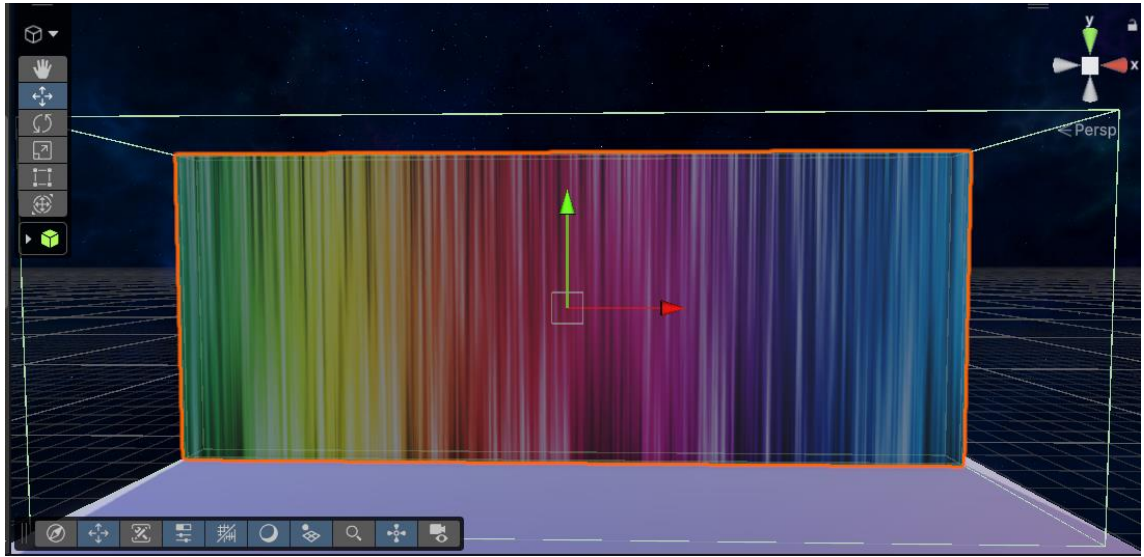


Figure 4.9: Visual representation of a Gate.

4.7 Light Pillar

Light Pillars are essential to this Project's main mechanic, gravity rotation. If the player comes into contact with a Light Pillar, their gravity will be flipped. It can be interacted from any gravitational rotation the player might be on and is used to teach the player of the Gravity mechanic and to guide players through visual cues.

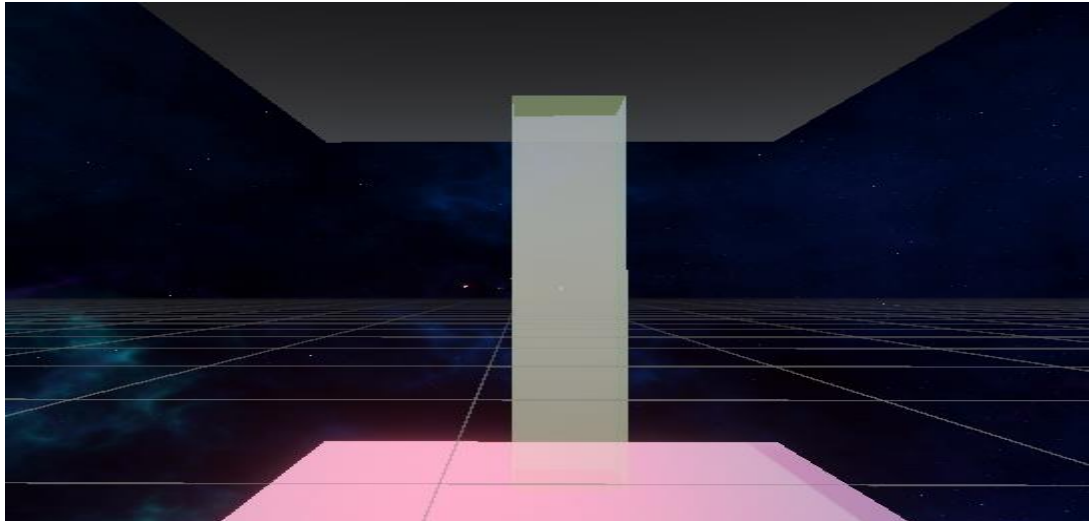


Figure 4.10: Visual representation of a Light Pillar. Regardless of where within the light pillar the player interacts with. Their gravity will be flipped.

4.8 NPC and Dialogue

Another mandatory goal for this Project was the addition of an interactable NPC or object whose purpose is to assist the player in giving them directions and useful information. This was achieved through creating the “Witch of The Stars” (see Story). She appears throughout the level giving information, directions on how to progress, and other forms of dialogue that contribute towards worldbuilding. While speaking to the NPC, the world will freeze, preventing anything from interacting with the Player as well as stopping the Player from moving during dialogue.



Figure 4.11: Visual representation of NPC and dialogue prompt.

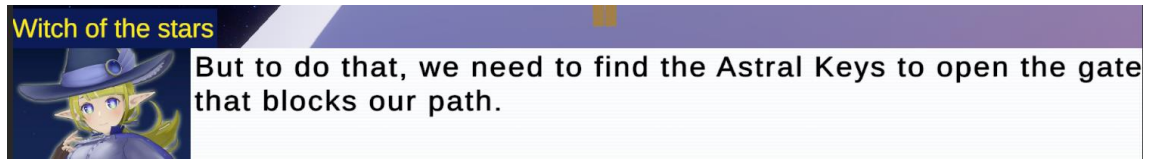


Figure 4.12: Example line of dialogue.

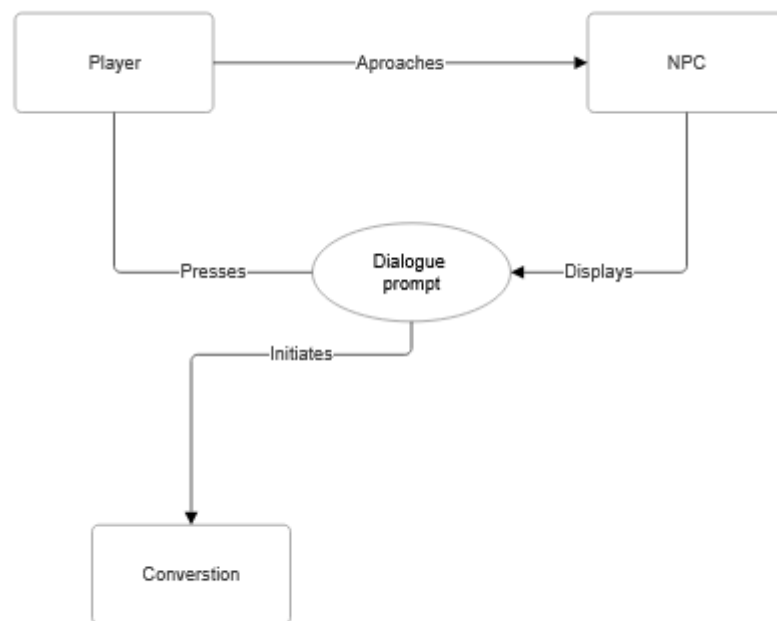


Figure 4.13: UML diagram displaying the interaction between an NPC and the Player.

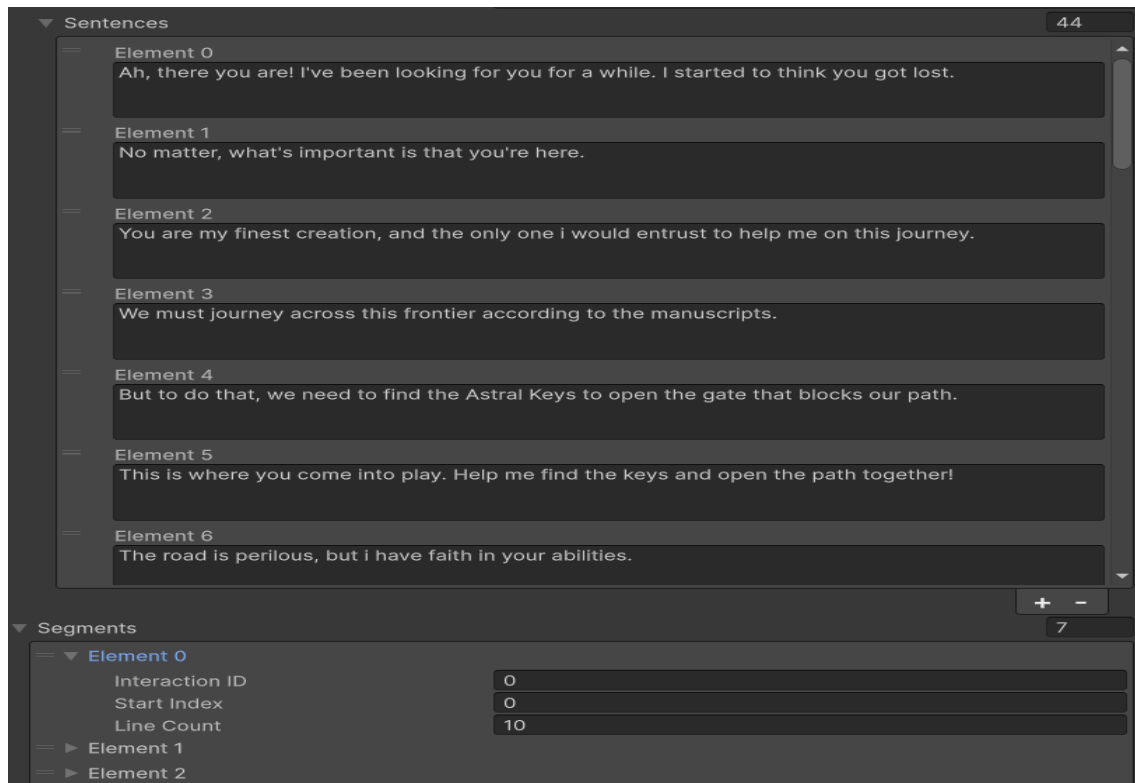


Figure 4.14: Small sample of NPC lines.

4.9 Player Character

The player Character is arguably the most important aspect of this Project. Through it, the player is able to move around the video game and interact with it. It also sets the foundation for the video games layout design to fit into the players move set and physics. Creating the player Character was the first and most challenging task when creating this Project as it set the pace for every other asset in the Project. I used a humanoid model from the Unity Asset Store (see Resources and References).

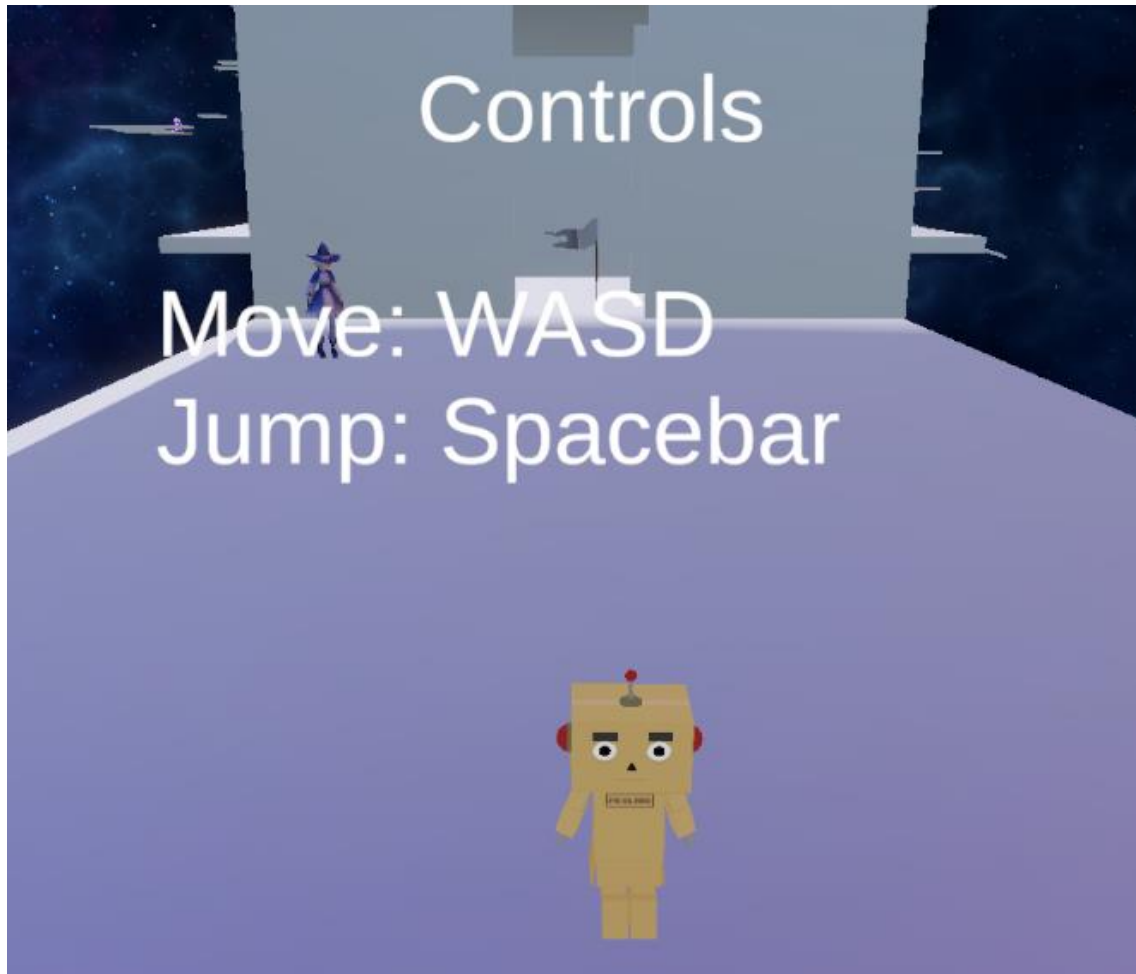


Figure 4.15: Visual representation of the player character and tutorial prompt.

5 User Interface (UI) and Camera

User interface is the method of displaying information to the user and it exists in many forms. From websites to applications to even a devices operating system, there exists a UI. Thus, it was of vital importance for this project to show clearly and in an easy-to-understand manner, information for the player. This includes the camera that displays the world to the player.

5.1 User Interface

In this Project, I implemented a UI containing basic information that is useful for the player, to be more precise, it contains: (see Figure 3.1)

- Health Bar (HP).
- Speed up duration (top right).
- Lives counter.
- Star counter.
- Astral Keys counter.
- Pause Menu as the circle on the top right.
- Dialogue section at the bottom of the screen.
- Useful information such as Tutorials, input prompts or messages at the center of the screen.



Figure 5.1: Visual representation of the Projects User Interface.

5.1.1 Pause Menu

By pressing the Pause Icon on the top right of the screen (see Figure 3.1), the player can temporarily pause the game. While paused, the video game will “Freeze”, meaning, all the GameObjects and player are unable to move to prevent game assets, such as asteroids, from interacting with the player while paused. In the Pause Menu, the player can interact with three objects on the screen: Two buttons and a slider.

- Resume: Closes the Pause Menu and “Unfreezes” the game
- Exit to main menu: Pressing this button will make an additional confirmational screen appear to ensure the player hasn’t pressed it by accident.
- Volume: This slider allows the player to adjust the volume of the music in the Project. Moving the slider to the right will increase the volume and moving it to the left will decrease the volume.

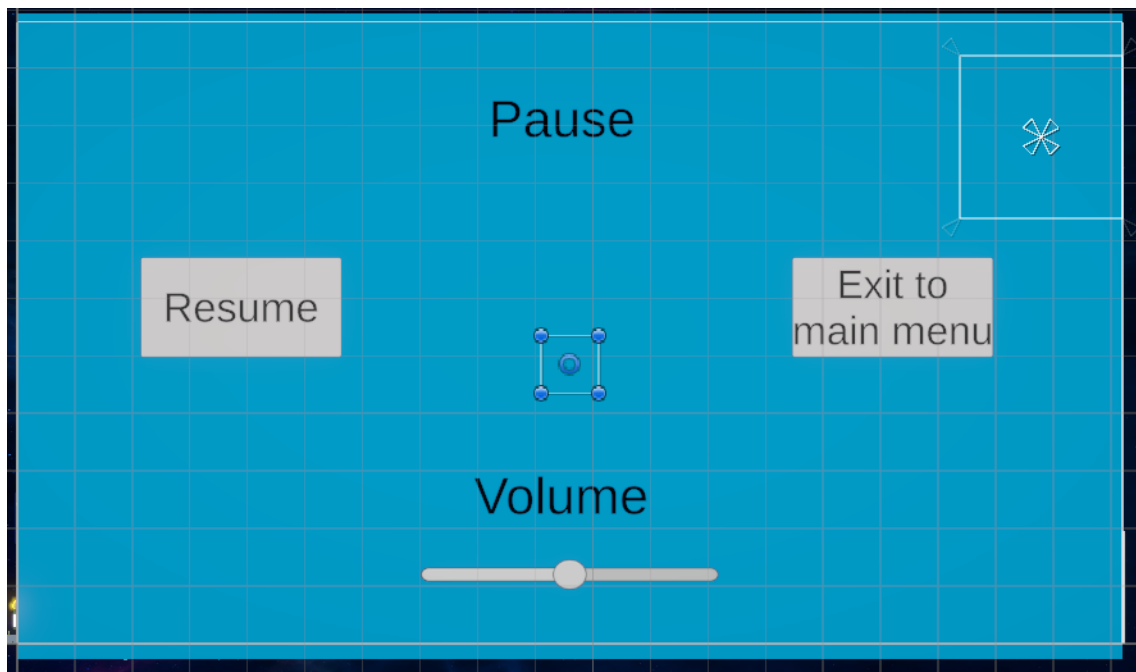


Figure 5.2: Visual representation of the Projects Pause Menu.

If the player presses “Exit to main menu” in the pause menu, a new menu will appear and present two choices: Press “No” and return to the pause menu or press “Yes” and return to the Title Screen. Doing so, will reset any progress made in the level by unloading the level scene and loading the Title Screen scene.

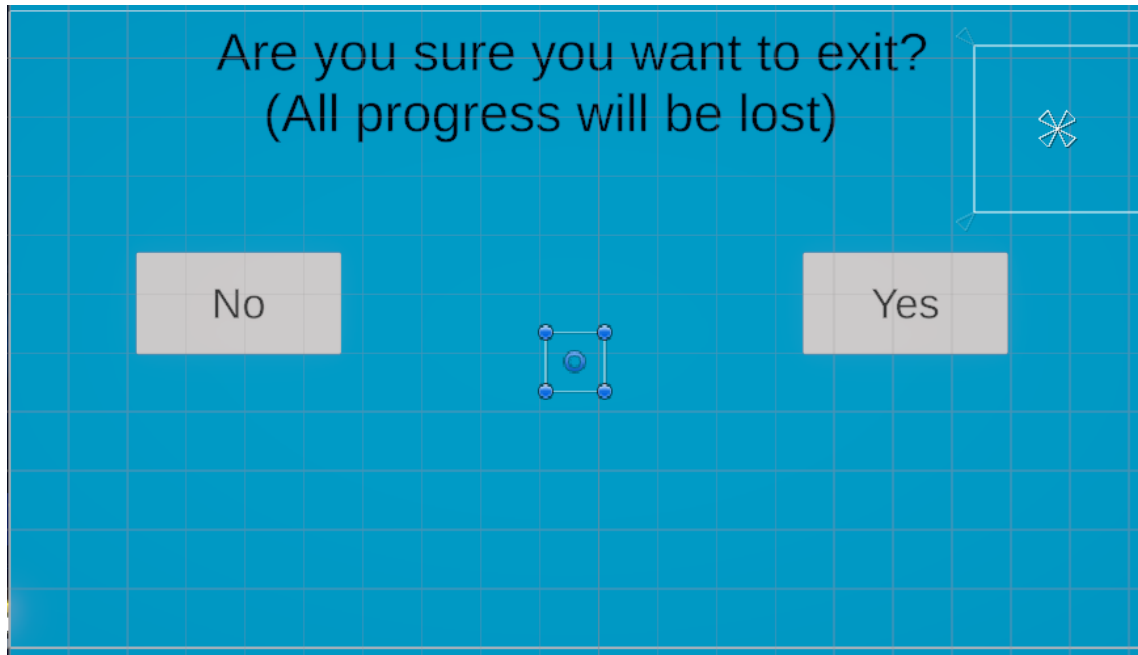


Figure 5.3: Confirmation Menu when the Player wishes to exit to the Main Title screen.

5.2 Camera

There is a plethora of ways to set up a camera. The player could have free 360-degree freedom to control the camera, a top-down view and many more. In this Project, I decided to go with a static camera that follows the player from behind. The only way to move the camera is by entering special triggers that move the camera by 90 degrees either to the left or right. The camera shift also works on inverted gravity. This way, I am able to control what is and isn't visible to the player and through it, make an overall better experience by suiting the level design to fit this camera style.

5.3 Game Over Sequence

If the Player runs out of lives (Life Counter reaches 00), the Project will begin its Game Over sequence. During the sequence, that lasts three seconds, the screen will display a message that they've run out of lives. Also, the music will temporarily pause for the duration of three seconds. Once that time has passed, the Project will reload the entire scene and the Player will be placed back in the beginning of the game, resetting all the items they have collected.

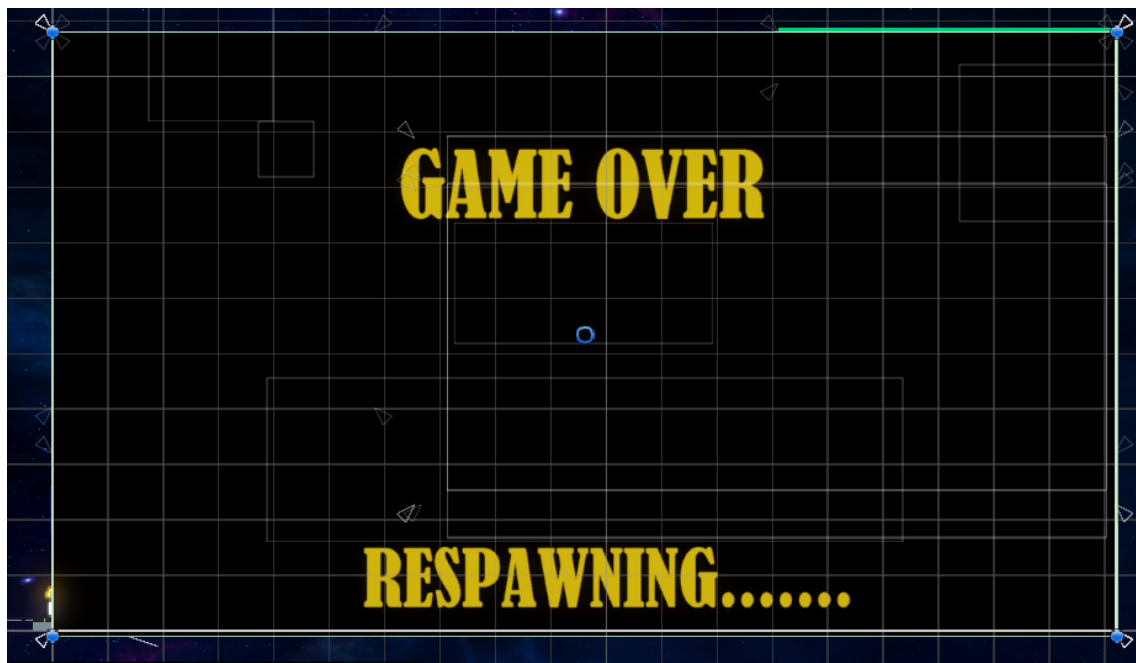


Figure 5.4: Visual representation of the Game Over screen.

6 Scripts

In this chapter, I will cover and analyze a number of scripts made for the Project. These scripts are the foundation in which the Project was constructed.

6.1 Move1

This script contains the custom movement for the player. It also handles the player's third-person camera by reading the player's X and Z coordinates and combines them into a singular value before applying it to "rb.linearVelocity" in order to not be influenced by the vertical velocity. Move1 also works with "PlayerGravity" (see 4.2) to function even when the gravity is flipped, this also includes the ability to jump even when inverted. The script also utilizes an animator to handle the player character's animations: Idle and running.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class move1 : MonoBehaviour
{
    Rigidbody rb;
    Animator animator;
    public PlayerGravity gravity;

    public float MoveSpeed = 18;

    public float Jump = 6;

    public Transform cam;

    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Start()
    {
        rb = GetComponent<Rigidbody>();
        animator = GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {
        if (DialogueManager.Instance != null && DialogueManager.Instance.IsDialogueActive)
        {
            rb.linearVelocity = new Vector3(0, rb.linearVelocity.y, 0); // stops the player from moving during dialogue
            return;
        }

        float horInput = Input.GetAxisRaw("Horizontal") * MoveSpeed;
        float verInput = Input.GetAxisRaw("Vertical") * MoveSpeed;

        Vector3 camForward = cam.forward;
        Vector3 camRight = cam.right;

        camForward.y = 0;
        camRight.y = 0;

        Vector3 forwardRelative = verInput * camForward;
        Vector3 rightRelative = horInput * camRight;

        Vector3 moveDir = forwardRelative + rightRelative;

        rb.linearVelocity = new Vector3(moveDir.x, rb.linearVelocity.y, moveDir.z);

        if (Input.GetButtonDown("Jump") && IsGrounded())
        {
            //if (Input.GetButtonDown("Jump") && Mathf.Approximately(rb.linearVelocity.y, 0))
            {
                float jumpDir = gravity.IsFlipped ? -1f : 1f;
                rb.linearVelocity = new Vector3(rb.linearVelocity.x, Jump * jumpDir, rb.linearVelocity.z);
            }

            //if (Input.GetButtonDown("Jump") && Mathf.Approximately(rb.linearVelocity.y, 0)) rb.linearVelocity = new Vector3(rb.linearVelocity.x, Jump, rb.linearVelocity.z);
            // if (rb.linearVelocity.magnitude > 0.1) transform.forward = new Vector3(rb.linearVelocity.x, 0, rb.linearVelocity.z);

            animator.SetFloat("RunningSpeed", Mathf.Abs(Input.GetAxisRaw("Horizontal"))) + Mathf.Abs(Input.GetAxisRaw("Vertical")));

            Vector3 flatVel = new Vector3(rb.linearVelocity.x, 0, rb.linearVelocity.z);
            if (flatVel.sqrMagnitude > 8.81f)
            {
                Vector3 upDir = gravity.IsFlipped ? Vector3.down : Vector3.up;
                transform.rotation = Quaternion.LookRotation(flatVel, upDir);
            }
        }

        bool IsGrounded()
        {
            Vector3 downDir = gravity.IsFlipped ? Vector3.up : Vector3.down;
            return Physics.Raycast(transform.position, downDir, 0.6f);
        }
    }
}

```

Figure 6.1: move1 script.

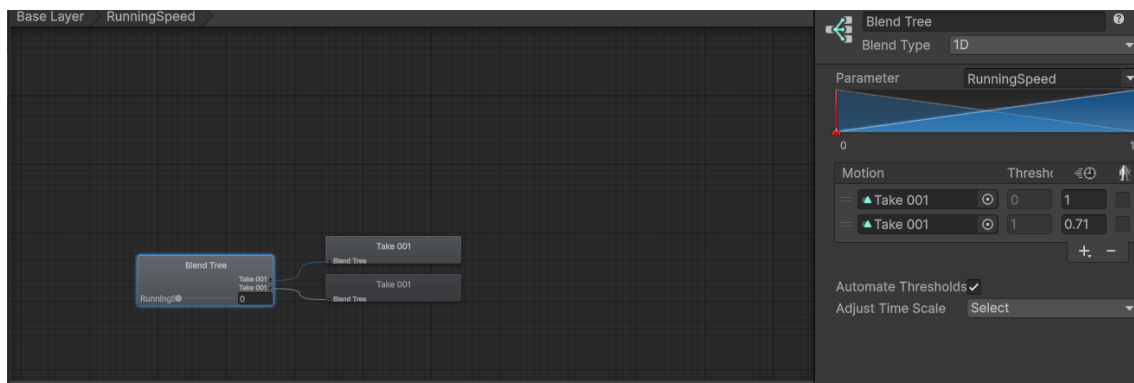


Figure 6.2: Player character animator.

6.2 PlayerGravity

This script is about changing the Players' gravity by 180 degrees and my inspiration to make this Project. Here, we can adjust the scale of gravity force (gravityScale). To distinguish when the player is inverted or not, there is a bool property named "IsFlipped". Every time the script would be executed (when the player interacts with a light pillar or presses the key B), the bool will change from false to true and vice versa. This value is highly important for the respawn script (see x reference) to function properly as it keeps track of the current gravity and if the player needs to respawn it can correctly re-orient them. It should be noted that the gravity inversion does not happen instantaneously. Instead, it has a buffer of 0.3 seconds to give the character flow when performing it.

```
using UnityEngine;

public class PlayerGravity : MonoBehaviour
{
    public static PlayerGravity Instance;
    public float gravityScale = -9.81f;
    private Rigidbody rb;
    public bool IsFlipped { get; private set; }
    void Start() => rb = GetComponent<Rigidbody>();

    private bool IsPlayerFlipped = false;

    void Awake()
    {
        Instance = this;
    }

    void FixedUpdate()
    {
        rb.AddForce(Vector3.up * gravityScale, ForceMode.Acceleration);
    }

    public void FlipGravity()
    {
        if (IsPlayerFlipped == false)
        {
            IsPlayerFlipped = true;
        }
        else if (IsFlipped == true)
        {
            IsPlayerFlipped = false;
        }
        else
        {
            IsPlayerFlipped = true;
        }
        gravityScale *= -1f;
        IsFlipped = !IsFlipped;
        StartCoroutine(RotatePlayer());
    }

    private System.Collections.IEnumerator RotatePlayer()
    {
        Quaternion startRot = transform.rotation;
        Quaternion endRot = startRot * Quaternion.Euler(180f, 0f, 0f);
        float t = 0f;
        float duration = 0.3f;

        while (t < duration)
        {
            transform.rotation = Quaternion.Slerp(startRot, endRot, t / duration);
            t += Time.deltaTime;
            yield return null;
        }
        transform.rotation = endRot;
    }

    public bool PlayerFlipped ()
    {
        return IsPlayerFlipped;
    }
}
```

Figure 6.3: PlayerGravity script.

6.3 Manage

This script acts as the main manager of this Project. It's responsible for setting up the User Interface when the scene is loaded as well as establishing connections between the UI and the various scripts of the Project that interact with it, such as collectibles, Lives, the Players healthbar and the pause menu. It also keeps track of the number of Astral Keys the Player has when they interact with a Gate.

```

using UnityEngine;
using TMPro;
using UnityEngine.SceneManagement;

public class Manage : MonoBehaviour
{
    public static Manage Instance;

    public TMP_Text starsText;
    public TMP_Text LivesText;
    public TMP_Text KeysText;

    public int StarCount { get; private set; }
    public int LivesCount { get; private set; }
    public int KeysCount { get; private set; }

    public Respawn respawn;
    public GameObject GameOver;
    public GameObject DialogueBox;
    public GameObject DialogueText;
    public GameObject NpcIcon;
    public GameObject PromptDialogue;
    public GameObject NpcNameText;
    public GameObject NpcNameBg;
    public GameObject TutorialText;
    public GameObject SpeedDuration;
    public GameObject NotSpeedDuration;
    public GameObject GameEnd;
    public GameObject NotEnoughKeys;
    public GameObject GravityTutorial;

    void Awake()
    {
        GameEnd.SetActive(false);
        GameOver.SetActive(false);
        DialogueBox.SetActive(false);
        DialogueText.SetActive(false);
        NpcIcon.SetActive(false);
        PromptDialogue.SetActive(false);
        NpcNameText.SetActive(false);
        NpcNameBg.SetActive(false);
        TutorialText.SetActive(false);
        SpeedDuration.SetActive(false);
        NotSpeedDuration.SetActive(false);
        NotEnoughKeys.SetActive(false);
        GravityTutorial.SetActive(false);

        if (Instance != null && Instance != this)
        {
            Destroy(gameObject);
            return;
        }
        Instance = this;

        // Reads values from the ui
        if (starsText != null && int.TryParse(starsText.text, out int parsedStars))
        {
            StarCount = parsedStars;
        }

        if (LivesText != null && int.TryParse(LivesText.text, out int parsedLives))
        {
            LivesCount = parsedLives;
        }

        if (KeysText != null && int.TryParse(KeysText.text, out int parsedKeys))
        {
            KeysCount = parsedKeys;
        }
    }
}

```

Figure 6.4: First part of Manage script.

```

public void AddStar(int amount = 1)
{
    StarCount += amount;

    if (StarCount >= 100)
    {
        StarCount = StarCount - 100;
        AddLife();
    }

    if (starsText != null)
    {
        starsText.text = StarCount.ToString("00");
    }
}

public void AddKey(int amount = 1)
{
    KeysCount += amount;

    if (KeysText != null)
    {
        KeysText.text = KeysCount.ToString("00");
    }
}

public int CheckKey (int currentKeys)
{
    currentKeys = KeysCount;
    return currentKeys;
}

public void AddLife(int amount = 1)
{
    LivesCount += amount;

    if (LivesText != null)
    {
        LivesText.text = LivesCount.ToString("00");
    }
    if (LivesCount == 0)
    {
        LivesCount = 5;
        if (LivesText != null)
        {
            LivesText.text = LivesCount.ToString("00");
        }
        StarCount = 0;
        if (starsText != null)
        {
            starsText.text = StarCount.ToString("00");
        }
        if (respawn != null)
        {
            respawn.RespawnAtStart();
        }
        GameOver.SetActive(true);

        if (Music.Instance != null)
        {
            Music.Instance.PauseMusic();
        }
        StartCoroutine(GameOverUI(3f));

        return;
    }
}

private System.Collections.IEnumerator GameOverUI(float delay)
{
    yield return new WaitForSeconds(delay);
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
}
}

```

Figure 6.5: Second part of Manage script.

6.4 Checkpoint

Due to the nature of the Platformer genre, having a checkpoint system in the Video Game is mandatory. Should the Player fail a jump or lose all their health, they must have a method of going back to a previous spot on the map and not place them back at the very beginning of the level. Every checkpoint has a bool value to know whether that checkpoint is inverted or not which is important for the “Respawn” script (see x reference).

```
using UnityEngine;

public class Checkpoint : MonoBehaviour
{
    public static Checkpoint Instance;
    public bool IsInverted;
    public PlayerGravity gravity;
    public GameObject onCheckpointEffect;
    private BoxCollider boxCollider;

    void Awake()
    {
        Instance = this;
    }

    void Start()
    {
        boxCollider = GetComponent<BoxCollider>();
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            float EffectOrientation = gravity != null && gravity.IsFlipped ? 90f : -90f;
            GameObject effect = Instantiate(onCheckpointEffect, transform.position, Quaternion.Euler(EffectOrientation, 0f, 0f));
            Destroy(effect, 0.7f);

            Vector3 worldCenter = transform.TransformPoint(boxCollider.center);
            Respawn respawn = other.GetComponent<Respawn>();
            if (respawn != null)
            {
                respawn.SetRespawnPoint(worldCenter);
                respawn.SetCheckpointInversion(IsInverted);
            }
        }
    }

    public bool checkPlayerDiff (bool difference)
    {
        difference = IsInverted;
        Debug.Log(difference);
        return difference;
    }
}
```

Figure 6.6: Checkpoint script.

6.5 Respawn

Under normal circumstances, creating a script that respawns the Player is a simple matter. However, the addition of inverted gravity has brought complications. If the Player were to activate a checkpoint while being under normal gravity, and die while being inverted, they would respawn while having the opposite gravity than intended. This led to situations where the Player would either phase through the floor or if there isn't a ceiling, fall off the level. In either case, this would potentially permanently cause the Player into a death loop until they run out of lives and trigger a Game Over. To prevent this from happening, fail safes have been created to keep track of both the players' current gravitational orientation as well as the last checkpoints. When the Player dies, the script takes "IsPlayerFlipped" (see 4.2) and "IsInverted" (see 4.4.) and compares them. If they don't match, meaning the Player gravity is different from the checkpoint's, the script will call FlipGravity and correct the difference, making both gravitations match. Upon respawning, a special effect will play, which the code has modified to rotate it by 90 degrees in the X-axis due to mismatch from the original prefab of choice. In earlier versions of the Project, one reason for respawning was if the Players character had passed a certain Y or -Y values (to account for both gravity rotations). This version was scrapped and replaced by invisible gameObject triggers that would call Respawn if the player collided with them. That way, instead of limiting myself in the Y-axis by two values, the invisible triggers offered flexibility when it came to vertical design. I should emphasize from testing that misuse of the triggers could lead to small gaps being formed, which if the Player accidentally went through one, they'd fall endlessly with no safety net. Alternatively, if more than one triggers exist at the same place and should the player fall to that spot, there is a risk of Respawn being called more than once, leading to the player losing more lives than intended.

```

using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Collections;

public class Respawn : MonoBehaviour
{
    public static Respawn Instance;
    public Image greenHealth;
    //public float PositiveThreshold;
    //public float NegativeThreshold;
    public GameObject onRespawnEffect;
    public PlayerGravity gravity;
    private Vector3 respawnPoint;
    public Vector3 gameOverCoords;

    // private bool flipped;
    //private bool difference;
    //private bool InvertedCheckpoint;
    private float wait = 0.1f;
    //public PlayerGravity gravity;
    private bool lastCheckpointInverted = false;

    void Awake()
    {
        Instance = this;
    }

    void Start()
    {
        respawnPoint = transform.position; // first spawn is the starting position
    }

    public void RespawnAtStart()
    {
        transform.position = gameOverCoords;
        Rigidbody rb = GetComponent<Rigidbody>();
        if (rb != null)
            rb.linearVelocity = Vector3.zero;
    }

    public void SetRespawnPoint(Vector3 newPoint)
    {
        respawnPoint = newPoint;
    }

    public void RespawnPlayer()
    {
        transform.position = respawnPoint;

        Rigidbody rb = GetComponent<Rigidbody>();
        if (rb != null)
            rb.linearVelocity = Vector3.zero;
    }
}

```

Figure 6.7: First part of Respawn.

```

/* void FixedUpdate()
{
    if (transform.position.y < NegativeThreshold || transform.position.y > PositiveThreshold)
    {
        deathRespawn();
    }
}

} */

public void SetCheckpointInversion(bool inverted)
{
    lastCheckpointInverted = inverted;
}

public void deathRespawn()
{
    bool flipped = PlayerGravity.Instance.PlayerFlipped();

    if (lastCheckpointInverted != flipped)
    {
        gravity.FlipGravity();
    }
    Correction();
}

public void Correction()
{
    greenHealth.fillAmount = 1f;
    Manage.Instance.AddLife(-1);

    Rigidbody rb = GetComponent<Rigidbody>();
    if (rb != null)
        rb.linearVelocity = Vector3.zero;

    transform.position = respawnPoint;

    float EffeOrientation = gravity != null && gravity.IsFlipped ? 90f : -90f;
    GameObject effect = Instantiate(onRespawnEffect, transform.position, Quaternion.Euler(EffeOrientation, 0f, 0f));
    Destroy(effect, 2f);
}
}

```

Figure 6.8: Second part of Respawn.

6.6 ObjectMovement

To add variety to the Video Game, I designed a script that manipulates the position of GameObjects within the level. In the Unity Inspector, you can freely edit the values of how far you want an Object to go and in what direction. The script allows for movement in multiple axis simultaneously. When the Object reaches the new set of coordinates, it will wait before returning to its original position by backtracking the same amount as it moved before moving again in the set direction. You can set how long the loop last as well as how long the object should “pause” in the new location before returning. To make the Player able to use an Object with this script as a moving platform, I’ve set it so when the Player collides with the object, it sets the Player as its parent object and they exit, it cancels it. It is also used in the Asteroid Objects as moving obstacles.

```

using UnityEngine;

public class ObjectMovement : MonoBehaviour
{
    public float xOffset = 0f;
    public float yOffset = 0f;
    public float zOffset = 0f;
    public float duration = 2f;
    public float pauseDuration = 1f;

    private Vector3 startPosition;
    private Vector3 targetPosition;

    void Start()
    {
        startPosition = transform.position;
        targetPosition = startPosition + new Vector3(xOffset, yOffset, zOffset);
        StartCoroutine(MoveLoop());
    }

    private System.Collections.IEnumerator MoveLoop()
    {
        while (true)
        {
            yield return StartCoroutine(MoveTo(targetPosition));
            yield return new WaitForSeconds(pauseDuration);
            yield return StartCoroutine(MoveTo(startPosition));
            yield return new WaitForSeconds(pauseDuration);
        }
    }

    private System.Collections.IEnumerator MoveTo(Vector3 destination)
    {
        Vector3 from = transform.position;
        float t = 0f;

        while (t < duration)
        {
            transform.position = Vector3.Lerp(from, destination, t / duration);
            t += Time.deltaTime;
            yield return null;
        }

        transform.position = destination;
    }

    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            collision.transform.SetParent(transform);
        }
    }

    private void OnCollisionExit(Collision collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            collision.transform.SetParent(null);
        }
    }
}

```

Figure 6.9: ObjectMovement script.

6.7 Gate

Each Gate requires a certain amount of Astral Keys, that can be set in the Unity Inspector, to open. As the Player approaches a Gate, a prompt will appear that displays a button that the Player needs to press to interact with the Gate. In the scenario that the Player has an insufficient number of Astral Keys when interacting with the Gate, a message will appear on the screen to inform them that they need to collect more Astral Keys as seen in the figures below and the Gate will remain in the path. If the Player does have enough Astral Keys, the Gate will move upwards and allow them to progress deeper into the level.

```
using UnityEngine;

public class Gate : MonoBehaviour
{
    private bool dialogueRange = false;
    public GameObject PromptDialogue;
    public GameObject NotEnoughKeys;

    public int requiredKeys;
    private int GateOpen;

    public float xOffset = 0f;
    public float yOffset = 0f;
    public float zOffset = 0f;
    public float duration = 2f;
    public float pauseDuration = 1f;

    private Vector3 startPosition;
    private Vector3 targetPosition;

    void Start()
    {
        startPosition = transform.position;
        targetPosition = startPosition + new Vector3(xOffset, yOffset, zOffset);
    }
    // Update is called once per frame
    void Update()
    {
        if (dialogueRange && Input.GetKeyDown(KeyCode.F))
        {
            GateOpen = Manage.Instance.CheckKey(GateOpen);

            if (GateOpen == requiredKeys)
            {
                StartCoroutine(MoveLoop());
            }
            else
            {
                PromptDialogue.SetActive(false);
                NotEnoughKeys.SetActive(true);
            }
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            dialogueRange = true;
            PromptDialogue.SetActive(true);
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            dialogueRange = false;
            PromptDialogue.SetActive(false);
            NotEnoughKeys.SetActive(false);
        }
    }

    private System.Collections.IEnumerator MoveLoop()
    {
        while (true)
        {
            yield return StartCoroutine(MoveTo(targetPosition));
            yield return new WaitForSeconds(pauseDuration);
            //yield return StartCoroutine(MoveTo(startPosition));
            //yield return new WaitForSeconds(pauseDuration);
        }
    }

    private System.Collections.IEnumerator MoveTo(Vector3 destination)
    {
        Vector3 from = transform.position;
        float t = 0f;

        while (t < duration)
        {
            transform.position = Vector3.Lerp(from, destination, t / duration);
            t += Time.deltaTime;
            yield return null;
        }

        transform.position = destination;
    }
}
```

Figure 6.10: Gate script.

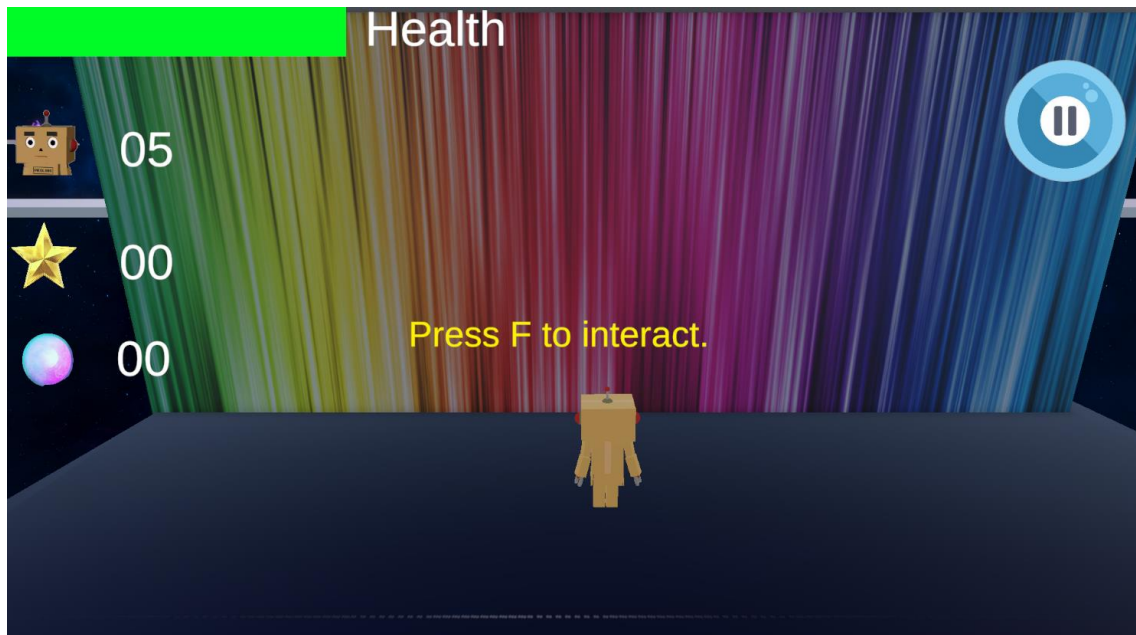


Figure 6.11: Prompt for the player to interact with the Gate.

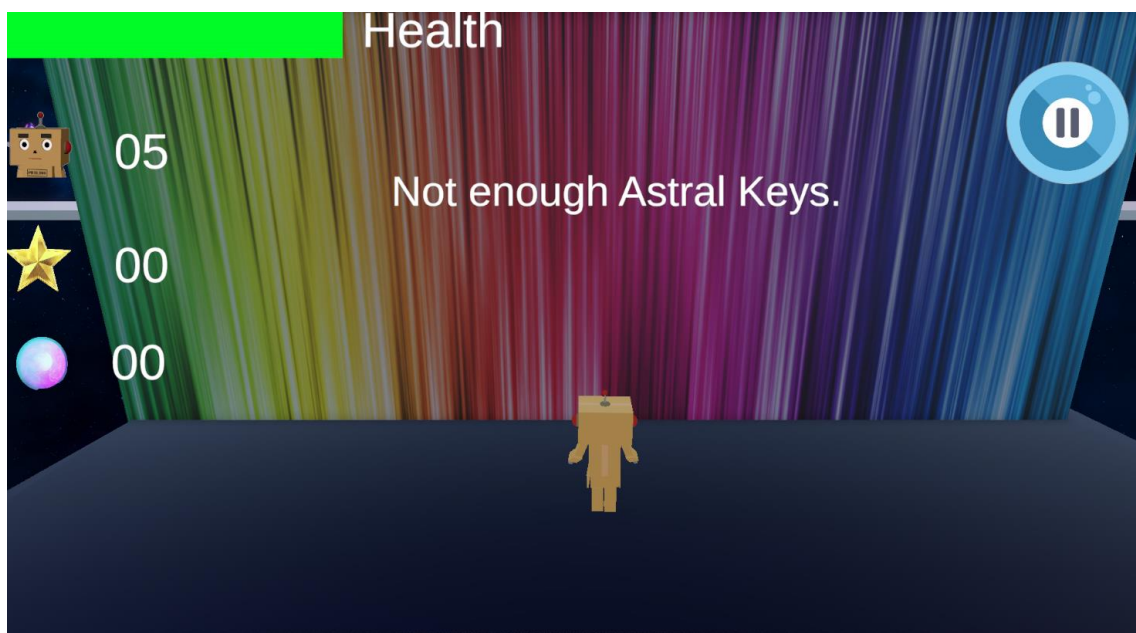


Figure 6.12: Displayed message if the Player is missing Astral Keys needed.

6.8 Asteroid

Asteroids are the main obstacle of this Project. They contain two colliders: a trigger that houses the asteroid script, and a physical collider so the Player doesn't faze through it. When the Player collides with an asteroid's trigger, the script is called and removes an amount from the player's healthbar, that can be edited in the Unity Inspector. If their healthbar reaches 0 or below, it will call Respawn script.

```
using UnityEngine;
using UnityEngine.UI;

public class Asteroid : MonoBehaviour
{
    public Image greenHealth;
    public float Damage;

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            if (greenHealth != null)
            {
                greenHealth.fillAmount -= Damage;
                if (greenHealth.fillAmount <= 0f)
                {
                    Respawn.Instance.deathRespawn(); // calls respawn script
                }
            }
        }
    }
}
```

Figure 6.13: Asteroid script.

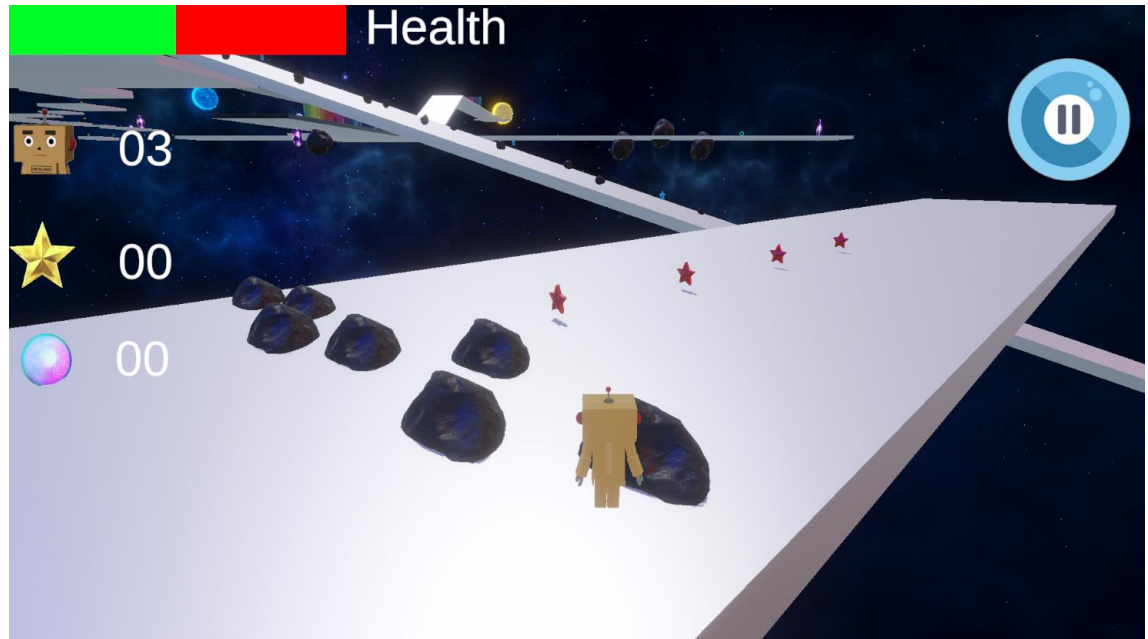


Figure 6.14: Instance of Player getting damaged by an asteroid.

6.9 Dialogue scripts

The dialogue system is broken up into three scripts, each performing a specific task in managing the dialogue functions.

6.9.1 Dialogue

Dialogue is responsible for containing all the data regarding the contents of the dialogue. It stores all the lines of dialogue into a queue and divides them into separate segments to use them independently for different cases (see figure 2.12).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[System.Serializable]
public class Dialogue
{
    public string name;

    [TextArea(3, 10)]
    public string[] sentences;
}

[System.Serializable]
public class DialogueSegment
{
    public int interactionID;
    public int startIndex;
    public int lineCount;
}
```

Figure 6.15: Dialogue script.

6.9.2 StartDialogue

This script is tied to the Witch of The Stars trigger. It acts as the starting point of the dialogue. It prompts the player to start the conversation and if initiated, will set up the dialogue UI (see Figures 2.10 and 2.11) and call DialogueManager (see 4.9.3). It will also transfer the data from Dialogue script alongside the selected segment which to display.

```

using UnityEngine;
using TMPro;

public class StartDialogue : MonoBehaviour

{
    private bool dialogueRange = false;
    public GameObject DialogueBox;
    public TMP_Text DialogueText;
    public GameObject NpcIcon;
    public GameObject PromptDialogue;
    public TMP_Text NpcNameText;
    public GameObject NpcNameBcg;
    public int Interaction;
    public Dialogue dialogue;
    public DialogueSegment[] segments;

    private bool waitingForRelease = false;

    void Update()
    {
        bool managerActive = DialogueManager.Instance != null && DialogueManager.Instance.IsDialogueActive;

        if (waitingForRelease)
        {
            if (!managerActive && Input.GetKey(KeyCode.F))
            {
                waitingForRelease = false;
                return;
            }
        }

        if (dialogueRange && Input.GetKeyDown(KeyCode.F) && !managerActive)
        {
            DialogueBox.SetActive(true);
            DialogueText.gameObject.SetActive(true);
            NpcIcon.SetActive(true);
            NpcNameText.gameObject.SetActive(true);
            NpcNameBcg.SetActive(true);

            DialogueSegment segment = GetSegment(Interaction);
            if (segment != null)
            {
                DialogueManager.Instance.StartDialogueSegment(
                    dialogue, segment.startIndex, segment.lineCount,
                    DialogueBox, DialogueText, NpcNameText, NpcNameBcg, NpcIcon);
                waitingForRelease = true;
            }
        }
    }

    private DialogueSegment GetSegment(int interactionID)
    {
        foreach (var s in segments)
        {
            if (s.InteractionID == interactionID)
            {
                return s;
            }
        }
        return null;
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            dialogueRange = true;
            PromptDialogue.SetActive(true);
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            dialogueRange = false;
            PromptDialogue.SetActive(false);
        }
    }
}

```

Figure 6.16: StartDialogue script.

6.9.3 DialogueManager

DialogueManager inherits the data from Dialogue script, given by StartDialogue and carries out the rest of the dialogue sequence. It begins by setting universal time to 0, freezing all GameObjects on the scene, including the Player. During the dialogue, each line is presented by

slowly displaying each letter of the line. By pressing F (which is the same button for initiating the dialogue sequence) will cycle through each line of the given segment or skip the current lines display animation if it hasn't finished. Once it reaches the end of the segment and the Player presses F, the conversation will end, the dialogues UI will disappear, and time will return to normal, allowing the player and different Objects to move again.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class DialogueManager : MonoBehaviour
{
    public static DialogueManager Instance;
    public float letterDelay = 0.03f;

    private Queue<string> sentences = new Queue<string>();
    private bool isDialogueActive = false;
    private bool isTyping = false;
    private Coroutine typingCoroutine;

    private GameObject dialogueBox;
    private TMP_Text dialogueText;
    private TMP_Text npcNameText;
    private GameObject npcNameBcg;
    private GameObject npcIcon;

    private string currentSentence;

    public bool IsDialogueActive => isDialogueActive;

    void Awake()
    {
        Instance = this;
    }

    void Update()
    {
        if (isDialogueActive && Input.GetKeyDown(KeyCode.F))
        {
            if (isTyping)
            {
                StopCoroutine(typingCoroutine);
                dialogueText.text = currentSentence;
                isTyping = false;
            }
            else
            {
                DisplayNextSentence();
            }
        }
    }
}

```

Figure 6.17: Part one of DialogueManager.

```

public void StartDialogueSegment(Dialogue dialogue, int startIndex, int lineCount,
    GameObject box, TMP_Text textField, TMP_Text nameField, GameObject nameBcg, GameObject icon)
{
    dialogueBox = box;
    dialogueText = textField;
    npcNameText = nameField;
    npcNameBcg = nameBcg;
    npcIcon = icon;

    npcNameText.text = dialogue.name;
    sentences.Clear();

    int endIndex = Mathf.Min(startIndex + lineCount, dialogue.sentences.Length);
    for (int i = startIndex; i < endIndex; i++)
    {
        sentences.Enqueue(dialogue.sentences[i]);
    }

    Time.timeScale = 0f;

    isDialogueActive = true;
    DisplayNextSentence();
}

private void DisplayNextSentence()
{
    if (sentences.Count == 0)
    {
        EndDialogue();
        return;
    }

    currentSentence = sentences.Dequeue();
    typingCoroutine = StartCoroutine(TypeSentence(currentSentence));
}

private IEnumerator TypeSentence(string sentence)
{
    isTyping = true;
    dialogueText.text = "";

    foreach (char letter in sentence)
    {
        dialogueText.text += letter;
        yield return new WaitForSecondsRealtime(letterDelay);
    }

    isTyping = false;
}

private void EndDialogue()
{
    isDialogueActive = false;
    dialogueBox.SetActive(false);
    dialogueText.gameObject.SetActive(false);
    if (npcIcon != null) npcIcon.SetActive(false);
    if (npcNameBcg != null) npcNameBcg.SetActive(false);
    if (npcNameText != null) npcNameText.gameObject.SetActive(false);

    Time.timeScale = 1f;
}

```

Figure 6.18: Part two of DialogueManager.

6.10 FollowPlayer

This script handles the Player's camera. It follows the Player every frame and changes the camera's rotation whenever the Player's gravity is flipped, or they pass a trigger that rotates it to the left or right. If the player switches gravity, the camera will rotate up or down by 180 degrees while the rotation triggers will rotate the camera by 90 degrees sideways. The camera is designed to strictly follow the player, making it static to only face a single direction.

```
using UnityEngine;

public class FollowPlayer : MonoBehaviour
{
    public Transform Player;
    public PlayerGravity gravity;
    public float flipDuration = 0.3f;
    public float sideRotationDuration = 0.3f;
    public float sideRotationAngle = 90f;

    private Quaternion baseRotation;
    private bool isFlipped = false;
    private bool isSideRotated = false;
    private bool lastRotateLeft = false;

    void Start()
    {
        baseRotation = transform.rotation;
    }

    void Update()
    {
        transform.position = Player.position;

        if (gravity.IsFlipped != isFlipped)
        {
            isFlipped = gravity.IsFlipped;
            StopAllCoroutines();
            StartCoroutine(RollCamera(isFlipped));
        }
    }

    private System.Collections.IEnumerator RollCamera(bool flipped)
    {
        Quaternion startRot = transform.rotation;
        Quaternion endRot = ComputeTargetRotation(flipped, isSideRotated, lastRotateLeft);

        float t = 0f;
        while (t < flipDuration)
        {
            transform.rotation = Quaternion.Slerp(startRot, endRot, t / flipDuration);
            t += Time.deltaTime;
            yield return null;
        }
        transform.rotation = endRot;
    }

    public void ToggleSideRotation(bool rotateLeft)
    {
        isSideRotated = !isSideRotated;
        lastRotateLeft = rotateLeft;
        StopAllCoroutines();
        StartCoroutine(SideRotate(isSideRotated, rotateLeft));
    }

    private System.Collections.IEnumerator SideRotate(bool sideRotated, bool rotateLeft)
    {
        Quaternion startRot = transform.rotation;
        Quaternion endRot = ComputeTargetRotation(isFlipped, sideRotated, rotateLeft);

        float t = 0f;
        while (t < sideRotationDuration)
        {
            transform.rotation = Quaternion.Slerp(startRot, endRot, t / sideRotationDuration);
            t += Time.deltaTime;
            yield return null;
        }
        transform.rotation = endRot;
    }

    private Quaternion ComputeTargetRotation(bool flipped, bool sideRotated, bool rotateLeft)
    {
        Quaternion rot = baseRotation;
        if (flipped) rot *= Quaternion.Euler(0f, 0f, 180f);

        if (sideRotated)
        {
            float angle = rotateLeft ? -sideRotationAngle : sideRotationAngle;
            rot *= Quaternion.Euler(0f, angle, 0f);
        }

        return rot;
    }
}
```

Figure 6.19: FollowPlayer script.

6.11 Health_gain

Health_gain is tied to the Health Recovery GameObject (see 2.1.2). When the Player collides with the GameObject, the script will check if the Players' remaining health is full or not. In the case that it's not, the Player will collect it and restore their health by an amount that can be edited in the Unity Inspector. If it's full, then the Player will pass through it and not be collected. This is to prevent accidentally collecting the health item when it wasn't needed to. On earlier versions, the recovery amount was set in the code, but by making it editable, it allows room for future development of other Healing items that heal for different amounts.

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class Health_gain : MonoBehaviour
{
    public float verticalSpeed;
    public float horizontalSpeed;
    public float ZaxisSpeed;
    public Image greenHealth;
    public GameObject onCollectEffect;
    public float AmountRecovered;
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        transform.Rotate(verticalSpeed, horizontalSpeed, ZaxisSpeed);
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            if (greenHealth != null && greenHealth.fillAmount != 1)
            {
                //greenHealth.fillAmount += 0.25f;
                greenHealth.fillAmount += AmountRecovered;
                greenHealth.fillAmount = Mathf.Clamp01(greenHealth.fillAmount);
                Instantiate(onCollectEffect, transform.position, transform.rotation);
                Destroy(gameObject);
            }
        }
    }
}
```

Figure 6.20: Health_gain script.

6.12 Speed_up

Speed_up is tied to the Speed-Up GameObject (see 2.1.3). When the Player collides with an Object containing this script, they will get a temporary speed boost. The speed value of which the Players base speed is upgraded to can be set via the Unity Inspector, as well as the duration of said speed boost. During this time, a bar will appear on the top right of the UI (see Figure 3.1) and slowly disappear over the course of a set duration. In this Project, I have set it so the speed boost lasts for 10 seconds and 1/10 of the Speed-Ups bar is drained every second, giving a clear visual on the Player of how much time they have left before it runs out. Once the time has passed, in our case 10 seconds, the Players speed will be reduced back to its normal speed value.

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class Speed_up : MonoBehaviour
{
    public float verticalSpeed;
    public float horizontalSpeed;
    public float ZaxisSpeed;
    public GameObject onCollectEffect;
    public Image SpeedDuration;
    public GameObject NotSpeedDuration;
    [Tooltip("The speed value to apply temporarily")]
    public float Speed_enh = 30f;
    [Tooltip("How long the boost lasts, in seconds")]
    public float BoostDurationInSeconds = 10f;

    void Update()
    {
        transform.Rotate(verticalSpeed, horizontalSpeed, ZaxisSpeed);
    }

    private void OnTriggerEnter(Collider other)
    {
        mover mover = other.GetComponentInParent<mover>();
        Instantiate(onCollectEffect, transform.position, transform.rotation);
        if (mover == null)
        {
            Debug.LogWarning("Speed_up: no mover found on " + other.name);
            return;
        }
        GetComponent<Collider>().enabled = false;
        if (GetComponent<Renderer>() != null)
            GetComponent<Renderer>().enabled = false;
        StartCoroutine(BoostSpeed(mover));
    }

    private IEnumerator BoostSpeed(mover mover)
    {
        float originalSpeed = mover.MoveSpeed;
        mover.MoveSpeed = Speed_enh;

        SpeedDuration.fillAmount = 1f;
        SpeedDuration.gameObject.SetActive(true);
        NotSpeedDuration.SetActive(false);

        float elapsed = 0f;
        while (elapsed < BoostDurationInSeconds)
        {
            yield return new WaitForSeconds(1f);
            elapsed += 1f;
            SpeedDuration.fillAmount -= 0.10f;
        }

        mover.MoveSpeed = originalSpeed;
        SpeedDuration.fillAmount = 1f;
        SpeedDuration.gameObject.SetActive(false);
        NotSpeedDuration.SetActive(true);

        Destroy(gameObject);
    }
}
```

Figure 6.21: Speed-up script.

7 Conclusions and Future Development

Considering my limited experience using Unity Engine in the past, I believe this Project has surpassed my original expectations in terms of features. It was thanks to my supervisor for encouraging me to keep making additions to the Video Game, even if many were outside the scope of my original plans, such as the pause menu, moving platforms, portals and keeping accountability for potential rare mishaps during gameplay that would create an unfair or even unplayable situation. The purpose of this Project was to create a 3D platformer that utilized gravity as its central mechanic as well as add a compelling dialogue system that makes the video game have more essence rather than being a hollow obstacle course. It should be noted that a Video Game can be considered good for a number of reasons whether it has an interesting story but lackluster gameplay, or smooth and responsive gameplay but little to no story. My project attempts to strike a balance between both story and gameplay.

It should be noted that this Project was made under a number of constraints and there are certain aspects I'd wish to improve or make new additions to the overall Project. The biggest improvement I wish to do is expand on the gravity mechanic. Currently there are only two gravitational fields, up and down. I wish to expand it so game objects would have their own gravitational fields or give the Player more choices when it comes to switching their own gravity. Another aspect to begin with is finding and acquiring new assets for the Video Game that would fit the theme and my vision of it. Additionally, I'd like to increase the variety of hazards to more than asteroids. Another aspect is the length of the Project as well as improve on the checkpoint system to be able to keep track of the player's progress even when closing the application. Finally, I'd like to experiment with adding a form of countdown and visual indicators before the suns would inevitably collide, causing a supernova in the process and forcing a game over.

8 Resources and References

-Articles

- Kamitsios, M., Papadimitriou, S., Chrysafiadi, K., & Virvou, M. (2018). Incorporating quizzes into a 3D educational game calling web services. In EdMedia + Innovate Learning 2018 (pp. 1170-1179). Association for the Advancement of Computing in Education (AACE).
- Kamitsios, M., Chrysafiadi, K., Virvou, M., & Sakkopoulos, E. (2018). A Stereotype User Model for an Educational Game: Overcome the Difficulties in Game Playing and Focus on the Educational Goal. 2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA). IEEE. <https://doi.org/10.1109/IISA.2018.8633655>
- Papadimitriou, S., Chrysafiadi, K., Kamitsios, M., & Virvou, M. (2018). FSP Creator: A Novel Web Service API Creator of Fuzzy Students Progress Profile. 2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA), 1-7. IEEE. <https://doi.org/10.1109/IISA.2018.8633662>
- Papadimitriou, S., Kamitsios, M., Chrysafiadi, K., & Virvou, M. (2021). Learn-and-play personalised reasoning from point-and-click to virtual reality mobile educational games. Intelligent Decision Technologies, 15(2), 321-332. <https://doi.org/10.3233/IDT-200204>
- Chrysafiadi, K., Kamitsios, M., & Virvou, M. (2023). Fuzzy-based dynamic difficulty adjustment of an educational 3D-game. Multimedia Tools and Applications, 82, 27525-27549. <https://doi.org/10.1007/s11042-023-14515-w>
- Chrysafiadi, K., Papadimitriou, S., Kamitsios, M., & Virvou, M. (2026; first online 2025). User Experience in Serious Games: Adaptivity in the Plot and Environment of the Game. In M. Virvou et al. (Eds.), Artificial Intelligence-Empowered Software Engineering 2024, Lecture Notes in Artificial Intelligence and Soft Computing, Vol. 52, pp. 215-230. Springer. https://doi.org/10.1007/978-3-031-98410-5_13

-Game Assets

- <https://assetstore.unity.com/packages/2d/textures-materials/sky/skybox-series-free-103633>
- <https://assetstore.unity.com/packages/3d/characters/picolabo-218332>
- <https://assetstore.unity.com/packages/vfx/particles/particle-collection-skj-2016-free-samples-72399>
- <https://assetstore.unity.com/packages/vfx/particles/casual-hit-318002>
- <https://assetstore.unity.com/packages/3d/props/ten-power-ups-217666>
- <https://assetstore.unity.com/packages/3d/environments/sci-fi/3d-asteroid-pack-263841>
- <https://assetstore.unity.com/packages/audio/music/vision-of-stars-293364>
- <https://assetstore.unity.com/packages/vfx/particles/pulsar-neutron-star-for-urp-309131>
- <https://assetstore.unity.com/packages/3d/animations/human-basic-motions-free-154271>
- <https://assetstore.unity.com/packages/3d/characters/stellarwitch-3d-309418>
- <https://assetstore.unity.com/packages/3d/environments/landscapes/raft-on-the-desert-free-low-poly-pack-141948>
- <https://assetstore.unity.com/packages/vfx/orbs-vfx-pack-254771>
- <https://assetstore.unity.com/packages/vfx/shaders/distortion-portal-shader-341124>
- <https://assetstore.unity.com/packages/vfx/particles/free-quick-effects-vol-1-304424>

-Other resources and inspirations

- <https://www.youtube.com/watch?v=U0165s9Fq9M>
- https://youtu.be/_nRzoTzeyxU
- <https://www.youtube.com/watch?v=DX7HyN7oJjE>
- <https://www.youtube.com/watch?v=MNUYe0PWNNs>
- Unity Learn <https://learn.unity.com/>
- Unity Asset Store <https://assetstore.unity.com/>