

Contents

Contents	1
Analysis.....	7
Background.....	7
Research	8
Outer Wilds.....	8
Gameplay Objectives	8
Celestial Body designs.....	10
The HUD.....	11
The Legend of Zelda: Breath of the Wild.....	12
How BotW makes exploration fun	12
The HUD.....	14
Far Cry 4.....	15
Atmosphere	15
Conclusion	16
Player Feedback.....	17
Interviews	17
Player 1's responses.....	17
Player 2's responses.....	18
Player 3's responses.....	19
Player 4's responses.....	20
Conclusion	21
Questionnaire	21
Questions	21
Responses	22
Problem Modelling	26
Planet Class Diagrams	26
Star System Generation	27
Gravity System	27
Star Mapping System	28
Tree Generation.....	28

Planet Terrain and Biome Generation	29
Fractal Brownian Motion	30
Marching Cubes	31
Hive tile Generation	32
A* Pathfinding:	33
Dijkstra's Algorithm.....	33
My own algorithm custom Pathfinding algorithm	34
Walk Animation Algorithm.....	34
My own algorithm.....	35
FABRIK algorithm	36
The rest of the walk cycle	37
Objectives	38
Project Limitations	40
Technical Limitations	40
Project Limitations	41
Design	42
Planet Terrain Generation	42
Planet Class Diagrams	42
Quad sphere mesh generation.....	43
Overview.....	43
Visuals.....	44
Data structures required.....	44
Pseudocode	45
Adding procedural noise to the mesh.....	45
Overview.....	45
Data structures required.....	46
Visuals.....	47
Pseudocode	50
Planet Biome Generation.....	52
Data structures required.....	52
Biome Generation.....	52
Overview.....	52
Visuals.....	52
Data structures required.....	53
Pseudocode	54
Planet Map Generation.....	56

Overview.....	56
Visuals.....	57
Data structures required.....	59
Pseudocode	59
Planet Atmosphere Generation	63
Writing the shader	63
Overview.....	63
Visuals.....	63
Pseudocode	64
Implementing the shader.....	67
Overview.....	67
Visuals.....	67
Data structures required.....	68
Pseudocode	68
Tree Generation.....	69
Overview / Visuals	69
Pseudocode	70
Gravity and Collision System.....	72
Overview	72
Data structures required.....	72
Pseudocode	72
Galaxy Generation	76
Star Generation.....	76
Overview	76
Pseudocode	77
Star Pathfinding	77
Overview / Pseudocode	77
Visuals.....	78
Data structures required.....	78
Hive Generation	79
Hive generation	79
Overview	79
Pseudocode	80
Data structures required.....	80
Pathfinding in the hive	80
Overview	80

Pseudocode	82
Tower generation	82
Overview / Visuals	82
Pseudocode	83
Ship Design	85
Visuals.....	85
Dialogue System	88
Overview.....	88
Visuals.....	89
Data structures required.....	89
Pseudocode	90
Technical solution	93
Android.cs	94
AndroidVision.cs	96
Atmosphere.shader	97
AudioHandler.cs.....	109
BlitMaterialFeature.cs.....	112
Branch.cs.....	116
CameraState.cs	119
CeilLight.cs	121
CloudFollower.cs.....	124
Clouds.shader	127
CloudSurround.cs.....	136
ColourGenerator.cs.....	140
CompassAligner.cs	143
ConstructionHandler.cs.....	146
ControlHelp.cs.....	148
CustomTile.cs	149
Dialogue.cs.....	151
Dijkstras.cs	154
DoorCollide.cs	156
DustFollower.cs.....	157
EffectsHandler.cs	159
ElevationData.cs	160
EvilBaseDoorOpen.cs	162
FabrikLeg.cs.....	164

Flat.cs	168
FlipSwitch.cs.....	172
Helipad.cs.....	177
HiveGen.cs	178
InteractDesktop.cs	184
Interface.cs	185
InventoryUI.cs.....	196
LetterEffect.cs	204
MeetingHandler.cs.....	207
MeshGen.cs	209
MiniMap.cs	216
Noise.cginc.....	222
NoiseShader.compute.....	226
NoiseStructs.cs.....	229
OpenDoor.cs	232
OreGen.cs	236
PauseMenu.cs	240
PhysicsUpdate.cs	243
Planet.cs.....	246
PlanetEffect.cs	251
PlanetGeneral.cs	257
PlanetLighting.cs	260
RaiseSeat.cs	268
RampOpen.cs	271
RoboEyes.cs	273
RobotWeight.cs.....	274
RoboVision.cs.....	277
SadGirl.cs	284
Save.cs	287
ScreenEffects.shader.....	288
ScreenEffectsHandler.cs	294
Segment.cs.....	295
Shelf.cs.....	304
ShipDoorOpen.cs	307
ShipMapAngle.cs.....	310
ShipRouter.cs	311

ShipWeight.cs	314
ShipWingOpen.cs	319
SittingRobot.cs	320
StarGenSystem.cs	321
SunGenSystem.cs	332
SurpriseAndroid.cs	335
Tetris.cs	338
TextRender.cs	354
Tile.cs	357
TowerGen.cs	361
TreeGen.cs	364
TriggerHiveGen.cs	369
UpdateStars.compute	370
Wall.cs	371
Weight.cs	374
ZeroWeight.cs	378
Testing	383
Tests	383
Changes following failed tests.	390
Evaluation	390
Evaluation of objectives	390
Objectives restated	390
Objective 1: Generate Procedural Planets	393
Objective 2: Generate the Hive (and Towers)	395
Objective 3: Command Line Interface	396
Objective 4: Dialogue System	396
Objective 5: Galaxy Generation	397
Objective 6: Player Movement	397
Objective 7: Star Mapping System	398
Objective 8: Gameplay, Menu, Options	398
Objective 9: Ship Functionality	399
Brief summary of objective evaluation	399
Feedback on solution	400
Responses to questions	400
Evaluation of responses	401
Future improvements of solution	401

Select screenshots of the solution	404
--	-----

Analysis

Background

My project:

I want to make a first-person exploration game in Unity. The game's title is Third Law, and the game objective is exploring a procedurally generated galaxy. This galaxy is composed of procedural star systems, which themselves are composed of procedural planets and the player can fly in their ship between. Along their journey, the player will gather resources, which can be used to upgrade their person and their ship, allowing for exploration that is more effective. The gameplay will follow some sort of short story, as I want the entire game to take around 5 hours to finish.

The coding required for this project consists of:

Creating algorithms to procedurally generate planets and stars. Creating shaders to make the celestial object look appealing and immersive, for example to give the planets atmosphere and the stars brightness. Using Newtonian physics to create a gravity engine: allowing planets to orbit stars. Spawning and procedurally generating foliage such as trees and ore on the planets. Creating UI such as generated map models for the planets, star systems and galaxies so the player knows where they are. Allowing the player to explore (manmade) inner structures of planets as well as buildings on its surface. Creating a systems that can perform analysis on planets, optimise paths the ship is taking, include a dialogue system to interact with the user.

Research

The background research for my project consisted of evaluating the effectiveness and use of techniques employed by similar games to my own, to see whether and how I want to implement them in my game. This effectively gave me a ‘jump start’ in designing my project, as I had a great sample of well-established and publicly approved design features at my disposal which I could use as foundations to innovate and build my project upon.

The games I chose for my research were –

- [Outer Wilds](#)
- [The Legend of Zelda: Breath of the Wild](#)
- [Far Cry 4](#)

Outer Wilds

The main features of Outer Wilds I want to research are the celestial body designs, the HUD, and the gameplay objectives. This is because these are design challenges I will also be facing in my project.

Gameplay Objectives

Outer Wilds is a first-person action-adventure exploration game, where the player’s goal (initially unknown) is to find the Eye of the universe. They achieve this by exploring a handcrafted solar system, along the way learning its secrets. For instance, the player learns how to exploit intentional game mechanics to travel to new places.



In the image above, the player has taken a photograph of the ‘Quantum Moon’ (top right). Through the game they learned they must photograph the moon to entangle its superposition with the ships so that it does not disappear when unobserved. Only by doing this can the player pass through the moon’s thick atmosphere and land on its surface.

Similarly, I would like learning knew, previously unknown, game mechanics to be a part of my game's progression. This consequently gives the player a lot of freedom in how they can act, as they could potentially accidentally discover the hidden mechanic before they are supposed to. I can counteract this to some extent by increasing the number of ways the player can interact with the environment. Combining these methods in unique and interesting ways can then create difficult to discover until intended techniques, each technique acting a progression stage in the game.

Good examples of this are in the game [Celeste](#), where certain advanced movement options are only taught to the player (such as the "wave-dash") late into the game.

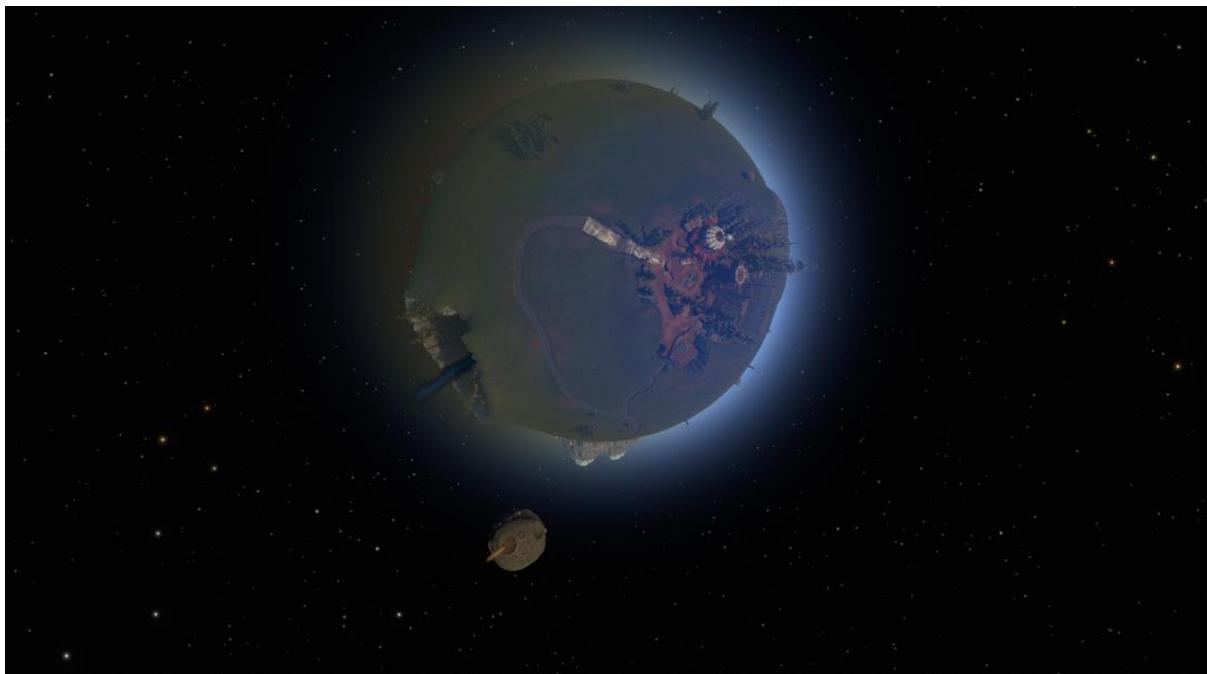
As an objective therefore, I want to implement combinable movement and interaction abilities for the player, so that an element of the progression comes in combining said abilities in logical, interesting ways.



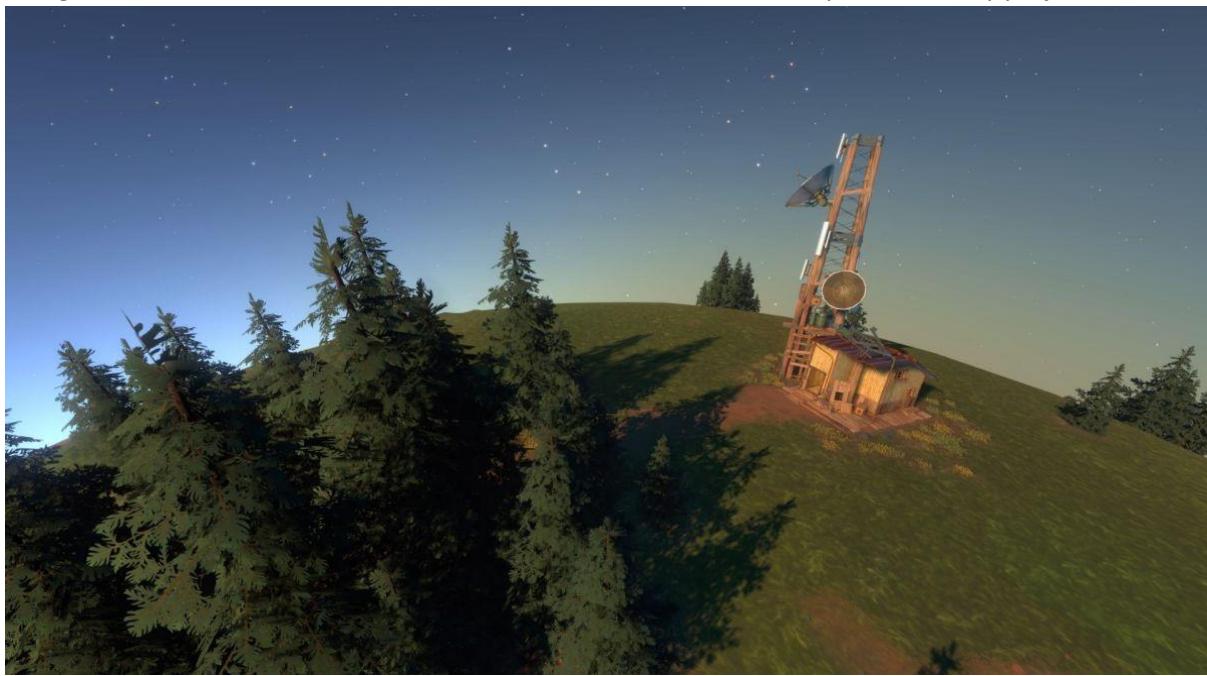
One of the main mechanics of Outer Wilds is that after 22 minutes, the sun explodes, forcing the player to restart their exploration, but with knowledge retained from their previous expeditions. This 'death-cycle' gameplay core is used in this game for primarily the same reason it is more famously used in [The Legend of Zelda: Majora's Mask](#), to save effort on content by making it recyclable. In both games this technique is used very effectively as it allows the game developers to focus on the quality of the content over the quantity, often leading to a better gameplay experience. It also allows the player to feel more intimate and engaged with the world, as they become very familiar with the solar system traversing it throughout their 'death-cycles'.

However, the obvious consequence of this is that the rate at which the pilot in Outer Wilds explores the star system and the rate at which Link explores Termina is slow, and so the progression of both the games is heavily focused on puzzle solving to achieve such exploration.

For my game, I want to focus more attention on the exploration, meaning the player needs more content to explore. However, I still want to maintain a high level of quality on the places the player explores. For this, I intend to use procedural generation.

Celestial Body designs

As seen above, the planet 'Timber Hearth' is brought to life with features such as a thin atmosphere, foliage such as trees, and a cratered moon, all of which I intend to implement in my project.

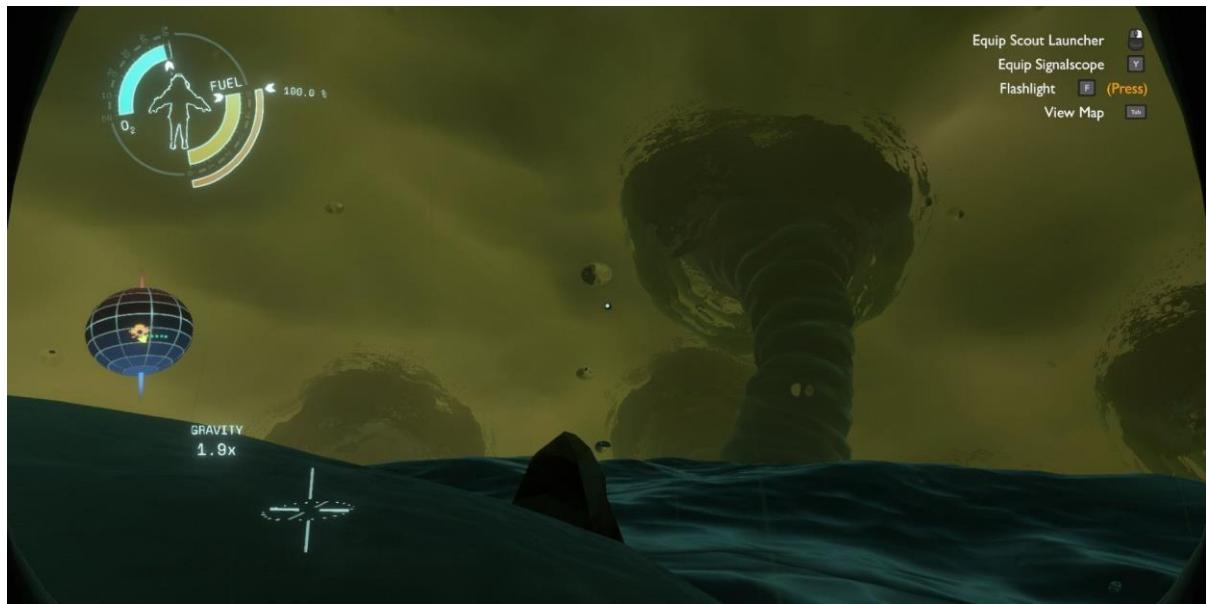


As the player approaches the planet, the LOD (level of detail) on the planet's surface increases smoothly, to optimise performance when the player is not close enough to the planet. For example, smaller meshes such as small trees and satellite dishes in the image above are only rendered when the player is within a close-enough proximity to them. If I want my game to perform at a high frame rate, I will also need to implement rendering boundaries as a feature of my game.

Something else I also like is the splitting up of 'Timber Hearth' into regions such as the main village, mountains, and woods. This makes exploring the planet more engaging as there is lots of variety in the planet composition. Another inclusion in my game should therefore be biomes, which could

have different foliage and colour pallets, depending perhaps in part on the climate where the biome is located. Biomes would also therefore have different weather cycles.

One thing the planets in Outer Wilds (in my opinion) are lacking is wildlife. Inclusion would further engage the player in the game as it would make the world feel more alive (if not literally making it more alive!). However, for me, implementing wildlife would be difficult from a visual perspective, as I lack the experience. I think a good alternative would be procedurally generated, robotic wildlife as the procedural generation takes care of the artistic variety, and robots lend themselves well to simple visual design.



The image above was taken from the surface of “Giant’s Deep”, a gas giant with a thick atmosphere, ocean, and tornadoes. I intend to implement these in my game, the thick atmosphere as type of atmosphere (the types being: thick, thin and none – each of which has a customisable colour pallet), the ocean as a type itself (the types being: swimmable, acidic and dry (just an ocean floor)).

The HUD

The HUD, shown in the image above, for the player controller in Outer Wilds is concise, intuitive, and informative.

The oblate spheroid (middle left) indicates the poles of the planet and the player’s relative position on the planet to them, as well as the ship’s position and the path the player has taken. This is an excellent solution to providing a map for the player. As the game is about exploration of planets, it does not give any of the actual composition of the planet away, and in doing so forces the player to create their own map of the planet. However, it still helps orient the player as much as a map should. Since my game is about exploring galaxies, I would like automated planet mapping to be a feature of my game, perhaps I would have a map like the above for when the player is exploring the planet, but a more detailed one when they are in the ship, so they know roughly what the planet has to offer in terms of resources.

There is an indication of the gravity scale (middle left) and the player’s orientation relative the planet normal at their position (lower left). Perhaps information such as this could be included as optional upgrades to the player’s suit, as these features in my experience while useful in Outer Wilds are non-

essential, and so implementing them as upgrades makes them more exciting for the player, and perhaps better appreciated.

The fuel, oxygen and health meters are displayed in the top left. These are the most important features of the HUD in Outer Wilds, and so are its largest element. As I am intending to have a lot of variety in my planets, I will need appealing, noticeable, yet not obstructive stats as part of my HUD to inform the player of the current temperature, radiation level, upcoming and current weather, current gravity magnitude, as well as their health, fuel, and oxygen capacity.

In the top right is listed all the keys the player can press that are not related to movement. This is useful as it reminds players what they can do and how to do it. If I am to add additional equipment this is one approach to informing the player of how to use it. Another would be an inventory system where the user can select a piece of equipment to equip, and then press the interact button (E) to use it.

The Legend of Zelda: Breath of the Wild

The Legend of Zelda: Breath of the Wild (BotW) is widely regarded as one of the, if not the, most successful game in terms of exploration. As the focus of my game is exploration, it would be wise of me to research how BotW achieves engaging exploration, as often in games exploration can feel like an annoying game of hide and seek.

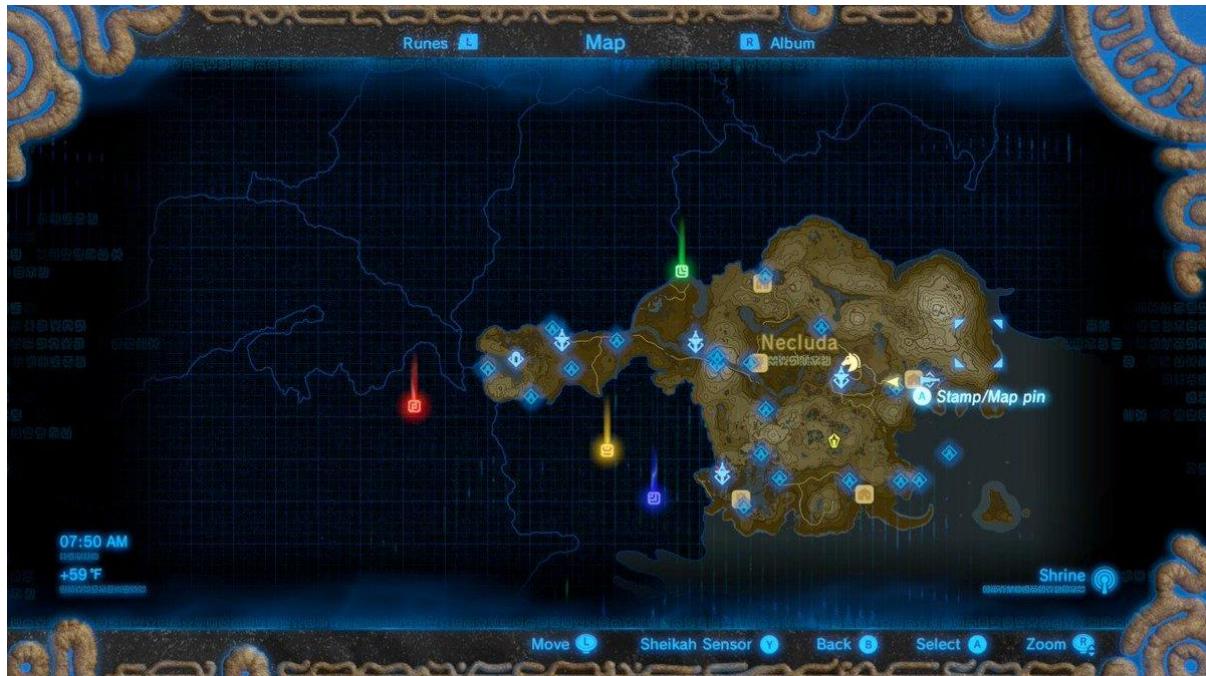
How BotW makes exploration fun



To make exploration easier, BotW includes a 'Sheikah Sensor' (see image above). This alerts the player to any surrounding resources they wish to gather, meaning it is almost impossible for the player to miss any desired resources they pass by. By aiding the player in this way, the 'Sheikah Sensor' removes a large part of the frustration that can be associated with resource gathering, because it dramatically increases the chances that the player will find the resource they were looking for.

The other part of the frustration of resource gathering is its repetitive nature. Although this could be seen as a challenge for the player, personally I don't like it and so will not be requiring high demands

of resources to be gathered by the players in my game. Instead, like in BotW, any resource gathering I include will be focused on collecting a *variety* of items, instead of a large quantity of similar items. With a variety of resources comes a variety of different challenges the player must overcome to gather them, so long as these challenges feel unique enough and don't become tedious, I believe this is the best approach to resource gathering.



By far, BotW mapping system is my favourite from any game. The game requires you to visit 'Sheikah Towers' scattered across the map to unlock the part of the map local to the tower visited. The map is clear enough for you to spot interesting locations you may want to visit, while not giving everything away. The fact when you first enter a new area, you have no map to guide you, really adds to the engagement the player has with their environment, as they must look themselves for the path ahead, and only once they are familiar enough with the area and so can find and scale the tower are they rewarded with the map. This will be a feature of my game; however, it will take a very different form than in BotW due to the very different nature of my game. The 'local map' unlocked in BotW will be the coordinates of local star systems to the player's location in my game. However, for each star system the orbiting planet's maps will be automatically filled in by the ship based on what the ship has seen of the planet. This is because the player will be exploring so many planets, it seems unreasonable and unnecessary for the player to have to unlock a map for each.



BotW allows the player to mark locations they spot in 'Hyrule' on their map, and to see long distance with the 'Sheikah Scope'. Together, this is a perfect way for the game developers to reinforce the emphasis on exploration and freedom in this game, as the 'Scope' essentially allows the player to create their own objectives and let their curiosity lead them through 'Hyrule'. In the image above, the player has marked the location of a 'Shrine' on their map, which they are presumably headed towards. In a similar way, I will include a set of binoculars (or something of the like) which the player can use to mark neighbouring star systems, or to flag the presence of previously unidentified celestial objects to the ship.

The HUD



BotW's HUD is very informative while not being obstructive to the player's view. In the bottom right, the local map, time, weather forecast, noise made by the player, temperature and 'Sheikah Senor Gauge' are all displayed. A lot of information neatly and stylishly compressed into a small area of the screen. For my game, I want to keep the player informed about several otherwise hidden variables, such as temperature, radiation level, forecast as well as the player's remaining oxygen, fuel and health. To do this while keeping the screen clean have a few options. Like BotW's HUB in the screenshot above, I could neatly compress my gauges into a corner of the screen.

Alternatively, I could retain the immersion of the game by including the HUD as a feature of the player's spacesuit, like is done in Outer Wilds and [Metroid Prime](#).

Far Cry 4

Note that I have not played this game, so I am researching how it appears to function – and drawing inspiration from that. My focus when researching Far Cry 4 was systemic design, and how it is implemented effectively in the game to create unique experiences for each player, and subsequently make them feel more immersed in and part of the world.

Atmosphere



The world of Far Cry 4 seems alive, partially due to the systemic design, partially due to the inclusion of particle effects such as dust, smoke, and local fog; each respectively displayed in the 3 pictures above I have included of the game. This is by no means unique to Far Cry 4, although it is implemented effectively in this game, and it is non-obvious until recognised how much it contributes to the atmosphere of the game. Without these particle effects, the world would seem disconnected from the events that take place in it. And so, I intend to include particle effects such as dust, smoke, and local fog in the game I am creating.

Conclusion

From the games I have researched, I have been inspired by techniques used to enhance the player's immersion and experience. Specifically, I have been inspired to implement features in my game listed below:

Features inspired by Outer Wilds:

- Including a variety of game mechanics which can be used together to overcome obstacles.
- A focus on exploration and procedural generation.
- Rendering boundaries for performance optimisation.
- Biomes with different foliage & colour pallets depending on climate.
- Procedurally generated wildlife.
- Atmosphere types, ocean types, weather types
- Informative HUD and mapping systems.
- Inventory / key bindings for: equipment

Features inspired by The Legend of Zelda: Breath of the Wild:

- A 'Sheikah Sensor' / Geiger counter like resource indicator.
- Variety of resources to gather.
- Automatically generated planet maps and unlockable star charts.
- A 'Sheikah Scope' / binoculars for marking or identifying features worth visiting.
- A spacesuit HUD with gauges and meters which are compact yet informative & intuitive.

Features inspired by Far Cry 4 (and other open-world games):

- Particle effects for atmosphere and immersion.

Player Feedback

Interviews

To get an idea of the direction I wanted my game to go in, I interviewed four potential players of my game to see what they thought about certain ideas I had regarding its development, and any ideas they had regarding its development too. These players were selected from my target audience, people familiar with first person 3D games who are interested in exploration focused games.

Player 1's responses

Have you played any games like the one I propose to design? If so, what did you like about them, and what didn't you like about them?

I have played one game like this, No Man's Sky. The game had a good idea of being able to buy ships from NPCs as they docked at stations or on planets. However, the game had a very slow and repetitive exploration cycle of just collecting fuel, jumping to a new star, and collecting fuel again.

What makes a game engaging and worth playing?

I think a game is most engaging when it constantly keeps the player having to have a minimal focus on the game, for example, having NPCs periodically spawn to attack the player or some kind of event changing how the player must interact with the environment.

What level of difficulty in exploration and resource gathering would you like in the game I am creating?

I think the exploration should be an easier part of the game, making sure that the player cannot die at first due to not experiencing something; this would turn players away from the game, as it ruins the momentum of the game. However, I think the resource gathering should be challenging, with some kind of added difficulty when collecting rarer resources. This would allow a player who is proficient in the game's mechanics obtain rarer, more useful resources to aid their progress. I also think that storage of resources could be a potential balancing issue for the game, as players might not have enough space to store the number of resources required to progress.

Do you like clear objectives in games, or should the objective of games be ambiguous or non-existent (like in Minecraft for example)?

I think that games which are open world should have objectives which are clear to the player, but they don't restrict the player's ability to freely explore and use their own initiative to do things (i.e., the player is not explicitly told to gather a resource, rather, that the resource is part of a crafting recipe for (x) item that the player must use to advance the game. This lets players explore the game while still having objectives to complete.

What features would you like to see added to my game? (I.e., things to explore, NPCs, side quests, skill trees – systems other games have implemented in addition to the core gameplay)

I think a good feature to add for late-game players would be an AI controlled ship, which can be set to mine a certain resource. This ship would be able to mine resources while the player is away, however it would not defend itself, meaning the player either must be on the lookout while the AI ship mines harder-to-get resources, or the ship only mines low-level resources to keep it safe. Automation features would mean the player can focus on exploration and other tasks.

Is it better to focus development on the scale (quantity) of the game, the quality of what you explore, or some sort of compromise in the middle?

I think that the focus of a game should be quality of content because many games have had a large-scale map; however, their lack of polished features made them difficult to use. Quality should come first, scale later.

Player 2's responses

Have you played any games like the one I propose to design? If so, what did you like about them, and what didn't you like about them?

I have played one game like this, No Man's Sky. The game had a good idea of being able to buy ships from NPCs as they docked at stations or on planets. However, the game had a very slow and repetitive exploration cycle of just collecting fuel, jumping to a new star, and collecting fuel again.

What makes a game engaging and worth playing?

I think a game is most engaging when it constantly keeps the player having to have a minimal focus on the game, for example, having NPCs periodically spawn to attack the player or some kind of event changing how the player must interact with the environment.

What level of difficulty in exploration and resource gathering would you like in the game I am creating?

I think the exploration should be an easier part of the game, making sure that the player cannot die at first due to not experiencing something; this would turn players away from the game, as it ruins the momentum of the game. However, I think the resource gathering should be challenging, with some kind of added difficulty when collecting rarer resources. This would allow a player who is proficient in the game's mechanics obtain rarer, more useful resources to aid their progress. I also think that storage of resources could be a potential balancing issue for the game, as players might not have enough space to store the number of resources required to progress.

Do you like clear objectives in games, or should the objective of games be ambiguous or non-existent (like in Minecraft for example)?

I think that games which are open world should have objectives which are clear to the player, but they don't restrict the player's ability to freely explore and use their own initiative to do things (i.e. the player is not explicitly told to gather a resource, rather, that the resource is part of a crafting recipe for (x) item that the player must use to advance the game. This lets players explore the game while still having objectives to complete.

What features would you like to see added to my game? (I.e. things to explore, NPCs, side quests, skill trees – systems other games have implemented in addition to the core gameplay)

I think a good feature to add for late-game players would be an AI controlled ship, which can be set to mine a certain resource. This ship would be able to mine resources while the player is away, however it would not defend itself, meaning the player either must be on the lookout while the AI ship mines harder-to-get resources, or the ship only mines low-level resources to keep it safe. Automation features would mean the player can focus on exploration and other tasks.

Is it better to focus development on the scale (quantity) of the game, the quality of what you explore, or some sort of compromise in the middle?

I think that the focus of a game should be quality of content because many games have had a large-scale map; however, their lack of polished features made them difficult to use. Quality should come first, scale later.

Player 3's responses

Have you played any games like the one I propose to design? If so, what did you like about them, and what didn't you like about them?

I have played one game like this, No Man's Sky. The game had a good idea of being able to buy ships from NPCs as they docked at stations or on planets. However, the game had a very slow and repetitive exploration cycle of just collecting fuel, jumping to a new star, and collecting fuel again.

What makes a game engaging and worth playing?

I think a game is most engaging when it constantly keeps the player having to have a minimal focus on the game, for example, having NPCs periodically spawn to attack the player or some kind of event changing how the player must interact with the environment.

What level of difficulty in exploration and resource gathering would you like in the game I am creating?

I think the exploration should be an easier part of the game, making sure that the player cannot die at first due to not experiencing something; this would turn players away from the game, as it ruins the momentum of the game. However, I think the resource gathering should be challenging, with some kind of added difficulty when collecting rarer resources. This would allow a player who is proficient in the game's mechanics obtain rarer, more useful resources to aid their progress. I also think that storage of resources could be a potential balancing issue for the game, as players might not have enough space to store the number of resources required to progress.

Do you like clear objectives in games, or should the objective of games be ambiguous or non-existent (like in Minecraft for example)?

I think that games which are open world should have objectives which are clear to the player, but they don't restrict the player's ability to freely explore and use their own initiative to do things (i.e. the player is not explicitly told to gather a resource, rather, that the resource is part of a crafting recipe for (x) item that the player must use to advance the game. This lets players explore the game while still having objectives to complete.

What features would you like to see added to my game? (I.e. things to explore, NPCs, side quests, skill trees – systems other games have implemented in addition to the core gameplay)

I think a good feature to add for late-game players would be an AI controlled ship, which can be set to mine a certain resource. This ship would be able to mine resources while the player is away, however it would not defend itself, meaning the player either must be on the lookout while the AI ship mines harder-to-get resources, or the ship only mines low-level resources to keep it safe. Automation features would mean the player can focus on exploration and other tasks.

Is it better to focus development on the scale (quantity) of the game, the quality of what you explore, or some sort of compromise in the middle?

I think that the focus of a game should be quality of content because many games have had a large-scale map; however, their lack of polished features made them difficult to use. Quality should come first, scale later.

Player 4's responses

Have you played any games like the one I propose to design? If so, what did you like about them, and what did you not like about them?

I have not played any similar games myself, although I am familiar with the game No Man's Sky which sounds very similar to your game. One thing I like about No Man's Sky is its vast galaxy and procedurally generated planet system. It always means you have something interesting and unique to explore.

What makes a game engaging and worth playing?

One thing that some games have that makes them worthwhile is procedural generation. This is because it makes the game interesting and unique as it always changing, and you never get the same experience twice. What also makes a game engaging is the level of immersion you feel while playing it; this could be through an interesting/relatable story or through a high level of graphical detail that simulates real life.

What level of difficulty in exploration and resource gathering would you like in the game I am creating?

I think the game must be at least moderately difficult so that there is an actual reward to playing. It probably shouldn't be too difficult as the scale of the game is very large and therefore, making it more difficult makes it feel like more of a grind which can become boring after a while. However, it should still take some skill to progress.

Do you like clear objectives in games, or should the objective of games be ambiguous or non-existent (like in Minecraft for example)?

I think a blend between the two makes for a more interesting game. For example, in GTA V, you can play the story mode missions, complete side quests, or you can just run around and do your own thing. While there are clear objectives, there is also an element of exploration and freedom involved. Even Minecraft has the "end goal" of beating the Ender Dragon but this is just an optional achievement.

What features would you like to see added to my game? (I.e., things to explore, NPCs, side quests, skill trees – systems other games have implemented in addition to the core gameplay)

I think a planet creation system would be an interesting idea. For example, the player could manually generate their own planet with their own specifications and then they own that

planet and can do what they want with it. You could also allow the user to import their own height map data to form the terrain of this planet as well as letting the user choose the environment, atmosphere, shape, weather, etc.

Is it better to focus development on the scale (quantity) of the game, the quality of what you explore, or some sort of compromise in the middle?

Since what we explore is procedurally generated, I think it would be better to have more things to explore as you never know what you're going to get. A terribly generated planet could be quite fun and as long as there are more places to go with actual value, I wouldn't mind.

Conclusion

Comparing the player's responses to each question, I found that collectively they think:

- I should focus on the quality of my game whilst keeping in mind its scale is important.
- My game should have clear objectives to motivate the player, but the player should also be allowed to tackle these in their own time.
- My game's difficulty should focus on its most exciting part (Player 1 thinks this should be resource gathering not exploration, whilst Player 3 thinks the opposite. Player 2 thinks the difficulty should focus on the difficult encounters and preparation. Player 4 thinks it should focus on the player's skill).
- My game can be engaging by including a variety of gameplay. As well as good graphics and an engaging story.
- I should primarily focus on fun, unique, interesting, and extensive exploration – including a variety of gameplay and interactions between NPCs,
- I should go out of my way to ensure this exploration doesn't become repetitive, boring, or meaningless.

From these responses I have decided additionally to:

- Prioritise quality, engaging & extensive exploration,
- Focus the difficulty of the game around its exploration (as this is what I want to be its most exciting part).
- Create an engaging story to go alongside the gameplay.
- Include upgrades to make resource gathering more effective (as suggested by Player 1).
- Include upgrades to allow for exploration of extreme locations (as suggested by Player 2).
- Include settlements of NPCs to interact with (as suggested by Player 3).

Questionnaire

While my interviews with players were valuable as the feedback was constructive, there are certain questions I would like to ask a larger sample of people, so that I get feedback more accurate to the average player's view.

Questions

The questionnaire I constructed is below:

- Q1: How important is story when making an immersive game (1-5)?
[\[SEE RESPONSE\]](#)
- Q2: How important is challenging difficulty when making an engaging game (1-5)?

[\[SEE RESPONSE\]](#)

- Q3: How important should combat be to make an exploration game difficult (1-5)?
[\[SEE RESPONSE\]](#)
- Q4: How important should puzzles be to make an exploration game difficult (1-5)?
[\[SEE RESPONSE\]](#)
- Q5: How important is the quantity of content when making a memorable game (1-5)?
[\[SEE RESPONSE\]](#)
- Q6: How important is player customisability when creating a memorable game (1-5)?
[\[SEE RESPONSE\]](#)
- Q7: How important are player upgrades to a game's progression (1-5)?
[\[SEE RESPONSE\]](#)
- Q8: How important is player skill to a game's progression (1-5)?
[\[SEE RESPONSE\]](#)
- Q9: Which console would you play my game on? [\[SEE RESPONSE\]](#)

Responses

Question Number	Response bar chart	Analysis	Response to analysis
-----------------	--------------------	----------	----------------------

Q1	<table border="1"> <thead> <tr> <th>Response</th> <th>Percentage</th> </tr> </thead> <tbody> <tr><td>1</td><td>14%</td></tr> <tr><td>2</td><td>36%</td></tr> <tr><td>3</td><td>35%</td></tr> <tr><td>4</td><td>13%</td></tr> <tr><td>5</td><td>1%</td></tr> </tbody> </table>	Response	Percentage	1	14%	2	36%	3	35%	4	13%	5	1%	<p>The mean response was 2.6, meaning the common view is that story in games can be immersive, but that an immersive game isn't just about story.</p>	<p>The results are as I predicted, so I will continue to focus on gameplay with story as a feature but not a focus of my game.</p>
Response	Percentage														
1	14%														
2	36%														
3	35%														
4	13%														
5	1%														
Q2	<table border="1"> <thead> <tr> <th>Response</th> <th>Percentage</th> </tr> </thead> <tbody> <tr><td>1</td><td>17%</td></tr> <tr><td>2</td><td>7%</td></tr> <tr><td>3</td><td>40%</td></tr> <tr><td>4</td><td>11%</td></tr> <tr><td>5</td><td>25%</td></tr> </tbody> </table>	Response	Percentage	1	17%	2	7%	3	40%	4	11%	5	25%	<p>The mean response was 3.2, but there was a high standard deviation, meaning the responses were very mixed regarding the importance of difficulty in engaging games.</p>	<p>I think the results show that people often enjoy playing easy and hard games. I will do my best to satisfy the majority of the responses, by including difficult optional challenges.</p>
Response	Percentage														
1	17%														
2	7%														
3	40%														
4	11%														
5	25%														
Q3	<table border="1"> <thead> <tr> <th>Response</th> <th>Percentage</th> </tr> </thead> <tbody> <tr><td>1</td><td>2%</td></tr> <tr><td>2</td><td>65%</td></tr> <tr><td>3</td><td>30%</td></tr> <tr><td>4</td><td>2%</td></tr> <tr><td>5</td><td>0%</td></tr> </tbody> </table>	Response	Percentage	1	2%	2	65%	3	30%	4	2%	5	0%	<p>The mean response was 2.1, meaning the common view is that difficult combat is not necessary to make an exploration difficult.</p>	<p>I think the results show people think there are many other equally valid ways of making a game difficult, which I agree with.</p>
Response	Percentage														
1	2%														
2	65%														
3	30%														
4	2%														
5	0%														
Q4	<table border="1"> <thead> <tr> <th>Response</th> <th>Percentage</th> </tr> </thead> <tbody> <tr><td>1</td><td>14%</td></tr> <tr><td>2</td><td>36%</td></tr> <tr><td>3</td><td>35%</td></tr> <tr><td>4</td><td>13%</td></tr> <tr><td>5</td><td>1%</td></tr> </tbody> </table>	Response	Percentage	1	14%	2	36%	3	35%	4	13%	5	1%	<p>The mean response was 2.5, showing that people also view puzzles as not fundamental to an exploration games difficulty.</p>	<p>Q4 has a higher mean than Q3, this means people think puzzles are slightly more important than combat to an exploration games difficulty, so I will focus more on difficult puzzles than difficult combat.</p>
Response	Percentage														
1	14%														
2	36%														
3	35%														
4	13%														
5	1%														

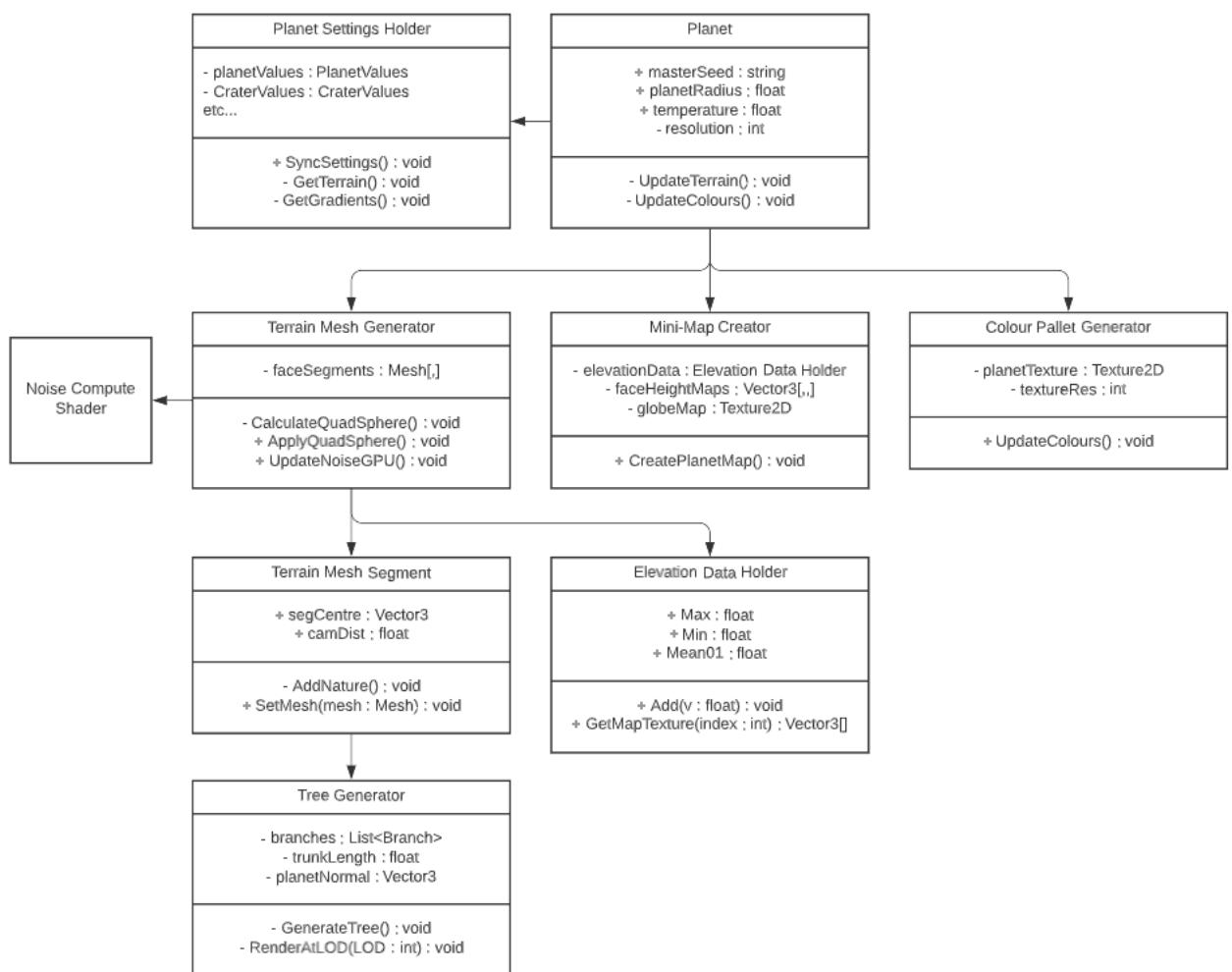
Q5	<table border="1"> <thead> <tr> <th>Response</th> <th>Percentage</th> </tr> </thead> <tbody> <tr><td>1</td><td>24</td></tr> <tr><td>2</td><td>3</td></tr> <tr><td>3</td><td>5</td></tr> <tr><td>4</td><td>27</td></tr> <tr><td>5</td><td>41</td></tr> </tbody> </table>	Response	Percentage	1	24	2	3	3	5	4	27	5	41	<p>The responses indicate two groups of people, those who find a short game memorable and those who find a long game memorable</p>	<p>The analysis makes sense, as short games can be more impactful, but long games can become part of our lives. The small scale of my project therefore suggests I should focus on a short(ish) but impactful game.</p>
Response	Percentage														
1	24														
2	3														
3	5														
4	27														
5	41														
Q6	<table border="1"> <thead> <tr> <th>Response</th> <th>Percentage</th> </tr> </thead> <tbody> <tr><td>1</td><td>24</td></tr> <tr><td>2</td><td>3</td></tr> <tr><td>3</td><td>5</td></tr> <tr><td>4</td><td>27</td></tr> <tr><td>5</td><td>41</td></tr> </tbody> </table>	Response	Percentage	1	24	2	3	3	5	4	27	5	41	<p>The mean response was 3.6. Like Q5, the responses indicate that while some people value customisability, others do not.</p>	<p>Unlike Q5, this means I should include customisability in my game, but that extensive customisation is not necessary.</p>
Response	Percentage														
1	24														
2	3														
3	5														
4	27														
5	41														
Q7	<table border="1"> <thead> <tr> <th>Response</th> <th>Percentage</th> </tr> </thead> <tbody> <tr><td>1</td><td>52</td></tr> <tr><td>2</td><td>31</td></tr> <tr><td>3</td><td>14</td></tr> <tr><td>4</td><td>4</td></tr> <tr><td>5</td><td>0</td></tr> </tbody> </table>	Response	Percentage	1	52	2	31	3	14	4	4	5	0	<p>The mean response was 1.9, this shows that people don't think player upgrades are essential for a game's progression.</p>	<p>I found the mean to be surprisingly low (although I would have responded lower). I think this is explained by Q8.</p>
Response	Percentage														
1	52														
2	31														
3	14														
4	4														
5	0														

Q8	<table border="1"> <thead> <tr> <th>Responses</th> <th>Percentage</th> </tr> </thead> <tbody> <tr><td>1</td><td>2%</td></tr> <tr><td>2</td><td>8%</td></tr> <tr><td>3</td><td>17%</td></tr> <tr><td>4</td><td>29%</td></tr> <tr><td>5</td><td>43%</td></tr> </tbody> </table>	Responses	Percentage	1	2%	2	8%	3	17%	4	29%	5	43%	<p>The mean response was 3.9, this shows that people think player skill is essential for a game's progression.</p>	<p>As Q7 and Q8 show, people much prefer to advance their own skill than their player's "level". Although I will still include upgrades in my game, I will make sure it is the player's skill that gives them access to such.</p>
Responses	Percentage														
1	2%														
2	8%														
3	17%														
4	29%														
5	43%														
Q9	<table border="1"> <thead> <tr> <th>Platform</th> <th>Percentage</th> </tr> </thead> <tbody> <tr><td>PC</td><td>63.4%</td></tr> <tr><td>Switch</td><td>16.9%</td></tr> <tr><td>Xbox</td><td>8.5%</td></tr> <tr><td>PlayStation</td><td>11.3%</td></tr> <tr><td>Mobile</td><td>0.0%</td></tr> </tbody> </table>	Platform	Percentage	PC	63.4%	Switch	16.9%	Xbox	8.5%	PlayStation	11.3%	Mobile	0.0%	<p>The vast majority want to play my game on PC, followed by Switch, PlayStation, Xbox, and thankfully nobody wants to play on mobile.</p>	<p>As expected, nobody wants to play my game on mobile. Seeing as most people want my game to be on PC, I will focus on creating my game for PC.</p>
Platform	Percentage														
PC	63.4%														
Switch	16.9%														
Xbox	8.5%														
PlayStation	11.3%														
Mobile	0.0%														

Problem Modelling

Planet Class Diagrams

Each planet has many aspects and for each I will create a class. Hence, my class diagram shows their connections and the most important variables stored within them. Many classes not mentioned here for example are disconnected from all others and use the `UnityEngine.Start` function exclusively to complete their goals. Other classes are used to store variables or static functions.



Class Name	Objective
Planet	To display debug settings for the planet and to contain and organise all sub-classes used to generate the planet.
Planet Settings Holder	To store variables fundamental to the Planet's generation.
Terrain Mesh Generator	Generate the quad sphere mesh in segments and generate the layered noise functions to give the planet terrain.

Terrain Mesh Segment	To handle the generation of objects such as trees that should only appear when the player is close.
Tree Generator	Generate and display the tree at the appropriate level of detail.
Elevation Data Holder	Store the altitude of each vertex of the planet, as well as meta data such as the maximum, minimum and mean altitude.
Mini-Map Creator	To gather the planets elevation data to generate a 3D heightmap of the planet and convert it to a 2D planet map.
Colour Pallet Generator	Generate the planet's terrain and atmosphere colour constrained by the temperature of the planet. Create multiple biomes each with its own, similar, colour pallet. Atmosphere colour pallet is 2D, one dimension interpolates between sunset colour and mid-day colour.

Star System Generation



It is worth noting that each star system is assigned a unique cube in space (like a grid reference) and the star system is then generated at a random position within this cube. This ensures star systems are not generated too close and allows me to easily control the size of the galaxy.

Gravity System

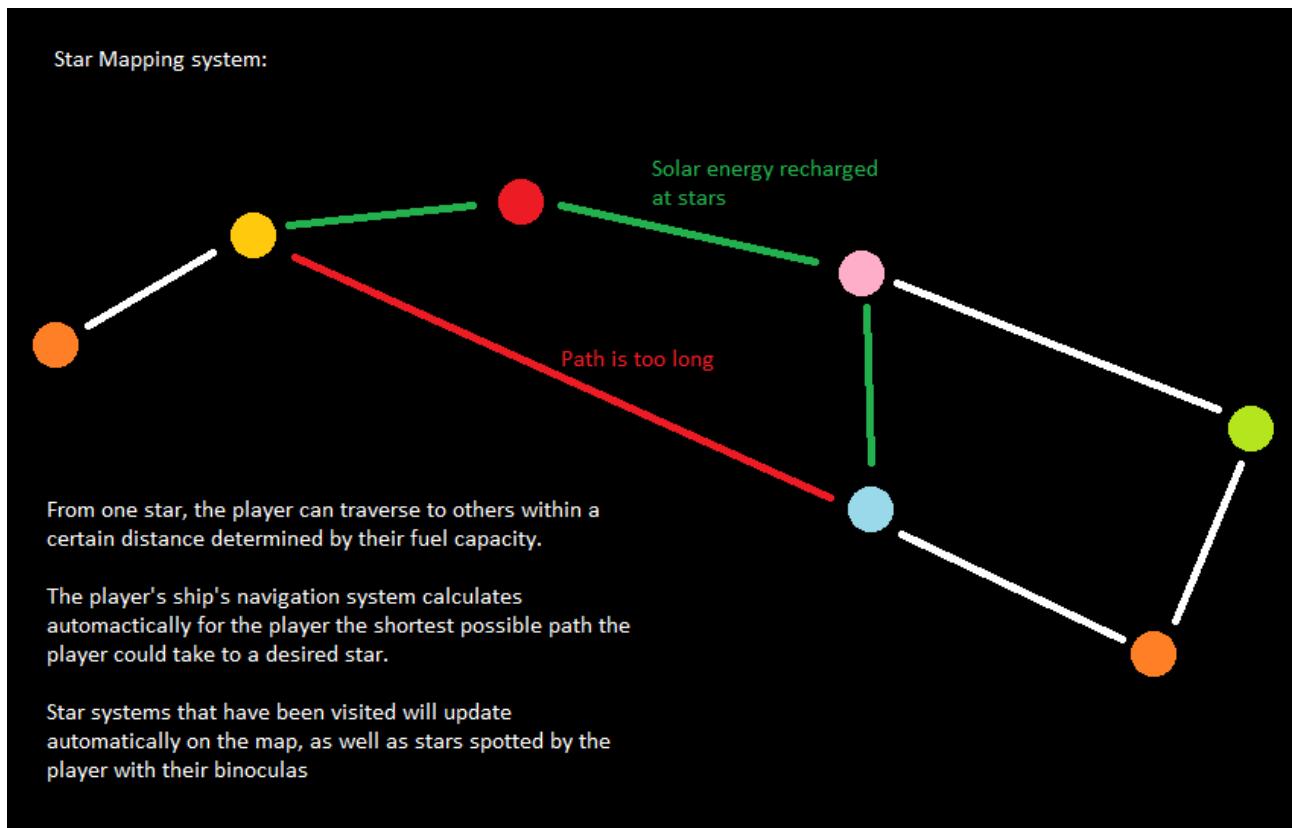
Each celestial body has a mass that determines how it interacts with other bodies in a star system, each is given a velocity vector which has a pre-set initial state dependant on their pre-set starting position relative to the sun. I pre-set these variables to ensure the star system does not immediately

turn chaotic as soon as the player arrives at the system. Following Newton's law of universal gravitation but with an inverse law (and not an inverse square law), after each frame of the game the acceleration vector of the k^{th} celestial body a_k is calculated with the equation:

$$a_k = G \sum_{n=1}^N \left(\frac{m_n}{r_{n,k}} \right) (\vec{r}_{kn}) \{n \neq k\}$$

Where N is the number of celestial bodies, m_n is the mass of the n^{th} celestial body, $r_{n,k}$ is the distance between the n^{th} and k^{th} celestial bodies, \vec{r}_{kn} is the normalized vector from the k^{th} to the n^{th} celestial body, and G is a tuneable constant. Each acceleration vector is then divided by the FPS and added to the velocity vector which is then divided by the FPS and added to the position vector of the celestial body to produce the position the celestial body will appear at in the next frame. Should planets collide, they will be repelled with sufficient force to prevent any imminent future reencounters.

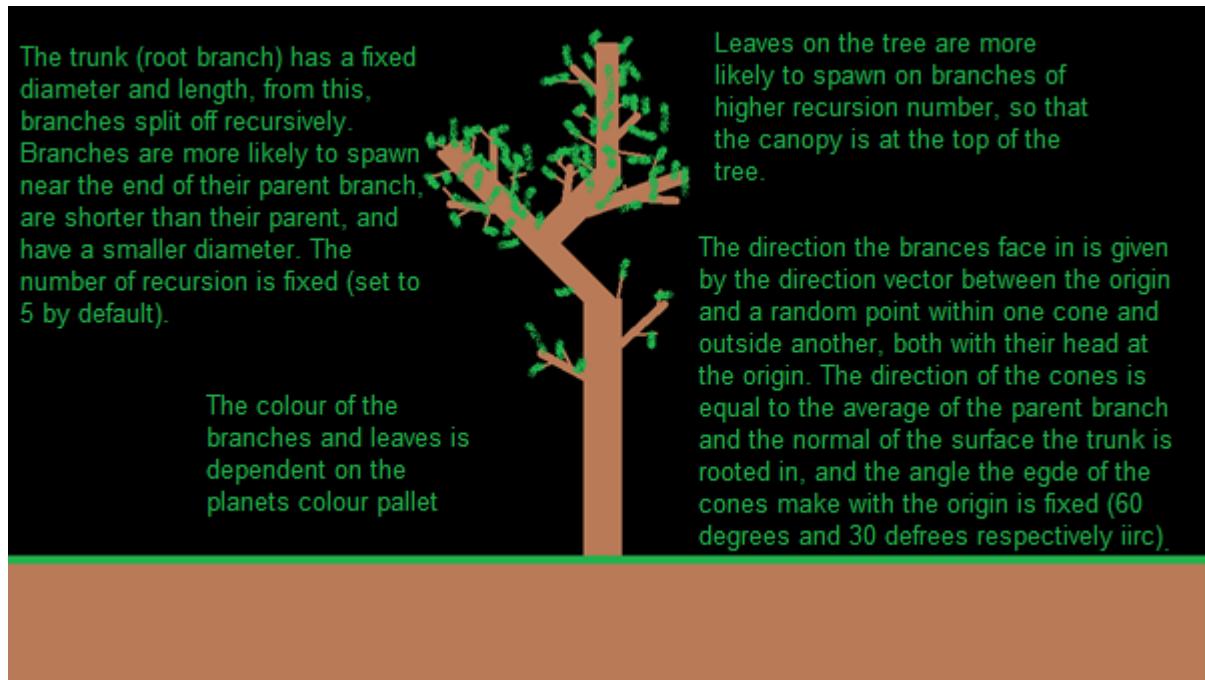
Star Mapping System



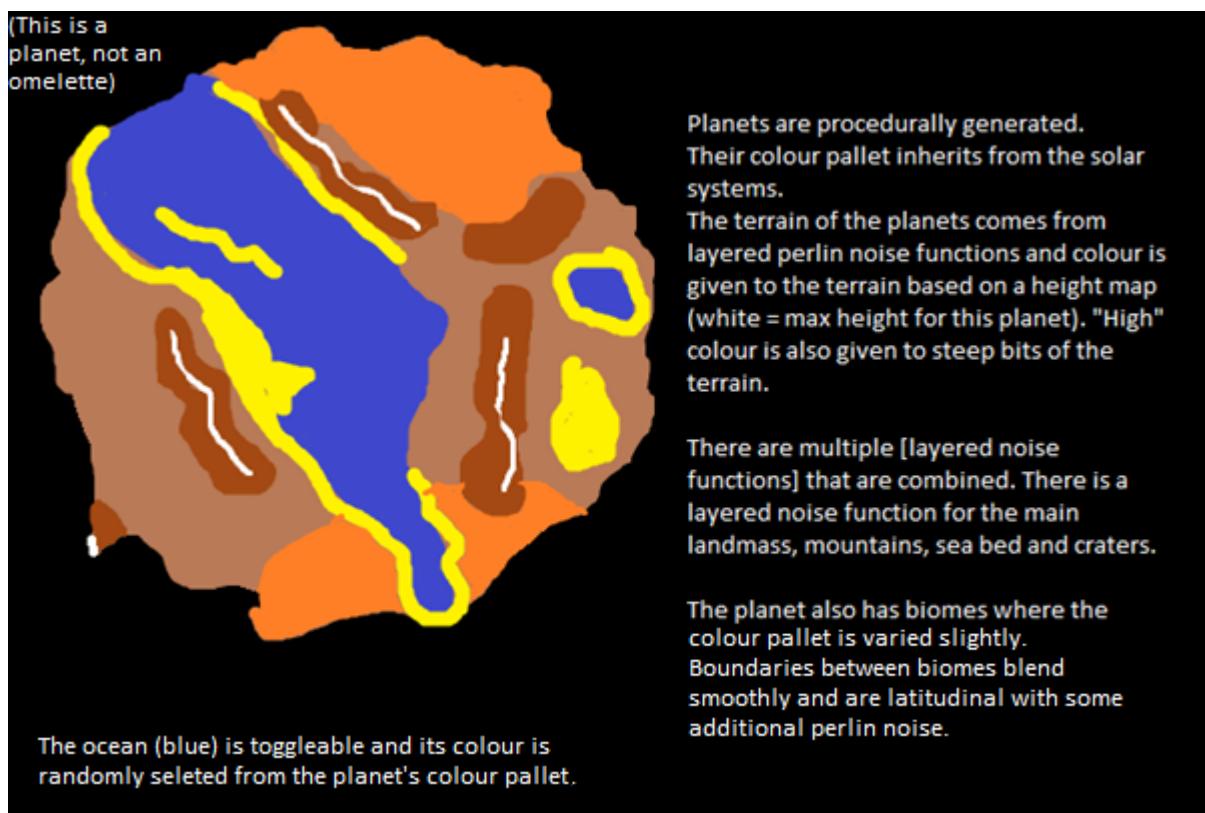
For the start mapping system, I have chosen to use Dijkstra's Algorithm. This is because the star map is simply a weighted graph, and the connected sections of the graph are small, so performance is not an issue. My alternative would be A* pathfinding, but this algorithm is really overkill for graphs with few arcs.

Tree Generation

I chose to implement my own algorithm for generating unique trees.



Planet Terrain and Biome Generation



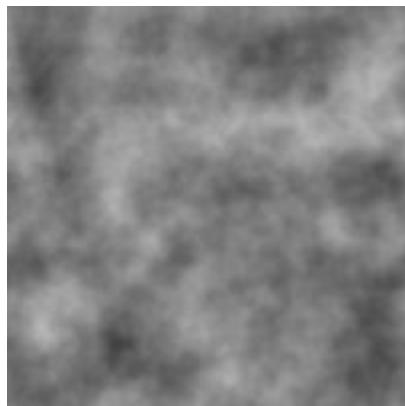
Algorithms for generating planet terrain:

Procedurally generating the planet's terrain is the most important aspect of my project. This is because the player will be spending most of their time exploring the planets. It therefore makes sense to research popular algorithms for generating procedural terrain and use my findings to implement unique landscapes into each of my planets.

Taking a step back, one approach I could have taken to generating planet's is handcrafting a selection and randomly picking which one the player will next explore. This is advantageous as it gives me the most control over what my planet's will look like. However, it is infeasible for me to create an entire galaxy full of original stars by hand and for this reason I want to procedurally generate my galaxy.

Two popular terrain generation techniques I researched are:

Fractal Brownian Motion

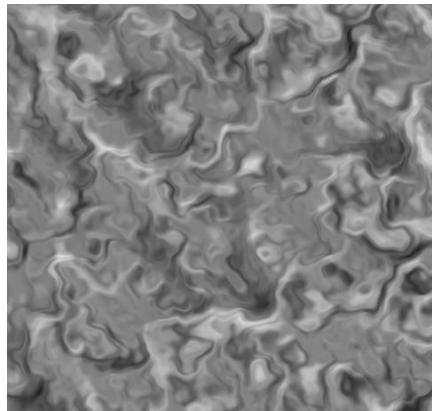


Fractal Brownian Motion is a technique to produce tuneable, detailed fractal noise functions. It works by layering Perlin noise functions in octaves, a higher octave layer typically has a higher noise frequency and smaller noise strength, so that higher frequency noise slightly modifies lower frequency noise. Usually, only 3 octaves of noise are sufficient, this is preferable as more octaves mean my planet's terrain will take longer to generate.

Using the Fractal Brownian Motion algorithm, I could sample 3d fractal noise from each vertex position on the quad sphere planet mesh, then translate these vertices away from the planet centre according to their respective sampled noise. Let this be a function of the mesh. I could have different terrain functions to warp the planet's surface in different ways, for instance, I could translate the vertices by only a little to distinguish land from ocean (land vertices translated out from the planet, ocean vertices in) and translate vertices by a lot in ridges to generate mountain ranges on the planet (if $N(x)$ is the noise sample given vertex x a ridged noise function is defined as $R(x) = |1-N(x)|$).

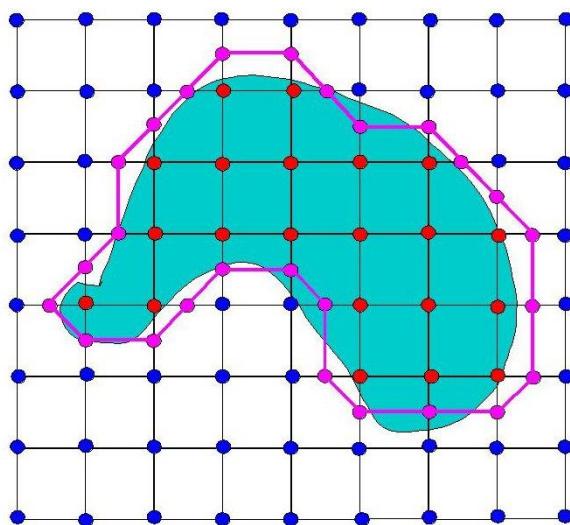
In the image above, $R(x)$ regions are red, green terrain has been translated out from the planet, blue terrain has been translated in.

Additionally, I could employ a technique called domain warping to offset the terrain function's noise function by another noise function, to give warped terrain like in the image below. This would make the terrain more challenging and interesting for the player to traverse.



Marching Cubes

This technique is advantageous compared to Fractal Brownian motion, because it allows for terrain to be generated above generated terrain, such as caves ceilings or overhanging cliffs. The technique works by assigning a Boolean value to each coordinate within a cube around the planet; adjacent True values are connected by triangles so that the final mesh acts as a boundary between all True and False values. If all values (and only these values) within a radius of the planet were True the resulting mesh would be a sphere, for instance.



However, to produce the appropriate connection between all adjacent pairs of coordinates, a significant amount of compute time is required. Because I want planets to be generated on-the-fly with high resolution, this is a significant drawback to the method for my project, to the extent that I intend to implement Fractal Brownian Motion in my project and not Marching Cubes.

Hive tile Generation



Starting at the start tile a new tile is instantiated adjacent to the previous tile. It is more likely to be in the direction the previous tile was instantiated in, so the corridors are straight. This continues n times. Then a new start tile is randomly chosen from the instantiated tiles, from which another "corridor" (now blue in this example) is developed.

Each tile has 6 walls one for each of the positive and negative directions of the x, y & z axis. Let us say the front wall is the positive z wall and the back wall is the negative z wall. If a new tile is instantiated in front of the previous, then the back wall of the new tile and the front wall of the previous tile are removed so that the player can walk from one to the other.

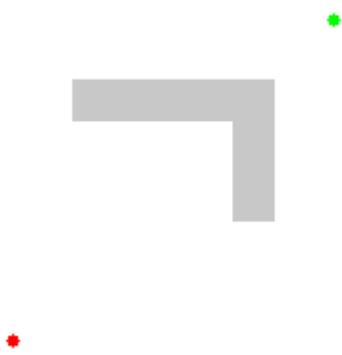
After the corridors are made, all walls between two instantiated tiles are removed (in our example this makes the end of the grey corridor form a loop).

Finally, I have included custom tiles such as stairs or environment specific rooms that are sometimes instantiated instead of tiles when making the corridor, to make the scene more interesting. These tiles take priority over regular tiles, and remove any within their bounding box so that there is no overlap of tiles.

Algorithms for navigating the Hive:

Additionally, I want a robot to be able to traverse between two points on the path so that it can either follow the player or “patrol” its local vicinity. This will occur rarely while playing the game (most “Hives” as I am calling them will be deserted as most planets will be deserted). To achieve this, I will need to implement a pathfinding algorithm.

After doing research I have concluded that I have three options:

A* Pathfinding:

This algorithm would work quite suitably for my program because it uses spatial reasoning to find the shortest path and in the “Hive” each tile is linked spatially to its adjacent tiles. Moreover, it is a reasonably quick algorithm to complete, which is advantageous for my game because I would need to run the algorithm every frame to continuously update the path from the robot to the player (since the player can move).

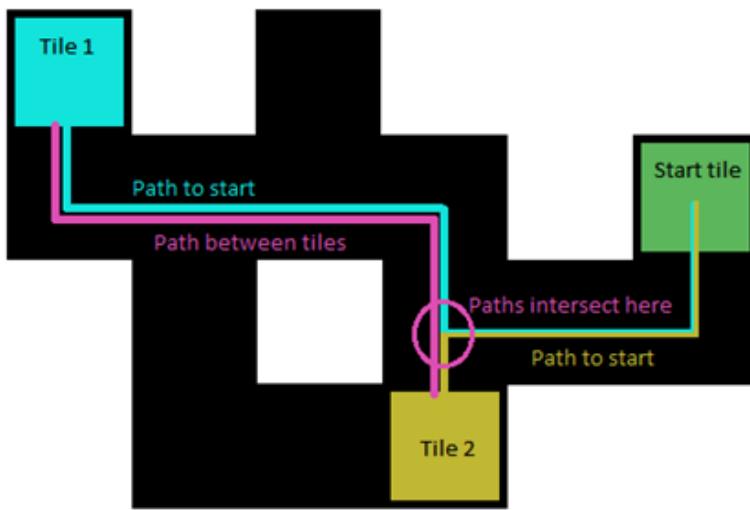
However, the way my tiles are joined together means that finding all adjacent tiles to one tile would require me to check the positions of every tile generated in the “Hive”. This would slow down the algorithm as this search would have to be run before the pathfinding algorithm. Moreover, the special tiles take priority over regular tiles when being instantiated which means certain tiles may be disconnected from the main body of the hive. This would mean the pathfinding algorithm would not be able to succeed in this circumstance and this would cause the algorithm and thus the game to run very slowly (if for instance the robot was disconnected from the player) as the algorithm would try to make a route through every tile, making the spatial reasoning aspect of the algorithm redundant.

Dijkstra's Algorithm

Like the A* pathfinding algorithm it requires the generation of an adjacency matrix, which for the entire “Hive” is slow to complete. It does not implement spatial reasoning to improve performance, so it will consider many more tiles/paths and be relatively slow to run. It is for this reason that the A* pathfinding algorithm is preferable to Dijkstra's algorithm in this circumstance as in my “Hive” a large, spatially connected graph, the A* algorithm will outperform Dijkstra's.

My own algorithm custom Pathfinding algorithm

Alternatively, I could take advantage of the parent-child relationship between the tiles (each tile [child] is instantiated adjacent to another tile [parent]) that is a by-product of the way I generated the “Hive.” This means that for any tile in the “Hive” if I get the parent of this tile, and the parent of this parent and so on, I will always get to the first tile generated (in the image the Start tile). This means I can get a path from any tile to the Start tile. If I want to get a path between any two tiles, I can get the paths from the two tiles to the Start tile and find the point closest to the tiles along those paths at which they intersect then link the paths together like in the image below:



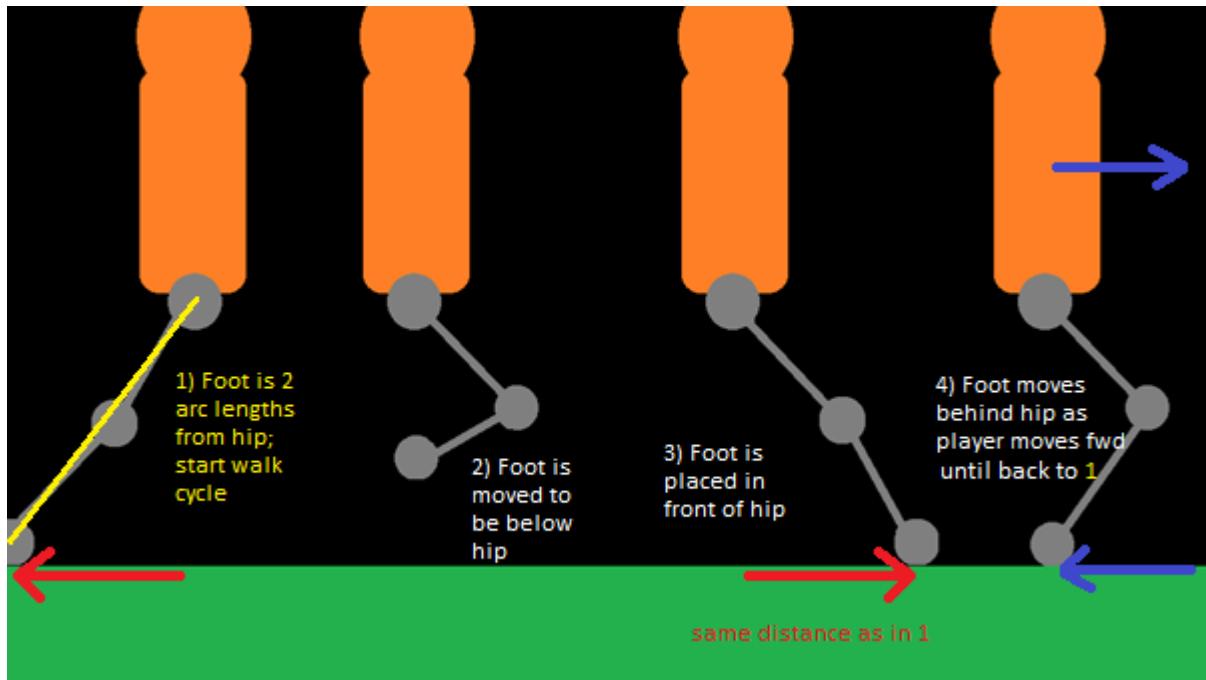
This algorithm is significantly faster than the A* algorithm, as it does not need to “try” anything, it just comes up with a path. If the algorithm fails because the two tiles are disconnected, it will still take the same time to finish the algorithm and without throwing an error.

However, one limitation of this simpler approach is that it does not guarantee that the path generated will be the shortest path between the two points. To help amend this I will perform Raycasts from the robot to points further along the path the robot is taking to see if the robot can directly traverse either to the player or a point further along the path. This means the robot will not walk in a zig zag shape (for instance) to get diagonally across an open room but will instead walk straight across. Moreover, if the robot cannot see the player the player cannot see the robot, which means if the robot isn’t taking the most obvious path to the player the player won’t realise, and when the player can see the robot, the robot will walk directly towards the player. For these reasons finding the shortest path isn’t a vital aspect to the pathfinding algorithm I want to implement.

Overall, this algorithm for me is preferable to the A* path finding algorithm because it is much simpler to implement, faster to run and will always give valid, sensible pathing between two tiles such that the player’s experience of the robot is that it is intelligent (as robots should be).

Walk Animation Algorithm

WALK CYCLE SEQUENCE (1):



Algorithms for the robot's walk cycle:

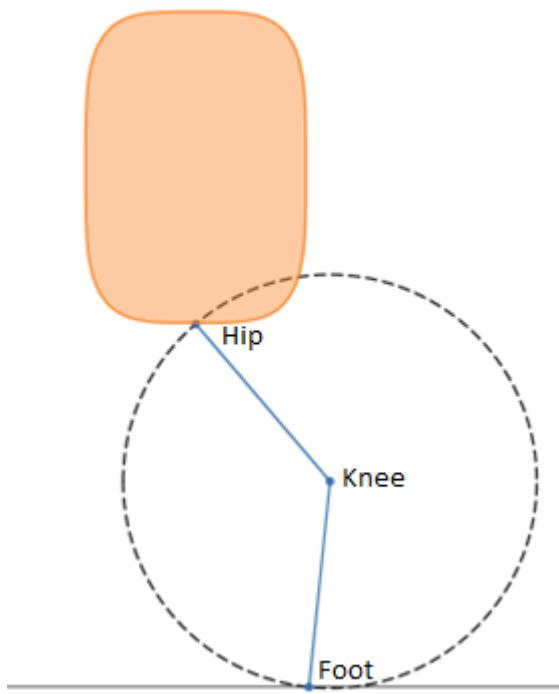
The most common approach for creating walk cycles in games is to animate them either using software such as Blender or hiring actors and using motion capture. As I have only minimal experience with Blender, any animations I make would feel rigid. Alternatively, I could use walking animation other people have made. Unfortunately, many of these come at a price, making this approach not within the scope of my project. Moreover, I don't really want to use animations in general because in my game the player and robots will be walking over terrain of varying gradient and movement over a variety of terrain is difficult to convey with animation.

Instead, I want to implement an algorithm that simulates a walk cycle. That is: when a foot is behind the robot it steps naturally in front fixing it onto the terrain at that point. Additionally, the two leg's walk cycles at any given point should be in approximate antiphase.

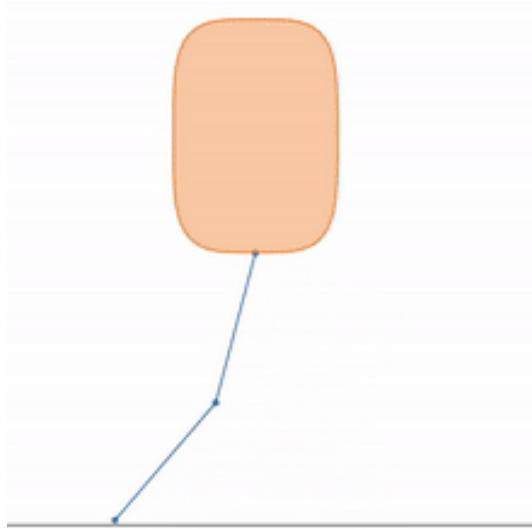
There are a couple algorithms that I could use to manage limb and joint positions for my walk cycle:

My own algorithm

Using what we know: the position of the hip, position of the foot and the limb length we can form an equation of a circle such that the hip and foot are on the edge.



This allows us to find the position of the knee for any position of the foot as it is at the centre of the circle, which means we can move the foot according to the WALK CYCLE SEQUENCE (1) and everything else will follow through:



[The above is an approximation of a walk cycle sequence created using desmos, linked [here](#)]

This algorithm is simple and cheap in 2D but is more complex in 3D, this is because in 3D there is a circle of viable knee positions. Another disadvantage is that the algorithm breaks if the foot is too far away from the hip. I could potentially resolve this issue with a fail state, but the next algorithm I researched doesn't have this issue.

FABRIK algorithm

This inverse kinematic algorithm is just as simple in 2D as 3D and is much more general purpose as well, for instance I could reuse this algorithm to create a creature with more than three joints.

Speaking abstractly, it works by iteratively adjusting the joints to transition from the previous foot position to the next foot position while keeping each limb length fixed:



[The above is a demonstration that I created in Unity of the FABRIK algorithm]

I much prefer this algorithm for the reasons listed above and so intend to use this one in my project.

The rest of the walk cycle

To make the walk cycle feel natural I need to make sure each step raises the foot off the ground and in an arc, then find an appropriate position for the feet to step to. When one foot is in the middle of their “arc” I want the other foot to snap to the ground directly below the player’s hip.

Objectives

1. Generate Procedural Planets
 - 1.1. Generate Planet Terrain
 - 1.1.1. Generate a Quad Sphere mesh.
 - 1.1.2. Add layered noise functions onto the Quad Sphere mesh for terrain. Different layers of noise functions for different terrain types: Continent noise, mountain noise, crater noise, seabed noise. Offset continent noise with additional noise for more variety. Run this algorithm on the GPU for faster execution.
 - 1.2. Generate Planet Colours
 - 1.2.1. Generate the planet's colour pallet, randomly based on the temperature of the planet.
 - 1.2.2. Generate a gradient of colours corresponding to terrain height, such that higher terrain gets "colder" colours. Blend a different colour into steep terrain.
 - 1.2.3. Add biomes with latitudinal boundaries, offset with noise that have different yet similar "gradients of colours" (1.2.2)
 - 1.3. Add Ocean and Atmosphere
 - 1.3.1. Ocean & atmosphere colour inherits from planet's colour pallet.
 - 1.3.2. The ocean shader should give the spherical ocean visual waves, by oscillating each sphere vertex's local scale.
 - 1.3.3. The atmosphere shader should give the atmosphere an optical density or "thickness", thinner atmospheres are more affected by the sun (stronger warm colour interpolations for sunset and sunrise). Atmosphere facing away from the sun is black.
 - 1.3.4. The cloud shader is a variant of the atmosphere shader, it has a thick atmosphere layer but inside is a hollow sphere, with cloudy noise covering its interior.
 - 1.3.5. Atmospheres can have colour bands; specific latitudes can be assigned different colours like for (1.2.3).
 - 1.3.6. Entrance and exit into and out of the regions of all these shaders should be seamless.
 - 1.3.7. Create a custom render pass with URP to render the atmospheres and oceans to the screen according to their distance from the player.
 - 1.4. Mini map generation
 - 1.4.1. Use terrain data from the planet's mesh to produce an equirectangular projection of the planet's terrain. Colour this map using the biome colours (1.2.3).
 - 1.4.2. The player should be able to view this map on their ship on request.
 - 1.5. Tree generation (as described by my Tree Generation Algorithm)
 - 1.6. Ore generation
 - 1.6.1. Zero to three pieces of ore are generated together, they can be collected to restore the player's fuel and increase their metal count.
 - 1.7. Each planet segment generates a random number of trees and ore within a given range.
 - 1.8. Foliage and cloud followers
 - 1.8.1. Grass and stones should be generated efficiently within proximity to the player.
 - 1.8.2. Clouds should make the planet seem 'alive', that is, a dynamic environment.
2. Generate the Hive (and Towers)
 - 2.1. Hive ("dungeon" or tile map) generation
 - 2.1.1. Randomly generate a "dungeon" (Hive) composed of corridors, stairs and prefabricated rooms following my Hive tile Generation algorithm.

- 2.1.2. Decorate the Hive with prefabricated props (boxes, papers, planks etc.)
 - 2.1.2.1. Add randomly generated shelves (they have 1 or 2 layers, 0 to 3 dividers, that naturally hold either an interactable computer or random prefabricated props)
- 2.1.3. Insert doors in corridors such that the door can only be seen from in front or behind.
 - 2.1.3.1. Doors should have an opening animation, the reverse for closing and stop their motion if the player is in the way.
- 2.1.4. Implement my own algorithm custom Pathfinding algorithm that allows NPCs to traverse between positions in the Hive, such as itself and the player, as described in the Problem modelling section.
- 2.2. Tower generation
 - 2.2.1. Build a tower in any direction at a position, each floor of the tower is prefabricated, floors should seamlessly connect. Windows are randomly added to walls on the exterior of the tower.
 - 2.2.2. Each floor in the tower is either made of a 3x3 or 5x5 tile grid. Ground floors have entrances, the top floor if it's a 5x5 this is either another 3x3 tower, a spire, or an air traffic control style top floor, where the player's ship can land and enter the tower from through a lever-controlled system, if it's a 3x3 this is a flat roof or a garden roof with a tree on top from my tree generation algorithm.
 - 2.2.3. For performance purposes, floors should only be instantiated when the player is close enough, a placeholder block should be in their place until this.
- 3. Make and render a CLI, like command prompt.
 - 3.1. The player can press E to start their interaction with the computer.
 - 3.2. Add a system to convert a string to pixels on a render texture, the opening text prompt should be: "ENTER *HELP* TO GIVE A LIST OF COMMANDS"
 - 3.3. Commands are stored in a dictionary and if the player types and enters a valid command, said command is executed. One command, GENERATE_MAP, is only available on the ship and displays a mini map of the planet (1.4); another, OPEN TETRIS.EXE, starts a game of Tetris.
- 4. Dialogue system
 - 4.1. Save progress of a series of conversations between Eva (on the ship) and the player. Conversations should be available every 5 minutes and once per life of the robot.
 - 4.2. Each sentence should be composed of 3 phrases each assigned a mood which affects how the phrase is rendered. Letters oscillate according to sine waves of a frequency dependant on the mood of the phrase they are in. This adds personality to the dialogue system.
 - 4.3. Sentences should transition in and out of view aesthetically by fading up at a rate according to an interpolation function $f(t) = t^n$ where $0 < t < 1$ and n is a tuneable parameter.
 - 4.4. Conversations should be stored and read from json files, annotated with each phrase's mood. Mood properties are also read from json files. This allows for streamlined creation of conversations by me.
- 5. Galaxy generation (described in the Problem modelling section)
 - 5.1. For performance purposes, all stars except the closest to the player are represented as 2D sprites, they should transition seamlessly to 3D models when the player is close.
 - 5.2. Planets should orbit the sun using equations of motion.
- 6. Player movement
 - 6.1. The player should be able to move in 3 dimensions.
 - 6.2. Players legs animate movement by implementing the FABRIK algorithm (described in the Problem modelling section)

- 6.3. Mouse movement left and right controls yaw rotation, up and down controls pitch, pitch is to be limited by tuneable parameters, if the player is on a planet. If they are not grounded, the player's body will rotate pitch, not just their head and this rotation will not be limited, so the player could do a backflip for example.
- 6.4. The player should be able to control the ship
- 6.4.1. The ship has toggleable wings that change whether left right mouse movement controls yaw or roll rotation. These wings may be useful on planet's where the ship's roll is adjusted so that the ship aligns with the planet's normal at the ships position.
7. Star mapping system
- 7.1. Each visited star will become visible on a star map the player can view. The player can select a coordinate in this star space and Dijkstra's algorithm will produce a path with refuelling points as nodes to get there from the player's current coordinate. This path then becomes part of the ship's HUD during flight.
8. Gameplay cycle, menu, options
- 8.1. Produce a functional start menu with options to exit, start a new game or continue.
- 8.1.1. The game saves the star map, dialogue state, inventory, and the player's star space coordinate. The game saves each time the player dies or enters a new coordinate in star space. Upon death, the player respawns at a random star space coordinate losing their inventory but retaining their dialogue state and star map.
- 8.2. Oxygen and fuel deplete over time when the player is not in their ship.
- 8.2.1. Oxygen is replenished in the ship and hive.
- 8.2.2. When the player runs out of oxygen, their fuel burns faster. When the player runs out of fuel, metal is consumed. When the player runs out of metal and fuel, they die.
- 8.3. Upgrades to the players statistics (such as fuel burn rate) can be acquired by talking to a constructor robot while possessing 100 metal to spend on it.
- 8.4. Pause menu has options to start a new game, self-destruct (force start of death sequence), view controls and quit the game.
- 8.5. Controls viewed should be specific to the actions the player can currently take.
9. Ship functionality
- 9.1. Ship legs deploy onto terrain to give the appearance they are supporting the ship.
- 9.2. Ship widget meters
- 9.2.1. Compass widget, aid fully for navigation, points to the north pole of the closest planet
- 9.2.2. Ship thermometer displays the calculated temperature of the ship.
- 9.2.3. Ship fuel depletes over time if the engine is on.
- 9.2.3.1. Ship fuel depletes faster if boost mode enabled.
- 9.2.3.2. Ship fuel replenishes via solar panels.
- 9.2.4. Velocity meter displays velocity and made-up RPM of ship.
- 9.3. Ship widgets and lights dim when engine off.

Project Limitations

Technical Limitations

- Players I have interviewed wanted me to add online features to this game such as a co-op mode. However, this would require that I pay for a server to host my players which is not within the scope of my project. These players accepted my valid point but requested that I implement an online mode in a later build of the game.

- Hiring a graphic designer for my game's aesthetic would have been a good idea to improve my player's engagement with the game but this too was costly for me as well. One player suggested this that I interviewed, but they agreed it was infeasible for the scope of my project.

Project Limitations

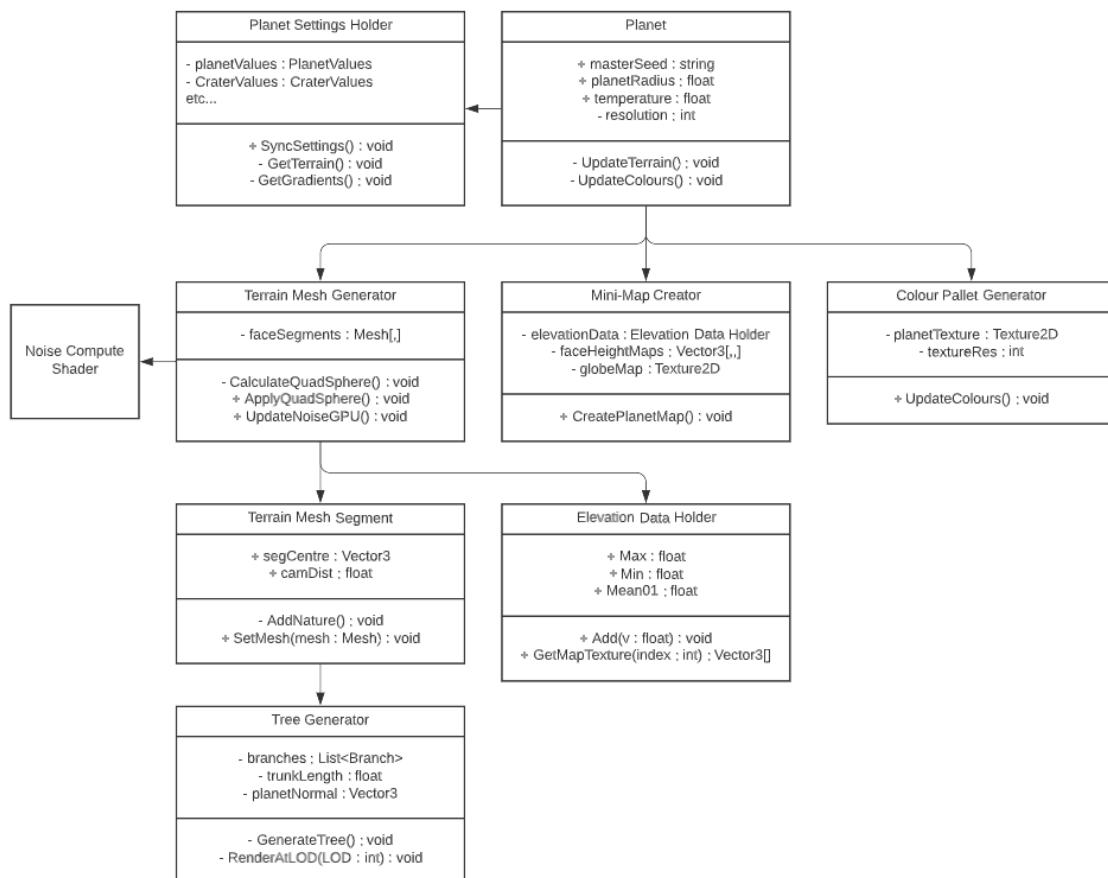
- I would have liked to implement machine learning into my NPCs for intelligent movement. However, the limited inclusion of NPCs in my game made this problem insignificant, and the time required to train the machine would be high, so I instead implemented decision trees to handle the NPC's movement. Moreover, my play testers had no problem with the NPC's AI as it stood, and they felt I had no real reason to upgrade it.
- I wanted to have physical clouds in my game for the planets. Unfortunately, doing so would require me to write a vertex shader, something I have limited experience in doing, and so the entire algorithm seemed a bit too complex for what it was worth. Instead, I can create a shader like my atmosphere shader that renders a thick layer of atmosphere above a thin layer, giving a similar effect to a cloud. Players I interviewed agreed that this was a suitable substitution for physical clouds as they acted mechanically the same, one suggested that perhaps I could include it after I hired a graphic designer as they would have a clearer idea of how a cloud could be designed.

Design

Planet Terrain Generation

Planet Class Diagrams

Each planet has many aspects and for each I will create a class. Hence, my class diagram shows their connections and the most important variables stored within them. Many classes not mentioned here for example are disconnected from all others and use the `UnityEngine.Start` function exclusively to complete their goals. Other classes are used to store variables or static functions.



Class Name	Objective
Planet	To display debug settings for the planet and to contain and organise all sub-classes used to generate the planet.
Planet Settings Holder	To store variables fundamental to the Planet's generation.

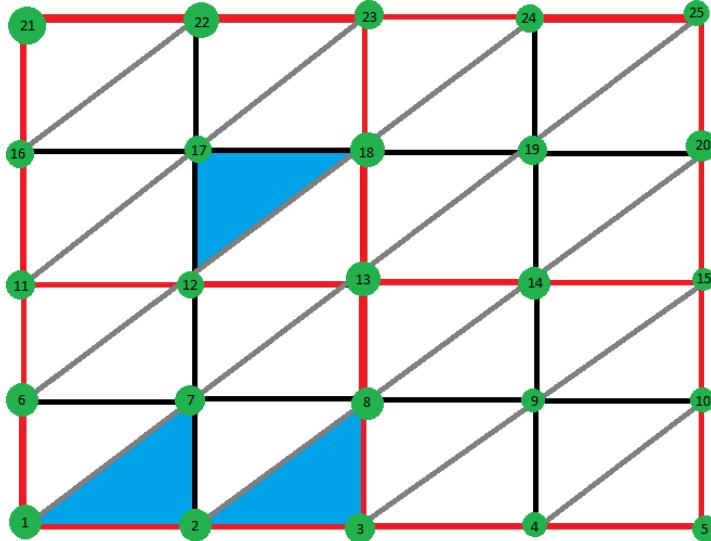
Terrain Mesh Generator	Generate the quad sphere mesh in segments and generate the layered noise functions to give the planet terrain.
Terrain Mesh Segment	To handle the generation of objects such as trees that should only appear when the player is close.
Tree Generator	Generate and display the tree at the appropriate level of detail.
Elevation Data Holder	Store the altitude of each vertex of the planet, as well as meta data such as the maximum, minimum and mean altitude.
Mini-Map Creator	To gather the planets elevation data to generate a 3D heightmap of the planet and convert it to a 2D planet map.
Colour Pallet Generator	Generate the planet's terrain and atmosphere colour constrained by the temperature of the planet. Create multiple biomes each with its own, similar, colour pallet. Atmosphere colour pallet is 2D, one dimension interpolates between sunset colour and mid-day colour.

Quad sphere mesh generation

Overview

As a first step towards creating planets for the player to traverse, I need to generate the mesh that the planets will be created from. A sphere mesh is suitable for this task. I settled on generating a quad sphere (as opposed to an icosphere) as the vertex placements on a quad sphere allow the mesh to be easily divided into segments, necessary since the vertex count is too high for the entire mesh to be rendered by one of Unity's Mesh Renderers.

For optimal performance on run-time, the Meshes produced for a specific planet at a specific resolution by the Quad-sphere Mesh Generation algorithm are saved to the asset resources, meaning that I only need to run the algorithm once, and then can simply load the Meshes from the asset resources each time I want to use them.

Visuals*Mesh Indices*

face = 1
resolution, r = 5

Let there be 4 segment meshes composing this face
(outlined in red)

The vertex (green) indicies show the order in which they are added to the segment meshes.

We compose the triangles with indicies local to each segment, but here I will show how to do it globally as the algorithm for locally is identical

The triangles indicies must be added in clockwise order for "our" side of the triangle to be rendered.

One such triangle is [1, 7, 2] another [2, 8, 3]
another [12, 17, 18].
Generally we have [n, n+r, n+r+1] and [n, n+r+1, n+1].

So our triangle int array T can be constructed by:

```
FOR n = 1 TO r^2 - r - 1 DO
    T.Add( n, n+r, n+r+1 )
    T.Add( n, n+r+1, n+1 )
ENDFOR
```

Data structures required

To store the 864 meshes generated, I will use a 2D array of Meshes. This is because there are 6 faces, and 144 meshes per face and grouping the meshes by their face simplifies my planet equirectangular map generation, where each face is projected independently onto the map.

Meshes are stored in Unity's built-in Mesh object. For the mesh to be rendered by Unity, each mesh must contain all the vertices within the mesh in a Vector3 array, all the triangles as groups of 3 vertex indices in an int array, and UV coordinates for each vertex are stored in a Vector2 array.

Object (instance name if applicable)	Contents (if applicable)
faceSegs as Mesh[6, 144]	Mesh
Mesh	vertices as Vector3[] triangles as int[] uvs as Vector2[]

Pseudocode

```
Function GenerateQuadSphereMesh(resolution As Integer)

    faceSegs = New List(Of Mesh) {6, 144}

    'For each face on the cube
    For face = 0 To 5 Do

        first_vertex As Vector3 = GetFirstVertexForFace ( face ) '[1, 1, 0] Or [-1, 1, 0] Or [0, 1, 1] etc.
        x_vector As Vector3 = GetXVectorAlongFace( face )
        y_vector As Vector3 = GetYVectorAlongFace( face )

        For x = 0 To resolution Do
            For y = 0 To resolution Do
                X = x / resolution
                Y = y / resolution
                'Get a linearly interpolated coordinate on this face of
                the mesh.
                'All the coordinates for the face form an even grid of
                scale based on the resolution (see diagram)
                pos As Vector3 = first_vertex + x_dir * X + y_dir * Y
                faceSegs[face, Floor( 12 * X ) * 12 + Floor( 12 * Y
            )].verticies.Add( pos.normalize ) 'normalize the vector To project the cube onto the
            unit sphere
                faceSegs[face, Floor( 12 * X ) * 12 + Floor( 12 * Y
        )].uvs.Add( New Vector2(X, Y) )
        Next
    Next
    Next

    CalculateMeshTrianglesSuchThatTheMeshIsRenderedProperly(mesh_segments) 'See the
    diagram for pseudocode for this function

    ApplySegmentsToPlanet(mesh_segments) 'Create segment gameObjects foreach
    mesh_segment and assign their parents In the scene hierarchy

End Function
```

Adding procedural noise to the mesh

Overview

The second step in planet terrain generation is adding procedural noise to the mesh. This makes it produce a natural variety of planets. To do this, I sample noise at vertex positions on the quad sphere we generated from fractal noise functions and scale each vertex by a factor given by a

function of this sampled noise. I use different noise functions parameters to produce the main land mass (continents) and mountains and layer them for the final mesh. This requires a lot of computation, so I will use compute-shaders that run on the GPU in parallel to calculate these scale factors and this will reduce the computation time by 2 orders of magnitude.

Data structures required

I will store the parameters used by the main land mass generator and mountain generator in two separate structs. To procedurally generate these parameters, I need to add a system to handle all the seeds I want to use for my planet generation, I can then use the ‘terrain seed’ to realistically randomise these parameters. To send these parameters and the mesh parameters to the compute-shader, I need to use compute buffers. Meta-data about the resultant terrain will then be calculated and stored in a separate class named Elevation Data, for use in a variety of scripts.

Object (instance name if applicable)	Contents (if applicable)
Structure NoiseValues	seed as int octaves as int scale as float persistance as float lacunarity as float
Structure ContinentNoiseValues	noiseValues as NoiseValues dropoff as float
Structure MountainNoiseValues	noiseValues as NoiseValues dropoff as float initialExtraOctaves as int
Structure RoughNoiseValues	noiseValues as NoiseValues startingOctave as float
Structure WarpNoiseValues	noiseValues as NoiseValues

The **NoiseValues** contents are demonstrated in the Pseudocode section.

The dropoff variable reduces the altitude of the continent and mountain terrain at points above a certain threshold. This is to eliminate unrealistically bumpy terrain.

The initialExtraOctaves variable controls how many layers of octave 1 noise are added to the mountain terrain. Increasing this integer generally increases the number of mountains on the planet.

RoughNoise is added to the terrain at points otherwise below sea-level, if the planet should not have an ocean. The startingOctave variable controls at which octave noise is added to the quad-sphere mesh. I expect this value to be greater than 1, since I want the noise to be relatively flat (as though it was a sea-bed).

WarpNoise is used to offset the ContinentNoise sampling points. The effect of this is shown in the Visuals section.

Object (instance name if applicable)	Contents (if applicable)
terrainSeed as int	
colourSeed as int	
environmentSeed as int	
masterSeed as string	b26(terrainSeed) + b26(terrainSeed) + b26(terrainSeed)

terrainSeed controls the appearance of the terrain generated (Used to generate the terrain as described in this section).

colourSeed controls the colour pallet of the planet (Specific uses shown later).

environmentSeed controls everything generated at run-time (Specific uses shown later).

A string masterSeed is produced so that a specific planet can be re-generated from a copyable string.

Pseudocode for function **b26**:

```
Const alphaIndexLookup As List(Of Each letter In the alphabet)

'Converts an integer to a string of lowercase letters
Function b26(i As Integer)

    alpha As String = alphaIndexLookup[i % 26]
    i /= 26
    If i IsNot 0 Then
        alpha += b26(i)
    End If
    Return alpha

End Function
```

Object (instance name if applicable)	Contents (if applicable)
elevations as float[]	float , storing each amplitude of each vertex of the terrain generated
vertSegments as List<Vector3[]>	Vector3[] , a vertex array for each mesh in faceSegs, added to the list in the order of the meshes in faceSegs
Class ElevationData	Max as float Min as float Mean01 as float upperQuartile01 as float lowerQuartile01 as float interQuartileRange01 as float

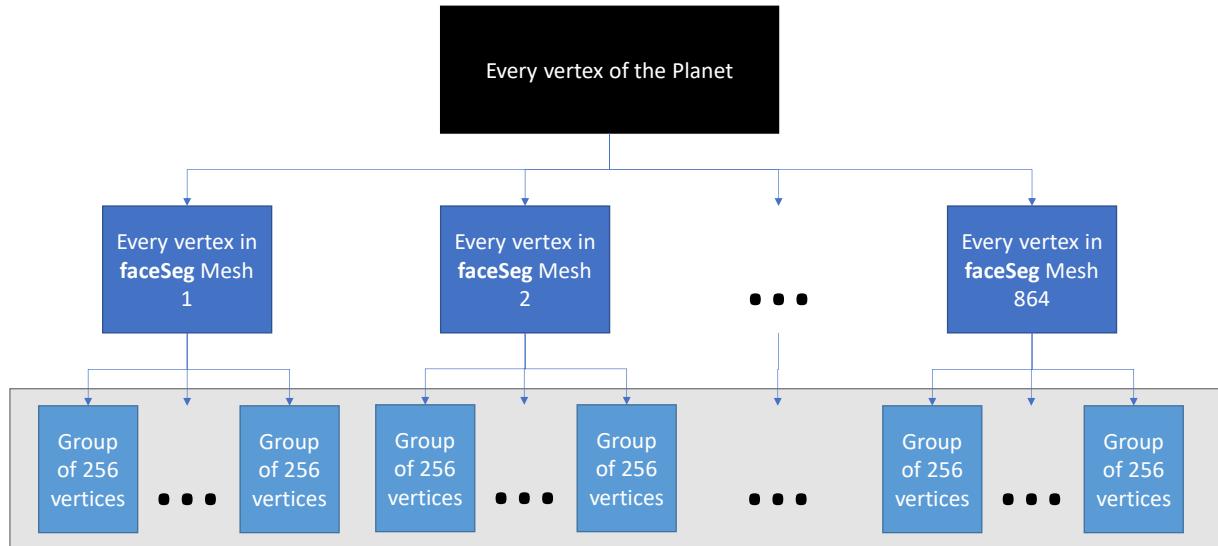
ElevationData variables are generated from the array of elevations given by the terrain generator compute-shader.

Visuals

Noise function GPU distribution

For the noise function to run for each vertex on the GPU we must copy all our noise variables into the compute-shader and add a compute buffer for each array we require in the program (this includes all the noise offsets we use as well as the array of quad-sphere mesh vertices we will scale).

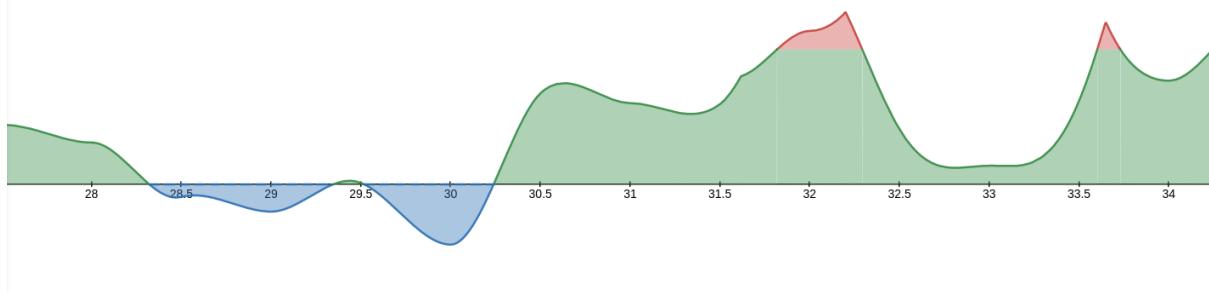
Here is how I will distribute the vertices so their noise can be produced in parallel:



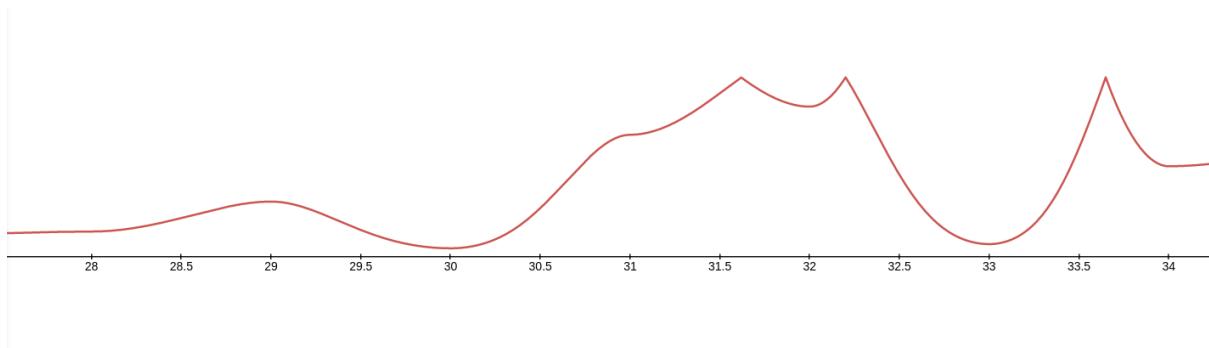
The GPU is fed each of the groups of 256 vertices, the noise function is then executed on each vertex in the group simultaneously.

Terrain Generation

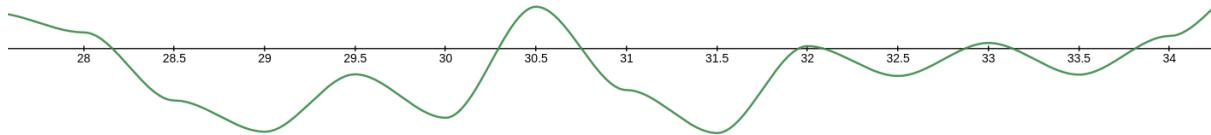
This noise is then layered to produce terrain like below:



Which is the sum of mountain terrain:



And the continent terrain:

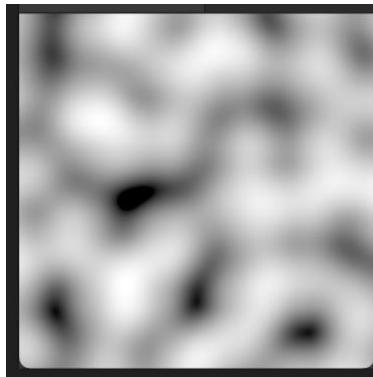


This is like how terrain will be generated for my planets. In the above, the altitude of the terrain at an x-ordinate is given by a noise function that takes this x-ordinate as an input. For my planets, the altitude of the terrain in a direction away from the planet is given by a noise function that takes this direction vector as an input.

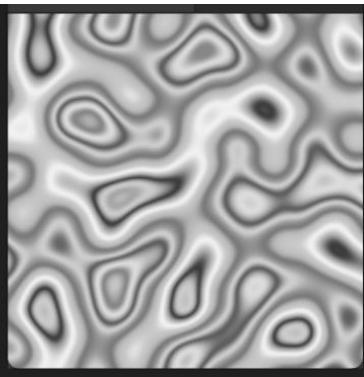
Warped Noise

For the continent noise function, this inputted direction vector is first offset by a position vector given by another noise function that takes the original direction vector as an input. This method describes warped noise function, which produces more detailed and interesting terrain than a standard noise function.

Standard Noise:

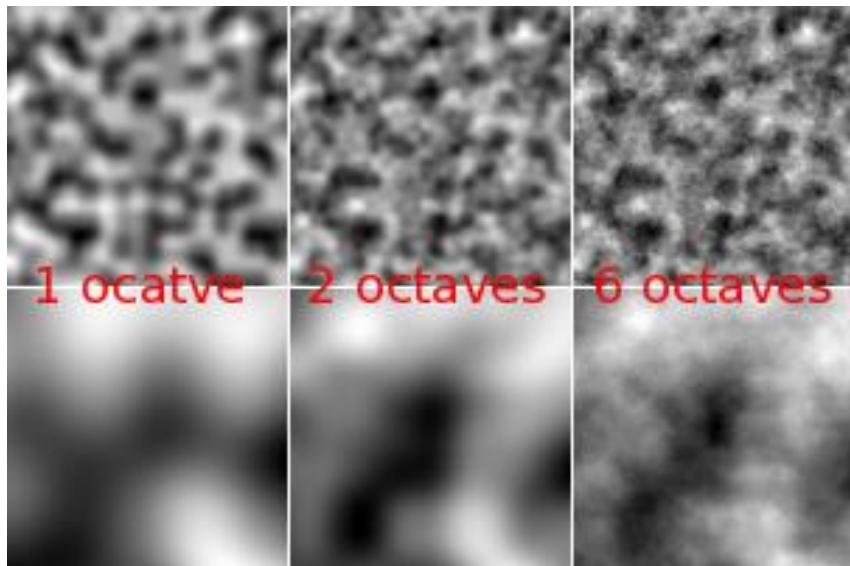


Warped noise:



Fractal Noise

Additionally, we layer multiple octaves of noise in our noise function to make the terrain more detailed:



Pseudocode

The pseudocode demonstrates how I will generate fractal noise (by layering noise of multiple octaves)

```
Function NoiseFunction(vert As Vector3, persistance As Float, lacunarity As Float,
scale As Float, octaves As Integer, offsets As List(Of Vector3))

    height As Float = 0.0
    amp As Float = 1.0
    freq As Float = 1.0

    For i As Integer = 0 To octaves - 1
        samplePoint As Vector3 = vert * freq + offsets(i)

        'the range of PerlinNoise3D is [-1, 1] so we add one because we want the
        vertex magnitude to, generally, increase
        perlinValue As Float = 1 + PerlinNoise3D(samplePoint * scale)
        [A]

        height += perlinValue * amp

        amp *= persistance
        freq *= lacunarity
    Next
    [B]

    Return height * vert
End Function
```

At [A], for the mountain noise function we insert the code:

```
If i == 0 Then

    For j As Integer = octaves To j < initialExtraOctaves
        samplePoint = vert * freq + offsets(j)
        perlinValue = Maximum(perlinValue, 1 + PerlinNoise3D(samplePoint * scale))
    Next
End If
```

To add our extra initial octaves to the mountain noise we are generating (Recall that this essentially adds more mountains to the planet).

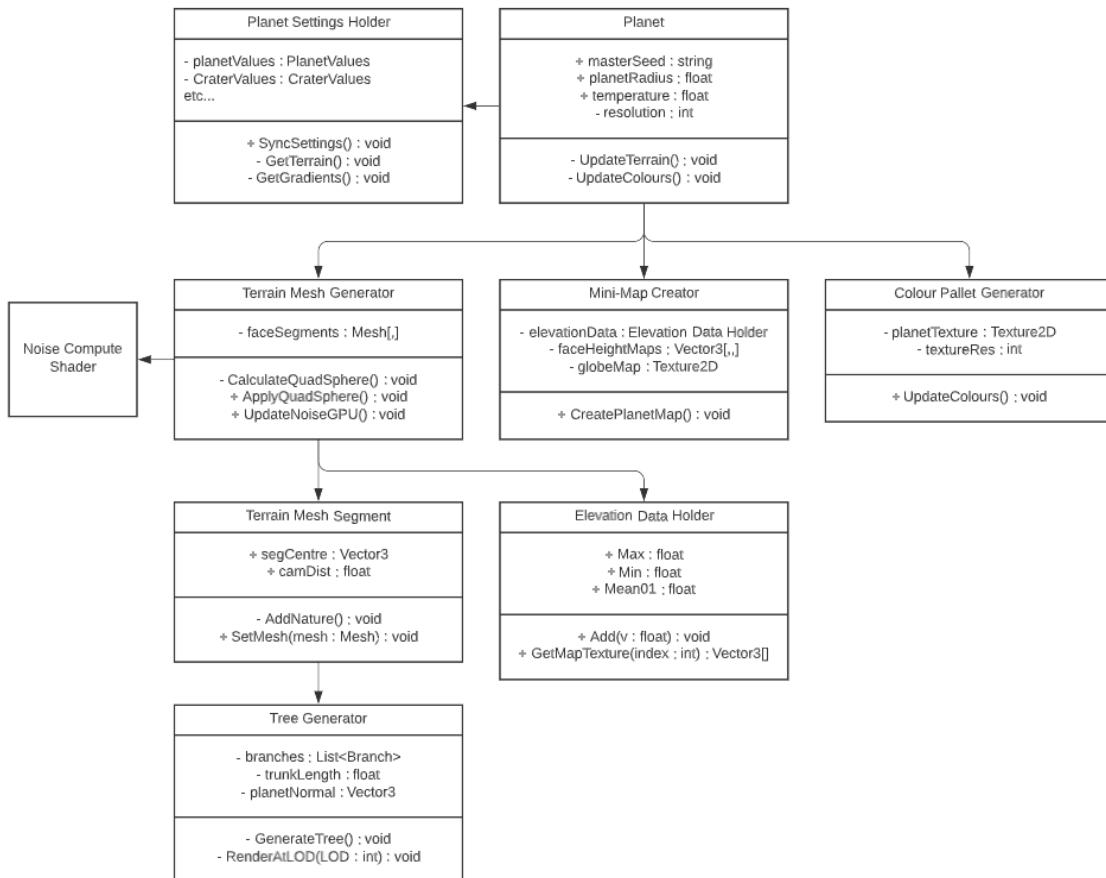
At [B], for the mountain and continent noise functions we insert the pseudocode:

```
If height > groundLevel Then
    'for mountains [height = height * height * dropoff]
    'for continents [height = Lerp(groundLevel, height, dropoff)]
End If
```

Where dropoff is a small positive real number. We square the height for mountains to retain their high peaks, before scaling the height by dropoff. For continents, the new height (altitude) is given scaling the difference between the altitude and ground level by dropoff, and then adding the ground level, which effectively flattens the continents producing realistic bumps in the terrain.

Planet Biome Generation

Data structures required



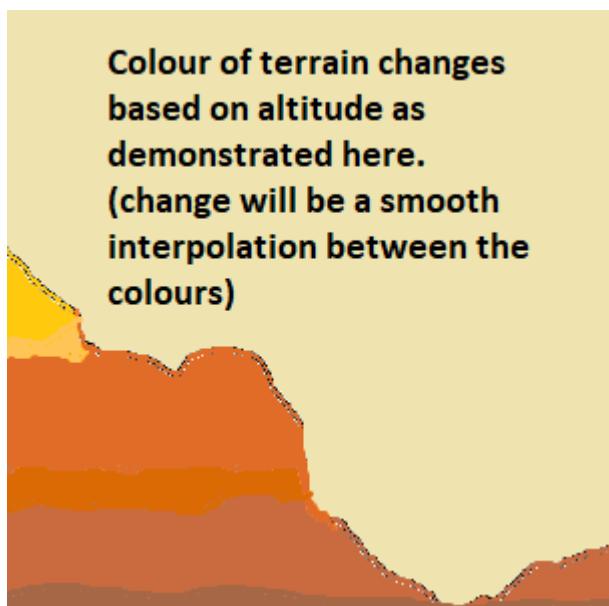
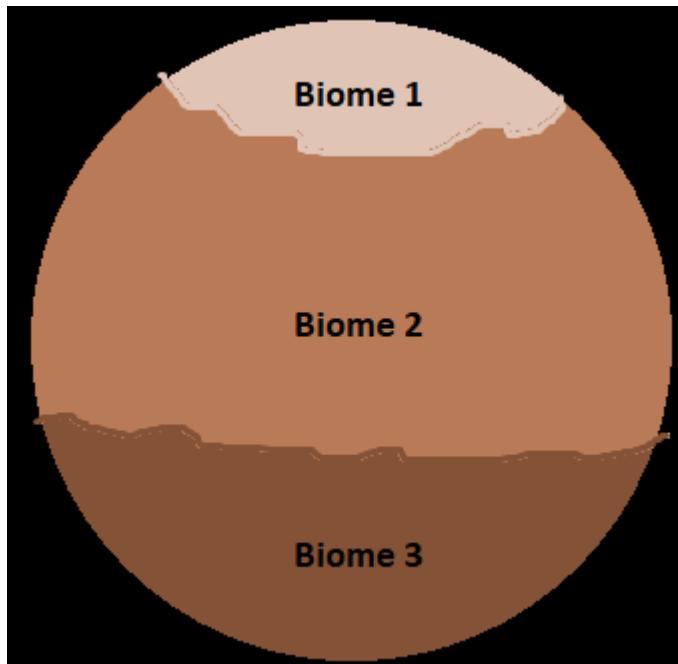
Biome Generation

Overview

Now that we have a planet with interesting terrain, we need to make it visually pleasing by first applying colours to the terrain. I have decided to split my planet latitudinally into three biomes, each of which has its own colour pallet that determines the colour of terrain at a specific altitude and gradient. These colour pallets are based on the temperature property of the planet. For instance, the colour pallets are tinted red for hotter planets and blue for cooler planets.

Visuals

Biomes are split by latitude



Please note that colour also changes based on the terrain gradient (more specifically, the dot product between the mesh normal vector at a point and the direction vector from the planet centre to the point).

Data structures required

Object (instance name if applicable)	Contents (if applicable)
biomeGradients as Dictionary<float, Gradient>	float , the minimum latitude of the biome (0 for the south pole, 1 for north pole) Gradient , the biome colour pallet, a gradient of colours applied to terrain in the biome from low altitude to high altitude.
planetTexture as Texture2D	Biome colour pallet used at each latitude of the planet.

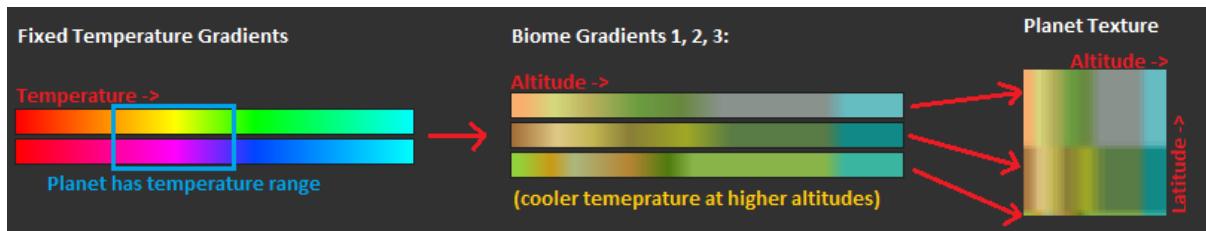
Gradient	time as float value as Color Defined by values set at specific times, getting the gradient value at times in-between returns a linear interpolation of the two nearest (in time) values set.
Texture2D	A 2D array in pixel space of Colors at specific coordinates.
Color	r as float g as float b as float

We use a dictionary to store the biomeGradients as this is the simplest way to associate two variables in an enumerable.

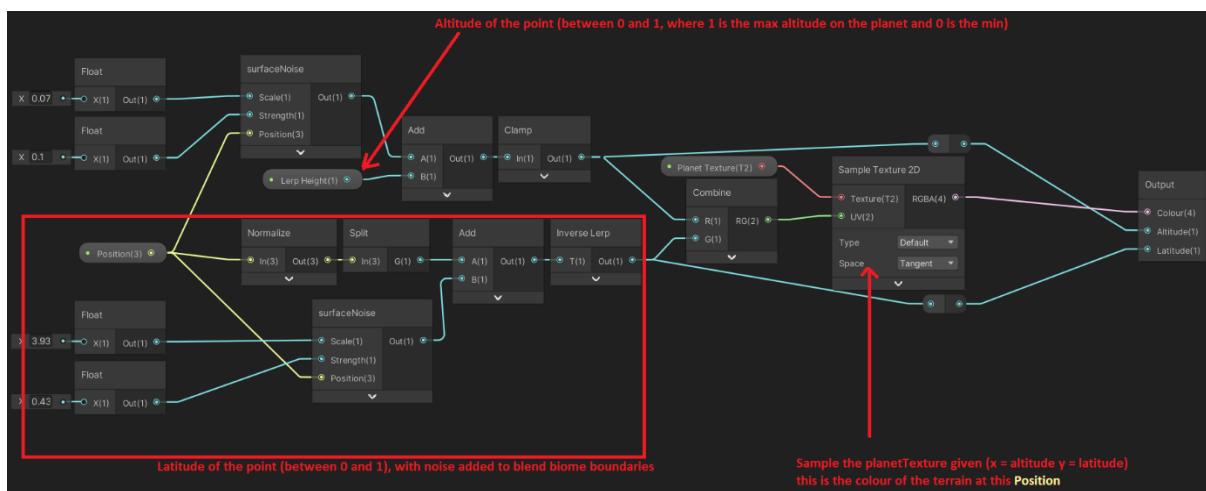
The planetTexture is then set into the planetTerrainShader, which assigned terrain colour based on the altitude of the terrain.

Pseudocode

Abstractly, here is an example of how the planetTexture is generated randomly given a temperature:



Here is a flowchart showing how the planetTerrainShader will use the planetTexture to shade a vertex (may need to zoom in to see).



Here is pseudocode showing how each biomeGradient is generated:

```

Function NewBiomeGradient(biomeTemperature As Float, heatLerp As Float) As Gradient
    heatGrad As Gradient = New Gradient()
    heatColourKeys As GradientColorKey() = New GradientColorKey() {3}
    heatAlphaKeys As GradientAlphaKey() = New GradientAlphaKey() {3}
    i As Integer = 0

    'Blend between the two fixed heat gradients by heatLerp and store in heatGrad
    For Each _t As Float In (0, 0.4, 0.6, 1)
        heatColourKeys(i).color = Lerp(fixedTempGrad1.Evaluate(_t),
fixedTempGrad2.Evaluate(_t), heatLerp)
        heatColourKeys(i).time = _t
        heatAlphaKeys(i).alpha = 1
        i += 1
    Next
    heatGrad.SetKeys(heatColourKeys, heatAlphaKeys)

    lowerMean01 As Float = elevationData.lowerMean01
    upperMean01 As Float = elevationData.upperMean01
    gradient As Gradient = New Gradient()
    keys As Integer = 6
    'Initial time is smaller if terrain is more common at lower altitudes, this
    concentrates more colours at lower altitudes if necessary
    time As Float = Lerp(0, lowerMean01, Random.value * Random.value)
    'Want at least one new colour beyond the upperMean
    minTimeInc As Float = (upperMean01 - time) / (keys - 1)
    'Want no new colours in the top 20% of altitudes, we will manually add these
    maxTimeInc As Float = (0.8 - time) / (keys - 1)
    temperature As Float = biomeTemperature
    'Temperature decreases with altitude
    temperatureInc As Float = Min(0.2, (1 - temperature)) / (keys - 1)
    colourKeys As GradientColorKey() = New GradientColorKey(keys + 2 - 1) {}
    alphaKeys As GradientAlphaKey() = New GradientAlphaKey(keys + 2 - 1) {}

    For j As Integer = 0 To keys - 1
        colourKeys(j).time = time
        'A colour at a time on the gradient is a 50/50 mix of the appropriate
        temperature colour for the time and a random colour
        '(makes planets more visually unique while still being coherent)
        colourKeys(j).color = Lerp(heatGrad.Evaluate(temperature), Random.color, 0.5)
        alphaKeys(j).alpha = 1
        time += Random.Range(minTimeInc, maxTimeInc)
        temperature += Random.Range(0, temperatureInc)
        temperature = Clamp01(temperature)
    Next

    'Manually add colours in the top 20% of altitudes (to resemble the cold white tips
    at the top of mountains)
    colourKeys(keys).time = 0.8
    colourKeys(keys).color = colourKeys(keys - 1).color
    alphaKeys(keys).alpha = 1
    colourKeys(keys + 1).time = 0.85
    colourKeys(keys + 1).color = Color.Lerp(heatGrad.Evaluate(biomeTemperature +
0.5F), Random.color, 0.5)
    alphaKeys(keys + 1).alpha = 1
    gradient.SetKeys(colourKeys, alphaKeys)
    Return gradient
End Function

```

Here is pseudocode showing how the planetTexture is generated:

```

Function GeneratePlanetTexture(textureRes As Integer)
    planetTexture = New Texture2D(textureRes, textureRes);

    lerpRange As Float = 0.05

    For lat As Integer = 0 To textureRes - 1

        lat01 As Float = Float(lat) / textureRes
        biomeBelow As KeyValuePair(Of Float, Of Gradient) = New KeyValuePair(Of Float,
        Of Gradient)(0, Nothing)
        biomeAbove As KeyValuePair(Of Float, Of Gradient) = New KeyValuePair(Of Float,
        Of Gradient)(2, Nothing)

        'Find the biome to apply at latitude lat [biomeBelow] and the biome above this
        one if there is one [biomeAbove]
        For Each biome As KeyValuePair(Of Float, Of Gradient) In biomeGradients

            If biome.Key <= lat01 And lat01 - biome.Key <= biomeBelow.Key - lat01 Then
                biomeBelow = biome
            ElseIf biome.Key > lat01 And biome.Key - lat01 <= biomeAbove.Key - lat01
Then
                biomeAbove = biome
            End If
        Next

        For alt As Integer = 0 To textureRes - 1
            alt01 As Float = Float(alt) / textureRes
            pixel As Color
            'Find the colour to use at altitude alt given belowBiome, and blend this
            colour with the colour given by aboveBiome, for a smooth transition between the two
            biomes
            If biomeAbove.Key <> 2 And biomeAbove.Key - lerpRange < lat01 Then
                pixel = Color.Lerp(biomeBelow.Value.Evaluate(alt01),
biomeAbove.Value.Evaluate(alt01), Mathf.InverseLerp(biomeAbove.Key - lerpRange,
biomeAbove.Key, lat01))
            Else
                pixel = biomeBelow.Value.Evaluate(alt01)
            End If

            'Assign the colour to pixel coordinate (alt, lat) on the planetTexture
            planetTexture.SetPixel(alt, lat, pixel)
        Next
    Next

    ApplyToMesh(planetTexture)
End Function

```

Planet Map Generation

Overview

To produce an equirectangular projection (which I will refer to as a map) of the planetTexture applied to the planet we need to associate a pixel on the map with the latitude and altitude of the vertex projected there. As an intermediate stage therefore, between the vertex array and the coloured map texture, we should produce a greyscale “height-map” texture, where the value (0 to 1) of each pixel corresponds to the altitude of the vertex projected there and given also the y-coordinate of the pixel as a measure of latitude, we can then, directly produce the correctly coloured map from the “height-map”. We include an intermediate step because, as shown above, the colouring of the planet is handled by a subroutine of the planetTerrainShader, given the altitude and

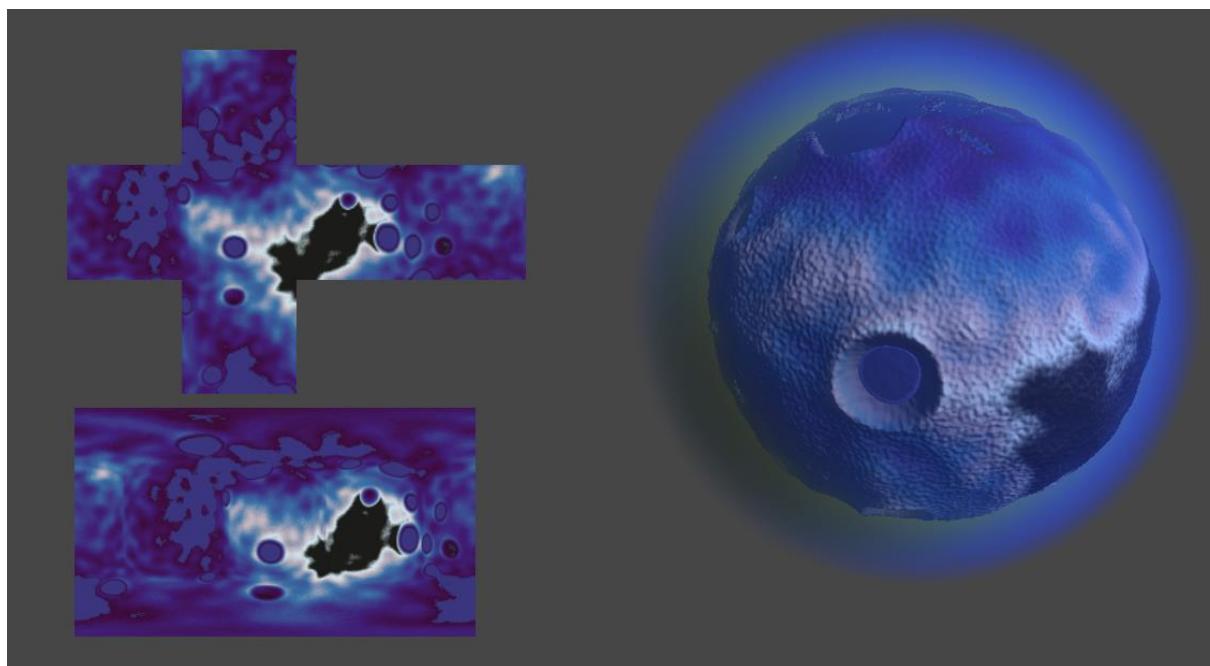
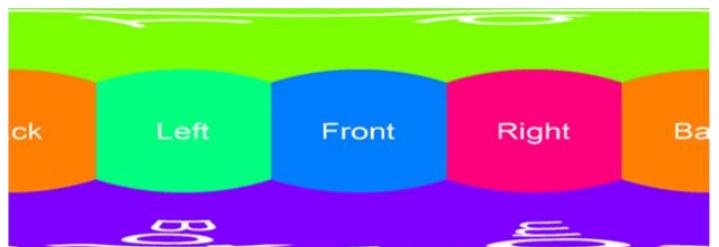
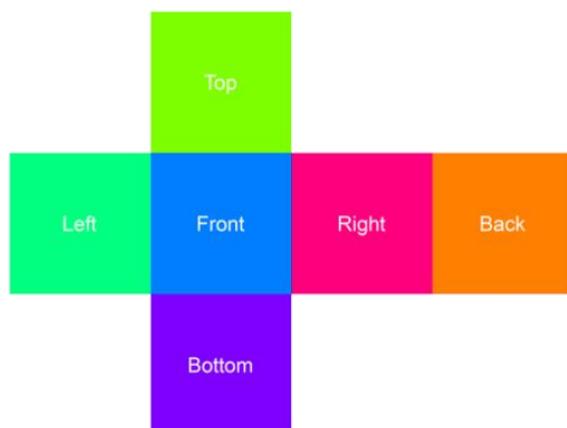
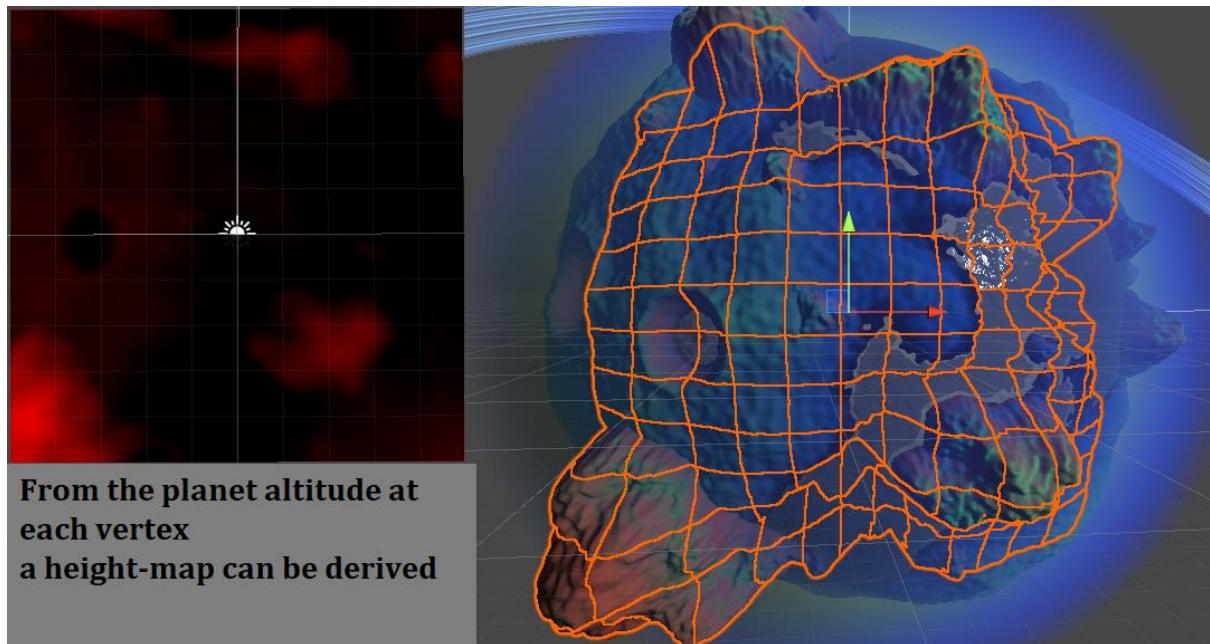
latitude of a Position. We can therefore use our “height-map” as an input to this subroutine to get our colours generated on the GPU.

Our planet is a 6-faced quad-sphere, made of 864 mesh-segments, and the ordering of both the vertices in each mesh (given a “bottom” and “left” the ordering goes from “bottom-left” to “top-right”) and these meshes in the faceSegs array (same ordering style vertices), by my design, allows us to directly retrieve a vertex from the faceSegs array corresponding to a coordinate on the cube (*) from which the quad-sphere was “inflated” (no complex algorithm required). The “height-map” is a 2D texture with longitude along the x-axis and latitude along the y-axis, so we can translate these coordinates to polar coordinates on a sphere, giving us a direction vector. We can then project this direction vector onto the cube (*) (using a process shown in the visuals section), giving us a coordinate on the cube (*). And as described above, with this cube coordinate we can finally retrieve the corresponding vertex and its altitude (the magnitude of the vertex).

Above, I overviewed my method to produce an equirectangular projection of vertex altitudes on the planet, I will go into more detail in the pseudocode section. We can adjust the resolution of the map simply by changing the texture’s pixel dimensions, so that the map is generated in an acceptable time-window.

Visuals

The following images show how each face of the planet corresponds to a face on the cube net, and how the faces on the cube net are combined to produce the equirectangular projection:



Data structures required

Object (instance name if applicable)	Contents (if applicable)
faceHeightMaps as Vector3[,,]	3D array of every vertex on the planet. A vertex at a uv coordinate on a face can be retrieved using faceHeightMaps[face#, uv.x, uv.y] . Given the fixed number of vertices I will be using: the shape of the array is [6, 612, 612] and the size is [2247264]
Gradient	time as float value as Color Defined by values set at specific times, getting the gradient value at times in-between returns a linear interpolation of the two nearest (in time) values set.
Texture2D	A 2D array in pixel space of Colors at specific coordinates.
Color	r as float g as float b as float

Pseudocode

Below is pseudocode for generating the array **faceHeightMaps** (which we can then use to “retrieve a vertex at a cube coordinate”):

```

Function CreateHeightMapCubeMesh(resolution As Integer, segsPerFaceRow As Integer,
vertSegments As List(Of Vector3))

    segsPerFace As Integer = segsPerFaceRow * segsPerFaceRow
    vertsPerSegRow As Integer = resolution / segsPerFaceRow + 1
    vertsPerSeg As Integer = vertsPerSegRow * vertsPerSegRow
    vertsPerFaceRow As Integer = vertsPerSegRow * segsPerFaceRow
    vertsPerFace As Integer = vertsPerFaceRow * vertsPerFaceRow
    faceHeightMaps As List(Of Vector3) = New List(Of Vector3) {5, vertsPerFaceRow - 1,
    vertsPerFaceRow - 1}

    For faceNo As Integer = 0 To 5

        'Texture2D sets pixels bottom row (left to right) to top row. Therefore, we
        need to convert the vertex arrangement of the face to this order
        'For face 0 this order is: left column (bottom to top) to right column
        'For face 1: right column (bottom to top) etc.

        leftStart As Boolean = faceNo = 0 Or faceNo = 2 Or faceNo = 5
        bottomStart As Boolean = faceNo = 0 Or faceNo = 1
        firstSeg As Integer = faceNo * segsPerFace
        bottomLeftSeg As Integer = 0
        bottomLeftVert As Integer = 0

        If leftStart And bottomStart Then
            bottomLeftSeg = firstSeg
            bottomLeftVert = 0
        ElseIf leftStart And Not bottomStart Then
            bottomLeftSeg = firstSeg + segsPerFaceRow - 1
            bottomLeftVert = vertsPerSegRow - 1
        ElseIf Not leftStart And bottomStart Then
            bottomLeftSeg = firstSeg + segsPerFace - segsPerFaceRow
            bottomLeftVert = vertsPerSeg - vertsPerSegRow
        ElseIf Not leftStart And Not bottomStart Then
            bottomLeftSeg = firstSeg + segsPerFace - 1
            bottomLeftVert = vertsPerSeg - 1
        End If

        For x As Integer = 0 To vertsPerFaceRow - 1

            For y As Integer = 0 To vertsPerFaceRow - 1

                'Get the vertex index given face uv coordinate (x, y)
                seg As Integer = bottomLeftSeg
                seg += x / vertsPerSegRow * segsPerFaceRow * (If(leftStart, 1, -1))
                seg += y / vertsPerSegRow * (If(bottomStart, 1, -1))
                verts As List(Of Vector3) = vertSegments(seg) 'Same vertSegments as
                described in Planet Terrain Generation
                localX As Integer = x - x / vertsPerSegRow * vertsPerSegRow
                localY As Integer = y - y / vertsPerSegRow * vertsPerSegRow
                vert As Integer = bottomLeftVert
                vert += localX * vertsPerSegRow * (If(leftStart, 1, -1))
                vert += localY * (If(bottomStart, 1, -1))

                faceHeightMaps(faceNo, x, y) = verts(vert)
            Next
        Next
    Next
End Function

```

Below is pseudocode for generating the “height-map” from our faceHeightMaps array:

```

Function CreateGlobeMap(resolution As Integer, segsPerFaceRow As Integer)
    vertsPerSegRow As Integer = resolution / segsPerFaceRow + 1
    vertsPerFaceRow As Integer = vertsPerSegRow * segsPerFaceRow
    normalLength As Integer = Floor(Sqrt(#normals on globeMap Mesh))
    w As Integer = 800
    h As Integer = 400
    globeMap = New Texture2D(w, h)
    vertNormal As List(Of Vector3) = New List(Of Vector3) {#normals on globeMap Mesh}

    'Based off http://paulbourke.net/panorama/cubemaps/ (Converting to a spherical
    projection from 6 cubic environment maps, roughly a quarter down the website)
    For x As Integer = 0 To w - 1
        xNP As Float = Float(x) / w * 2 - 1 'NP means transformed to the range [-1, 1]

        For y As Integer = 0 To h - 1
            yNP As Float = Float(y) / h * 2 - 1
            vertex As Vector3 = VertAtPolarCoord(xNP, yNP, vertsPerFaceRow)
            globeMap.SetPixel(x, y, Color.white * InverseLerp(Min altitude, Max
altitude, vertex.magnitude))
        Next
    Next

    For x As Integer = 0 To normalLength - 1
        xNP As Float = Float(x) / normalLength * 2 - 1

        For y As Integer = 0 To normalLength - 1
            yNP As Float = Float(y) / normalLength * 2 - 1
            vertex As Vector3 = VertAtPolarCoord(xNP, yNP, vertsPerFaceRow)
            vertNormal(y * normalLength + x) = vertex
        Next
    Next

    'Apply the globeMap texture and vertNormal normals to the mesh of the map. We can
    then access this data from a shader which will then colour our "height-map" exactly
    like the planet, making it a regular map!
    ApplyToMesh(globeMap)
    ApplyToMesh(vertNormal)
End Function

```

And here is the pseudocode for the VertAtPolarCoord function I used above:

```

Function VertAtPolarCoord(x As Float, y As Float, vertsPerFaceRow As Integer) As
Vector3

    'I created a Geogebra file to help me visualise converting a direction to a point
    on a 'skybox' https://www.geogebra.org/m/kwkbpd5k. Variables and operations below
    reference the Geogebra file
    s As Float = x * PI
    t As Float = y * PI / 2
    a As Float = 1 / (Cos(t) *Cos(s))
    b As Float = 1 / Sin(t)
    c As Float = 1 / (Cos(t) *Sin(s))
    m As Float = Min(Abs(a), Abs(b), Abs(c))
    P As Vector3 = New Vector3(1 / a, 1 / b, 1 / c)
    M As Vector3 = m * P
    C As Vector3 = m * Dot(m, P) / Abs(Dot(m, P))
    c = (c + Vector3.one) / 2
    faceNo As Integer = If(m = Abs(a), If(a > 0, 0, 5), If(m = Abs(b), If(b > 0, 4,
2), If(c > 0, 3, 1)))

    samplePoint As Vector2 = Vector2.zero

    If faceNo = 0 Then
        samplePoint = New Vector2(c.z, c.y)
    ElseIf faceNo = 1 Then
        samplePoint = New Vector2(c.x, c.y)
    ElseIf faceNo = 2 Then
        samplePoint = New Vector2(c.z, c.x)
    ElseIf faceNo = 3 Then
        samplePoint = New Vector2(1 - c.x, c.y)
    ElseIf faceNo = 4 Then
        samplePoint = New Vector2(c.z, 1 - c.x)
    ElseIf faceNo = 5 Then
        samplePoint = New Vector2(1 - c.z, c.y)
    End If

    'Get the four closest vertices to the polar coordinate and interpolate between
    them (this would make the map look smoother at high resolution)
    sampleX As Float = samplePoint.x * (vertsPerFaceRow - 1)
    sampleY As Float = samplePoint.y * (vertsPerFaceRow - 1)
    dd As Vector3 = faceHeightMaps(faceNo, Floor(sampleX), Floor(sampleY))
    du As Vector3 = faceHeightMaps(faceNo, Floor(sampleX), Ceil(sampleY))
    ud As Vector3 = faceHeightMaps(faceNo, Ceil(sampleX), Floor(sampleY))
    uu As Vector3 = faceHeightMaps(faceNo, Ceil(sampleX), Ceil(sampleY))
    Return Lerp(Lerp(dd, du, sampleY - Floor(sampleY)), Lerp(ud, uu, sampleY -
Floor(sampleY)), sampleX - Floor(sampleX))
End Function

```

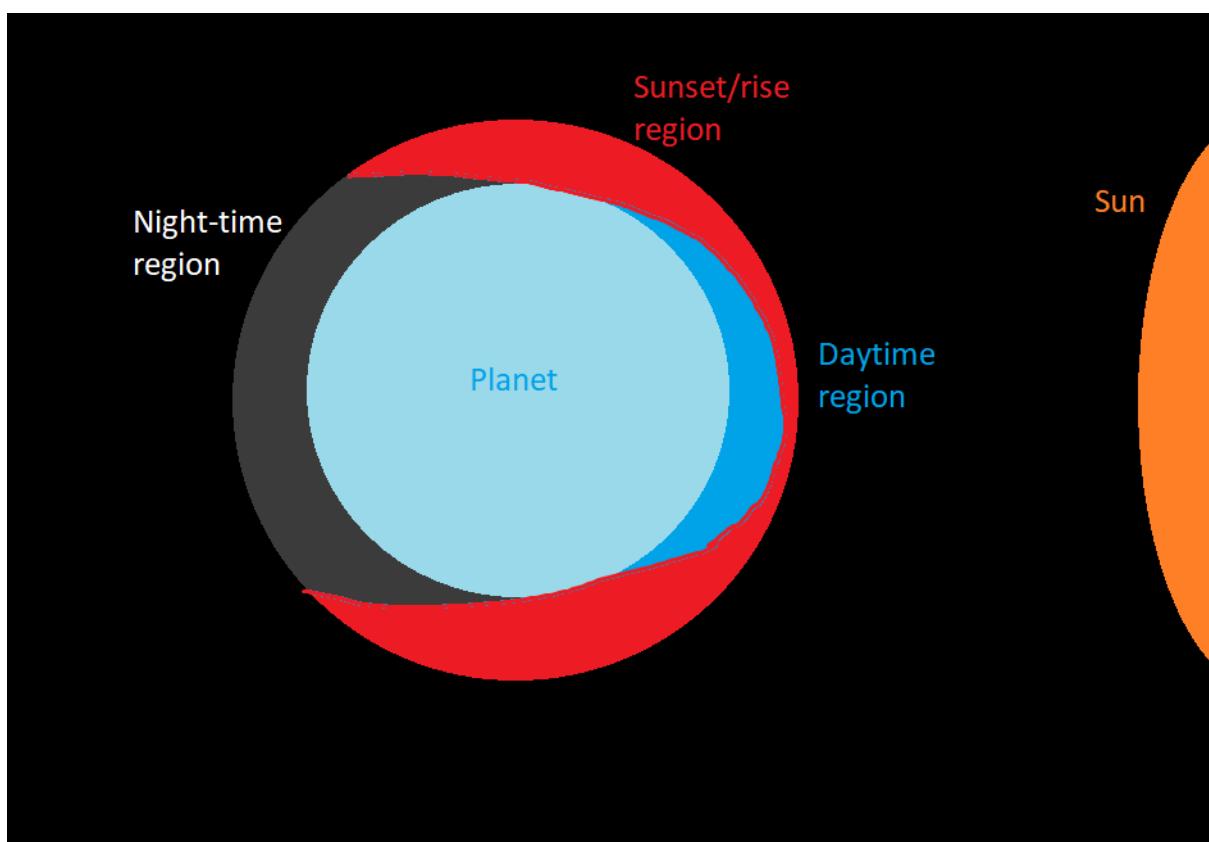
Planet Atmosphere Generation

Writing the shader

Overview

Now that we have planets with interesting terrain and suitable colours, we should add an atmosphere to make the planets seem more distinct and separate from the space around them. Since some planets are larger, more massive and hotter than others, it makes sense that the atmosphere generated has parameters to vary; namely, optical depth, colour pallet, and radius. Optically dense (which I will refer to as dense) atmospheres, generally produced for cooler planets, make less light permeate the atmosphere, which means if you are on the planet the horizon appears fogger, the sun becomes less visible and sunsets/rises have a lesser effect on the atmosphere colour. **I also want some of my atmospheres to resemble Jupiter's by having noisy colour bands at latitudes.**

Visuals



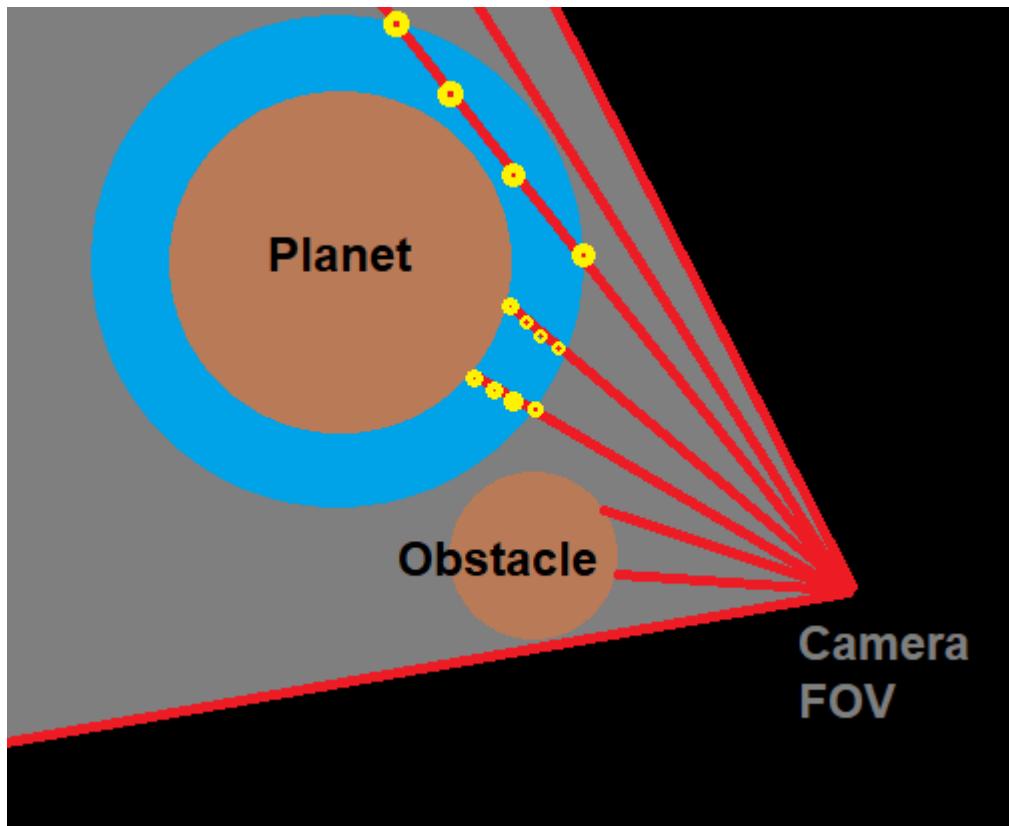
There are three colours which the three regions of the atmosphere are generated from. For the daytime region, this sampled from the planetTexture. For the sunset/rise region, this is a red-shifted colour of the sun. For the night-time region, this is black. For dense atmospheres, the effect of the

sunset/rise region is reduced significantly.

```

For each ray that intersects atmosphere:
    Sample optical density at each hoop on ray
Hence get total optical density along ray inside atmosphere
Multiply optical density along ray by distance of ray inside atmosphere
Inverse interpolate to get optical density between 0 & 1
return this as the optical density of the direction away from the camera

```



Pseudocode

The pseudocode for the shader is abstract, because I anticipate a lot of fine tuning to make all atmospheres aesthetically acceptable.

```
atmosphereDensity01 As Float

Public Function GetColourOfPixel(pixelUV As Float2)

    originalColour As Float4 = SampleTexture(mainTexture, pixelUV)
    ray As Float3 = GetDirectionOfPixel(pixelUV)
    distanceToAtmosphere As Float = RaySphere(ray, planetCentre, atmosphereRadius)
    opticalDensity01 As Float = GetOpticalDensityGiven(distanceThroughAtmosphere,
distanceToAtmosphere)
    lightIntensity01 As Float = GetLightIntensityGiven(distanceThroughAtmosphere,
distanceToAtmosphere)
    intersectPoint As Float3 = cameraPosition + ray * distanceToAtmosphere
    latitude01 As Float = InverseLerp(-atmosphereRadius, atmosphereRadius,
intersectPoint.y - planetCentre.y) + 0.1 * PerlinNoise()
    atmColour As Float4 = SampleTexture(atmColourTable, (latitude01,
opticalDensity01))
    atmColour = Lerp(Colour.black, atmColour, lightIntensity01)
    atmColour = Lerp(originalColour, atmColour, atmosphereDensity01)
    If distanceThroughAtmosphere < fogRange Then
        atmColour = Lerp(originalColour, atmColour, InverseLerp(0, fogRange,
distanceThroughAtmosphere))
    End If
    Return atmColour

End Function
```

As a bonus, we can also include the code to generate rings for our planets in this shader:

```

'Shown 1 ring here, planets may have 2 rings where additional accountancy for the
rings overlapping is required
Public Function ApplyRing(originalColour As Float4, ray As Single, distanceToRing As
Float)

    pointOnRing As Float3 = CameraPosition + ray * distanceToRing
    radius As Float = (pointOnRing - planetCentre).magnitude

    If (radius < minRingRadius And radius > maxRingRadius) Then
        Return Lerp(originalColour, ringColour, PerlinNoise(radius))
    End If

    Return originalColour

End Function

Public Function GetColourOfPixel(pixelUV As Float2)

    originalColour As Float4 = SampleTexture(mainTexture, pixelUV)
    ray As Float3 = GetDirectionOfPixel(pixelUV)

    distanceToRing As Float = dot(planetCentre - CameraPosition) / dot(rayNormal)
    distanceToAtmosphere As Float = RaySphere(ray, planetCentre, atmosphereRadius)

    If (distanceToRing < distanceToAtmosphere) Then
        originalColour = ApplyRing(originalColour, ray)
    End If

    'Atmosphere code here...

    If (distanceToRing > distanceToAtmosphere) Then
        atmColour = ApplyRing(atmColour, ray)
    End If

End Function

)

```

However, for this shader we first need to generate a texture which we can sample to give us the colour of the atmosphere at a latitude and how that colour changes when transitioning from the daytime region into the sunset/rise region.

```

Function GetAtmosphereTexture(atmosColour As Color, sunsetColour As Color, noiseBlend
As Float)

    numKeys As Integer = 8
    textureRes As Integer = 100

    atmosGradient As Gradient = New Gradient()
    colourKeys As GradientColorKey[] = New GradientColorKey[numKeys]

    For i As Integer = 0 To numKeys

        If i == numKeys - 1 Then
            colourKeys[i].color = colourKeys[0].color 'smooth wrap around
        Else
            colourKeys[i].color = Lerp(atmosColour, Random.Color, noiseBlend)
        End If

        i01 As Float = i / (numKeys - 1)
        colourKeys[i].time = i01 + Random.Range(-0.01, 0.01)

    Next

    atmosGradient.Set(colourKeys)
    atmosTexture = New Texture2D(textureRes, textureRes)

    lerpH As Float = sunsetColour.hue

    For hue As Integer = 0 To textureRes

        For i As Integer = 0 To textureRes

            latitudeColour As Color = atmosGradient.Evaluate(i / (textureRes - 1))

            latitudeColour.hue = Lerp(latitudeColour.hue, lerpH, hue / (textureRes - 1))

            atmosTexture.SetPixel(hue, i, latitudeColour)

        Next

    Next

End Function

```

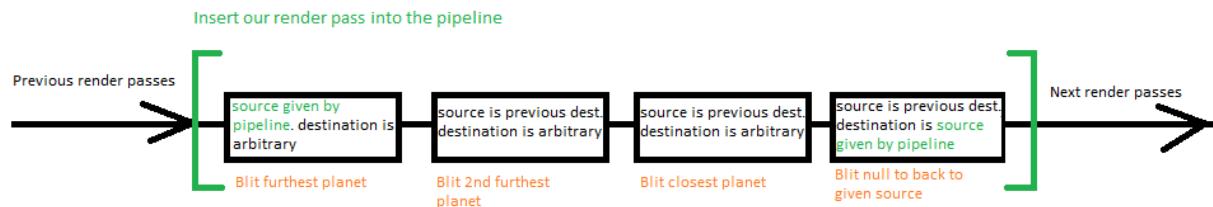
Implementing the shader

Overview

Because there is no surface on which we would want the atmosphere shader to work, we need to run it for every pixel on the screen (each frame). This means we need to implement the shader as a custom render pass in Unity's universal render pipeline. If there are multiple planets, we will need to add multiple render passes (with the planets closer being rendered last). We do this by *blitting* each atmosphere material in the appropriate order.

Visuals

In order to perform a blit, we need the identifier of the previous blit (the source), an arbitrary destination (handle) to blit to, and the material to blit. Here is how our blits will be inserted into the pipeline:



Data structures required

Object (instance name if applicable)	Contents (if applicable)
materials as Material[]	Array of materials (instances of the atmosphere shader) for each atmosphere ordered by distance to camera.
sources as RenderTargetIdentifier[]	Ordered array of source identifiers necessary to chain blits together.
destinations as RenderTargetHandle[]	Ordered array of arbitrary destinations to blit to.

Pseudocode

```

destinations As RenderTargetHandle[] 'store in class array so we can clean-up later

Function CustomRenderPass(sourceID As Integer, materials As List(Of Material))

    numBlits As Integer = materials.Length
    sources As RenderTargetIdentifier[] = New RenderTargetIdentifier[numBlits]
    destinations = New RenderTargetHandle[numBlits]

    i As Integer = 0
    For Each (material As Material In materials)

        If i == 0 Then
            sources[i] = sourceID
        Else
            sources[i] = destinations[i - 1].Identifier()
        End If

        destinations[i].Init()

        Blit(sources[i], destinations[i], material)

        i++
    Next

    Blit(destinations[i - 1], sourceID, Nothing)

End Function

```

Tree Generation

Overview / Visuals

The trunk (root branch) has a fixed diameter and length, from this, branches split off recursively. Branches are more likely to spawn near the end of their parent branch, are shorter than their parent, and have a smaller diameter. The number of recursion is fixed (set to 5 by default).

The colour of the branches and leaves is dependent on the planets colour pallet



Leaves on the tree are more likely to spawn on branches of higher recursion number, so that the canopy is at the top of the tree.

The direction the branches face in is given by the direction vector between the origin and a random point within one cone and outside another, both with their head at the origin. The direction of the cones is equal to the average of the parent branch and the normal of the surface the trunk is rooted in, and the angle the edge of the cones make with the origin is fixed (60 degrees and 30 degrees respectively iirc).

Pseudocode

```

Function GenerateTree()

    root = New GameObject("Root (1)")
    root.layer = gameObject.layer
    root.transform.position = transform.position
    root.transform.parent = transform
    Branch.treeObj = Me
    Branch.trunkLength = trunkLength
    Branch.planetNormal = planetNormal
    Branch.leafDensity = leafDensity

    fullRange As Vector2 = New Vector2(-Mathf.PI, Mathf.PI)
    branches = New List(Of Branch)()
    trunk As Branch = New Branch(Vector3.zero, trunkLength, 0.5F, Branch.planetNormal,
fullRange, 0)
    branches.Add(trunk)
    k As Integer = 0

    While k < branches.Count
        angleBuffer As Float = Random.Range(-Mathf.PI, Mathf.PI)

        For i As Integer = 0 To branches(k).splitsInto - 1

            If branches(k).branchWidth / branches(k).splitsInto < minBranchWidth Then
                Exit For
            ElseIf branches(k).consecutiveSplits > consecutiveSplits Then
                Exit For
            End If

            splitPoint As Vector3

            If i = 0 Then
                splitPoint = branches(k).endPoint
            Else
                splitPoint = Vector3.Lerp(branches(k).startPoint, branches(k).endPoint,
Random.Range(Random.Range(0.25F, 0.5F), 1))
            End If

            branchWidth As Float = branches(k).branchWidth / branches(k).splitsInto
            branchLength As Float = Random.Range(minBranchLength,
branches(k).branchLength)
            directionRange As Vector2 = New Vector2(angleBuffer + 2 * Mathf.PI * i /
branches(k).splitsInto, 0)
            directionRange.y = directionRange.x + 2 * Mathf.PI / branches(k).splitsInto
            branch As Branch = New Branch(splitPoint, branchLength, branchWidth,
branches(k).branchNormal, directionRange, branches(k).consecutiveSplits)
            branches.Add(branch)
        Next

        k += 1
    End While
End Function

```

Public Class Branch

```

Public treeObj As TreeGen
Public trunkLength As Float
Public leafDensity As Integer
Public planetNormal As Vector3
Private created As Boolean = False
Private branch As GameObject
Public startPoint As Vector3
Public endPoint As Vector3
Public branchLength As Float
Public branchWidth As Float
Public branchNormal As Vector3
Public splitsInto As Integer
Public consecutiveSplits As Integer

Public Sub New(_startPos As Vector3, _branchLength As Float, _branchWidth As
Float, parentNormal As Vector3, directionRange As Vector2, _consecutiveSplits As
Integer)
    consecutiveSplits = _consecutiveSplits + 1
    startPoint = _startPos
    branchLength = _branchLength
    branchWidth = _branchWidth
    endPoint = GetBranchEndPoint(parentNormal, directionRange)
    branchNormal = (endPoint - startPoint).normalized
    splitsInto = Random.Range(1, 3)
    branch = New GameObject("Branch of Layer " & consecutiveSplits)
    branch.layer = treeObj.gameObject.layer
    branch.transform.parent = treeObj.root.transform
    branch.transform.localPosition = Vector3.zero
End Sub

Private Function GetBranchEndPoint(parentNormal As Vector3, directionRange As
Vector2) As Vector3
    z As Float

    If consecutiveSplits = 1 Then
        z = Random.Range(Mathf.Cos(Mathf.PI / 12), 1)
    Else
        z = Random.Range(Mathf.Cos(Mathf.PI / 6), Mathf.Cos(Mathf.PI / 3))
    End If

    a As Float = Random.Range(directionRange.x, directionRange.y)
    normalDir As Vector3 = New Vector3(Mathf.Sqrt(1 - z * z) * Mathf.Cos(a),
    Mathf.Sqrt(1 - z * z) * Mathf.Sin(a), z)
    Return startPoint + Quaternion.FromToRotation(Vector3.forward, (planetNormal +
    parentNormal) / 2) * normalDir * branchLength
End Function
End Class

```

Gravity and Collision System

Overview

Each object with a Weight component is assigned a mass and an initial velocity. Their position is updated once every 0.02 seconds in accordance with the force of gravity acting on each weight, due to every other weight. Every planet, moon and sun has a Weight component.

ZeroWeights are massless Weights that experience the force of gravity but do not influence the motion of other Weights. As a class, ZeroWeight inherits from Weight. The player and their spaceship has a ZeroWeight component. ZeroWeights also check for collisions and do not pass collision meshes even if they are forced in that direction by gravity.

For this collision detection system to work, we must specify which movement we want to apply collision detection to, and which movement we want to just apply. For instance, we want to do collision detection on the displacement of the player relative to the planet they are on but ignore the displacement of the player as it moves with the planet round the sun. This is because if we update the planet's position first, the player will be beneath the planet's surface before their position is updated, which will break the collision detection system. So, each update, we want to displace the player and the planet together by the planet's displacement (i.e., without collision detection) and then displace the player by the player's displacement relative to the planet (with collision detection).

Data structures required

Object (instance name if applicable)	Contents (if applicable)
Class Weight	mass as float position as Vector3 velocity as Vector3
Class ZeroWeight : Weight	mass = 0 sigWeight as Weight (The Weight (that is not the sun) which the ZeroWeight is most significantly attracted to, we define the planet/moon the player is "on" as this sigWeight.)

The collision detection system I used I significantly modified to fit my project, but the original code is linked here: <https://github.com/marmitoTH/Unity-Kinematic-Body/blob/master/Assets/Scripts/KinematicBody.cs>

Pseudocode

Pseudocode for the Weight (and ZeroWeight class is below).

```

Public Class Weight

    Protected Static gConst As Float = 1

    Public mass As Float
    Public position As Vector3
    Protected velocity As Vector3

    Public Virtual Function GetAcceleration()

        acceleration As Vector3 = Vector3.zero

        For Each (otherWeight As Weight In FindAll(Weight))
            If otherWeight == This Then
                Continue
            End If
            acceleration += gConst * otherWeight.mass / (otherWeight.position - position).sqrMagnitude * (otherWeight.position - position)

        Next

        Return acceleration
    End Function

    Public Virtual Function Teleport(displacement As Vector3)
        position += displacement
    End Function

End Class

Public Class ZeroWeight : Weight

    Public sigWeight As Weight

    Public ZeroWeight()
        mass = 0
    End

    Public Override Function GetAcceleration()

        acceleration As Vector3 = Vector3.zero

        maxSqrAcc As Float = 0
        sigWeight = Nothing

        For Each (otherWeight As Weight In FindAll(Weight))

            If otherWeight == This Then
                Continue
            End If

            acc As Vector3 = gConst * otherWeight.mass / (otherWeight.position - position).sqrMagnitude * (otherWeight.position - position)

            acceleration += acc
            If acc.sqrMagnitude > maxSqrAcc Then
                maxSqrAcc = acc.sqrMagnitude
                sigWeight = otherWeight
            End If

        Next

        Return acceleration
    End Function

End Class

```

```
Next  
Return acceleration  
End Function  
Public Function MoveRelative(displacement As Vector3)  
    safeDisplacement = KinematicBody.Step(displacement, Me)  
    Teleport(safeDisplacement)  
End Function  
End Class
```

Note that for my game, I have decided that gravitational field strength is inversely proportional to the distance between Weights, as opposed to the inverse square like in reality. This, interestingly, means the velocity required for any orbit of a Weight is a constant, so is independent of the orbital radius. However, I chose this because it stops the force of attraction from quickly being really small to really large, this may confuse the player.

These classes are implemented in the physics update function, pseudocode below:

```
Function PhysicsUpdate()

    displacements As Dictionary(Of Weight, Vector3) = New Dictionary(Of Weight, Vector3)

    For Each (weight As Weight In FindAll(Weight))
        acceleration As Vector3 = weight.GetAcceleration()
        weight.velocity += acceleration * 0.02 'Updates every 0.02 seconds
        displacements.Add(weight, weight.velocity * 0.02)
    Next

    zeroDisplacements As Dictionary(Of ZeroWeight, Vector3) = New Dictionary(Of
ZeroWeight, Vector3)

    For Each (weight As Weight In FindAll(Weight))
        If weight.mass == 0 Then
            zeroWeight As ZeroWeight = weight
            'Teleport zeroWeight by the same displacement as the Weight they are most
attracted towards
            zeroWeight.Teleport(displacements[zeroWeight.sigWeight])
            zeroDisplacements.Add(zeroWeight, displacements[zeroWeight] -
displacements[zeroWeight.sigWeight])
        Else
            weight.Teleport(displacements[weight])
        End If
    Next

    'Teleport with collision detection each ZeroWeight by their displacement relative to
the Weight they are most attracted towards
    For Each (zeroWeight As ZeroWeight In FindAll(ZeroWeight))
        zeroWeight.MoveRelative(zeroDisplacements[zeroWeight])
    Next

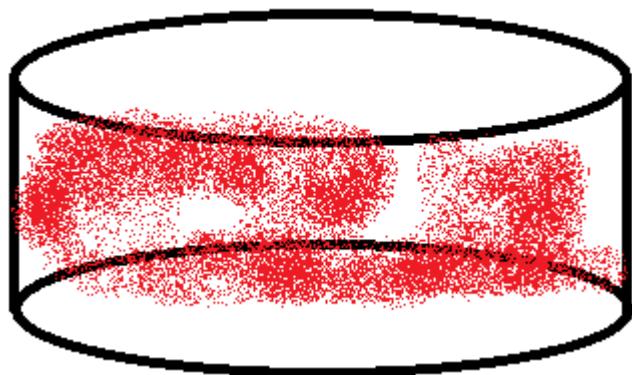
    'Finally move everything such that the camera is close to the origin,
    'to prevent floating point error from jittering the camera
    origin As Vector3 = Camera.position
    For Each (weight As Weight In FindAll(Weight))
        weight.Teleport(-origin)
    Next
End Function
```

Galaxy Generation

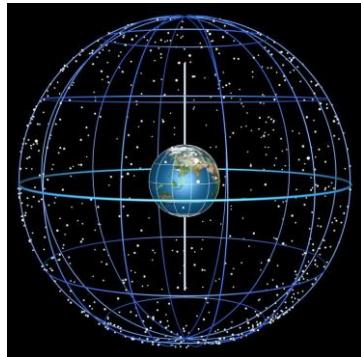
Star Generation

Overview

The player should be able to travel to any visible star. These stars compose a galaxy which takes the shape of a thin cylinder with a spiral pattern.



Because of the camera's clip plane, each star needs to be rendered within about a million units of the camera, for close things like the player's ship to also be visible. In actuality, the galaxy will be much larger than 1 million units. This means we need to project each star onto a celestial sphere surrounding the player and render the sphere instead of the stars.



(Here the earth represents the player)

However, stars on our celestial sphere need to change position and size so when the star is close enough for the star system to generate, the player does not notice the swapping of the star on the sphere with a 3d sun.

Pseudocode

```

Function GetCelestialSphereProjection(actualPosition As Vector3, out
celestialSpherePos As Vector3, out projectedScale As Float)

    'stars are at least rendered by minPixelStarSize pixels, with help from:
https://stackoverflow.com/questions/21648630/radius-of-projected-sphere-in-screen-space
    minVisibleRadiusAt1Dist As Float = minPixelStarSize / Sqrt(Pow(Screen.height * 0.5
/ Tan(0.5 * Camera.fieldOfView), 2) + minPixelStarSize * minPixelStarSize)

    celestialSpherePos As Vector3 = (actualPosition - Camera.position).normalized *
celestialSphereRadius + Camera.position

    sqrStarDist As Float = celestialSpherePos.sqrMagnitude

    horizonRadius As Float = Max(minVisibleRadiusAt1Dist, Sqrt(sqrStarDist -
starRadius * starRadius) * starRadius / sqrStarDist

    projectedScale As Float = 2 * horizonRadius * celestialSphereRadius

End Function

```

Star Pathfinding

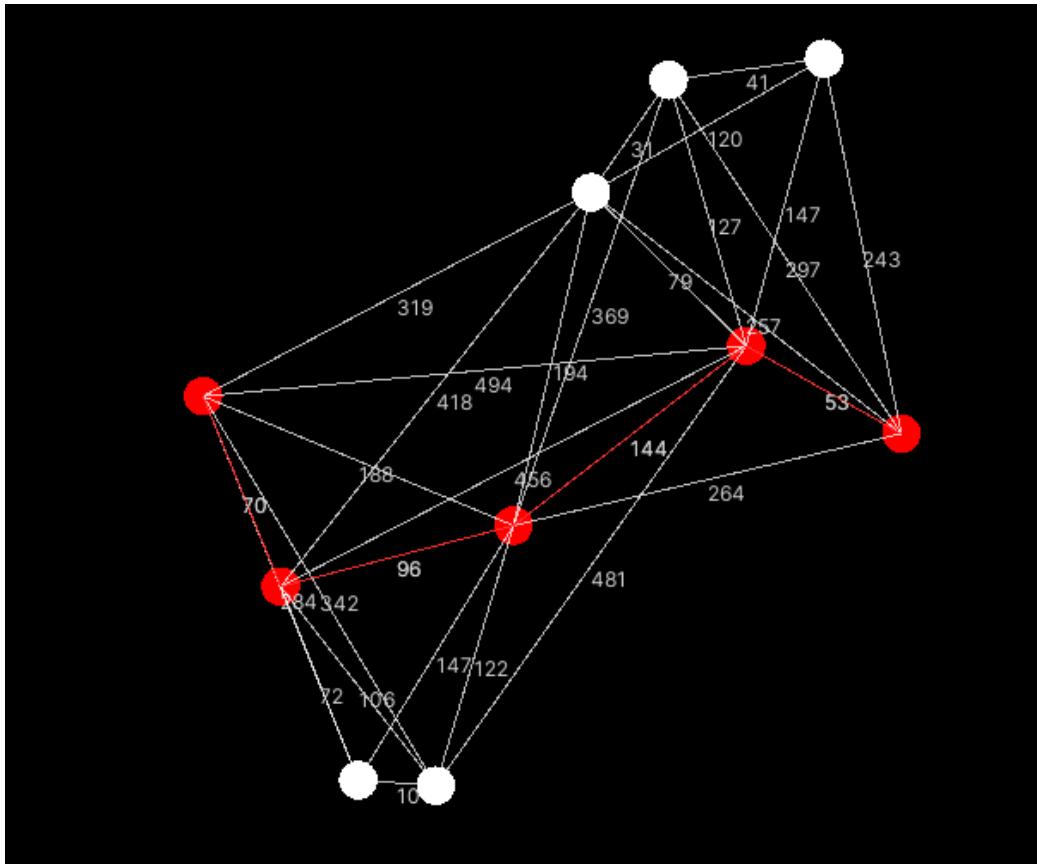
Overview / Pseudocode

Let each star be a node on a graph, and the square of the distance between each pair of stars be the weight of the arc between them. Then, using Dijkstra's algorithm, we can generate routes between stars that maximise visiting as many stars as reasonable along the path. This is important as the ship the player uses to traverse the stars has limited fuel and will need to stop off at multiple stars along their journey to refuel. This "square-distance" method is also preferable to regarding distances above a threshold to be infinity as it ensures the graph will always be connected, and in high star density regions, the square-distance method will still prioritise stops.

Dijkstra's algorithm takes a start node, end node and adjacency matrix as its input, and produces a list of nodes minimising the route length between the start and end nodes as its output. It does this by assigning three variables to each node: flag for if the node has been visited, the current minimum length to node, its parent node (the node imminently preceding this on the minimum route to this). Starting with the path as the start node, each adjacent node to the path updates its minimum current length by comparing each arc between it and the path and adding its weight to the path length up until this arc. The unvisited/adjacent node with the minimum current length value is added to the path, and its parent is set as the node it is linked to on the path by the arc that gave the minimum current length. This process repeats until the end node is appended to the path, giving the shortest route between the start and end nodes.

Visuals

The red path is the shortest path between the leftmost and rightmost nodes in the graph, generated using Dijkstra's algorithm:



Data structures required

Object (instance name if applicable)	Contents (if applicable)
Struct DijkstraRow	visited as bool currentLength as float parent as DijkstraRow (Each Dijkstra row corresponds to a star [like in an adjacency matrix])
adjMatrix as float[,]	Adjacency matrix of square distances between every pair of stars

Hive Generation

Hive generation

Overview

The Hive is an office labyrinth accessed by riding a lift down below a tower on a planet. The hive can be utilised as a short-cut between these towers. Below I overview how a Hive can be procedurally generated for use in my game:



Starting at the start tile a new tile is instantiated adjacent to the previous tile. It is more likely to be in the direction the previous tile was instantiated in, so the corridors are straight. This continues n times. Then a new start tile is randomly chosen from the instantiated tiles, from which another "corridor" (now blue in this example) is developed.

Each tile has 6 walls one for each of the positive and negative directions of the x, y & z axis. Let us say the front wall is the positive z wall and the back wall is the negative z wall. If a new tile is instantiated in front of the previous, then the back wall of the new tile and the front wall of the previous tile are removed so that the player can walk from one to the other.

After the corridors are made, all walls between two instantiated tiles are removed (in our example this makes the end of the grey corridor form a loop).

Finally, I have included custom tiles such as stairs or environment specific rooms that are sometimes instantiated instead of tiles when making the corridor, to make the scene more interesting. These tiles take priority over regular tiles, and remove any within their bounding box so that there is no overlap of tiles.

Pseudocode

```

'Run for each corridor in the hive
Function AddCorridor(pos As Vector3Int, dir As Vector3Int, tile As Tile, length As
Integer)
    'If the desired corridor length is reached
    If length = 0 Then Return

    'Random vector with prob of being in direction dir else random direction
    perpendicular to dir
    dir = RandomBiasDir(dir, prob)
    new_tile As Tile = NewTileInDir(dir, pos)

    pos += dir
    'The function is called recursively until the desired length is reached
    AddCorridor(pos, dir, new_tile, length - 1)
End Function

```

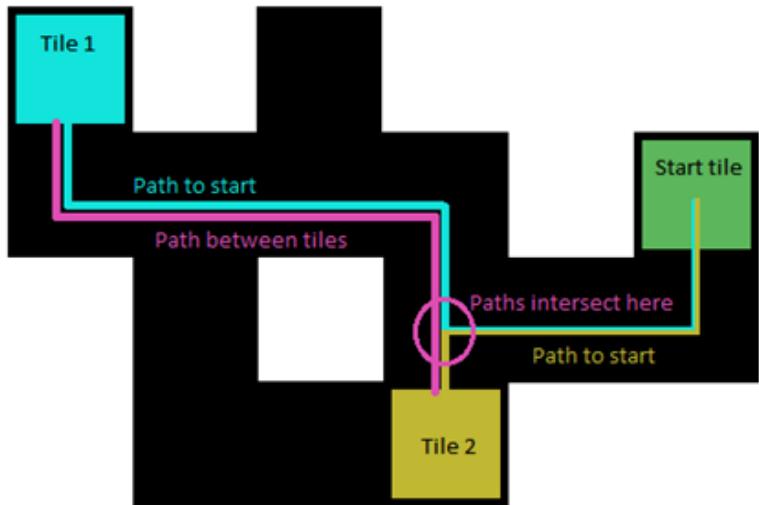
Data structures required

Object (instance name if applicable)	Contents (if applicable)
tileMap as Dictionary<Vector3Int, Tile>	Dictionary of tile objects and their positions, this defines the hive generated
customTileMap as Dictionary<Vector3Int, CustomTile>	Dictionary of tile objects and their positions, each null Tile in tileMap signals the program to refer to the customTileMap and the CustomTile which encompasses this Tile
Tile	
CustomTile	

Pathfinding in the hive

Overview

I want robots (NPCs) to be able to roam around any procedurally generated office. To achieve this, I can take advantage of the parent-child relationship between the tiles (each tile [child] is instantiated adjacent to another tile [parent]) that is a by-product of the way the “Hive” is generated. This means that for any tile in the “Hive” if I get the parent of this tile, and the parent of this parent and so on, I will always get to the first tile generated (in the image the Start tile). Therefore, I can get a path to follow from any tile to the Start tile. If I want to get a path between any two tiles, I can get the paths from the two tiles to the Start tile and find the point closest to the tiles along those paths at which they intersect then link the paths together like in the image below:



It is worth noting that this algorithm does not necessarily find the shortest path, but that is not a requirement for my problem as the NPCs in the Hive, unless there is an unobstructed straight path to the player (and in which case they may beeline for the player), simply roam around.

Pseudocode

```

'This is an oversimplification.
'A function is required to get the start and end points as tiles from world vectors
(scaled, translated, rotated)
'A function is required to get the path through a custom tile
'And there will be lots of edge cases & errors to handle
Function List(Of Vector3Int) PathBetween(start As Vector3Int, End As Vector3Int)

    startToOrigin As List(Of Vector3Int) = PathToOrigin(start)
    endToOrigin As List(Of Vector3Int) = PathToOrigin(start)

    startToEnd As List(Of Vector3Int) = New List(Of Vector3Int)
    intersectIndex As Integer = 0
    For Each (Vector3Int key In startToOrigin)
        If (endToOrigin.Contains(key)) Then
            intersectIndex = endToOrigin.IndexOf(key)
            Break
        End If
        startToEnd.Add(key)
    Next
    For i = endToOrigin.Count - 2 To intersectIndex Step -1
        startToEnd.Add(endToOrigin[i])
    Next

    Return startToEnd

End Function

Function List(Of Vector3Int) PathToOrigin(start As Vector3Int)
    startToOrigin As List(Of Vector3Int) = New List(Of Vector3Int)
    parent As Vector3Int = start
    flag As Boolean = True

    While flag
        startToOrigin.Add(tileMap[parent])

        If parent == tileMap[parent].key Then 'true iff at origin
            flag = False
        End If

        parent = tileMap[parent].key
    End While

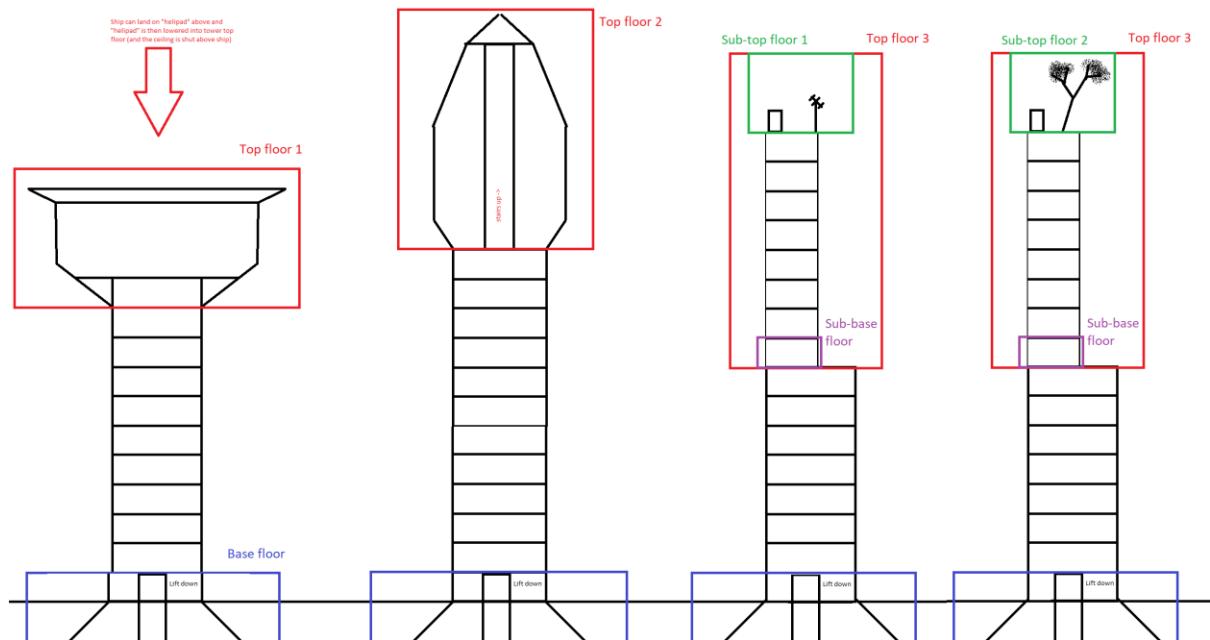
    Return startToOrigin
End Function

```

Tower generation

Overview / Visuals

The player enters a hive by taking the lift down inside a tower. Each tower is composed of layers of pre-generated flats. Below are examples of towers (may need to zoom in)



Every generated tower consists of a base floor (there is only one base floor), middle floors (there is only one middle floor) and a top floor (there are four top floors as illustrated). To differentiate the towers, each has a random number of middle floors within a range. Moreover, each floor is rotated $90^\circ \{n \in \mathbb{Z}\}$ and has a random arrangement of windows, lights and “props”.

Each tile (like in Hive generation) has a random probability of spawning a random prop on top of it. These include: 3 variants of boxes, 2 variants of papers, an upright plank, and a shelf.

Shelf

Each shelf has either a lower level, upper level, or both. If the shelf has a lower level, a random support (3 variants) is attached to the lower shelf (to keep it standing). Else if the shelf has an upper layer, a random support (3 different variants) is attached to the upper shelf. On the top level of the shelf, there is a random probability that a sitting robot, computer or a set of divisions with an additional top layer will spawn on it. If there is a bottom level of the shelf, a random set of divisions (4 variants) is added to link it to the top.

Top floor 1

Top floor 1 has a helipad that the player can land their ship on and flip a switch to lower the helipad into the tower, with the roof closing above them.

Sub-top floors

These top floors are smaller mini towers on top of the main tower. Their generation is identical to the “larger” tower generation, but each floor is smaller in area.

Pseudocode

Below is pseudocode for generating the tower:

```
Public Function GenerateTower()

    stories = Random.Range(5, 15)

    topPrefab = If(stories < 10, topPrefabs(0), topPrefabs(Random.Range(0,
    topPrefabs.Length))) 'Only add Top floor 1 / Top floor 2 if tower tall enough

    prevRot As Integer = startRot
    flats = New List(Of Flat)()

    For i As Integer = 0 To stories - 1
        rot As Integer

        'Don't want the same rotation of the flat prefab twice in a row
        Do
            rot = Random.Range(0, 4) * 90
        Loop While rot = prevRot

        prefab As GameObject = If(i = 0, basePrefab, If(i = stories - 1, topPrefab,
        flatPrefab))

        'We care about normal up because the tower is generated on a spherical planet
        flat As Flat = Instantiate(FlatPrefab, position = i * 5 * normalUp, rotation =
        Quaternion.LookRotation(Vector3.Cross(normalUp, Vector3.forward), normalUp) *
        Quaternion.Euler(0, rot, 0)).GetComponent(Of Flat)()

        'These functions will mostly consist of instantiating prefabrications
        flat.GenerateFlat()
        flat.GenerateProps()

        flats.Add(flat)

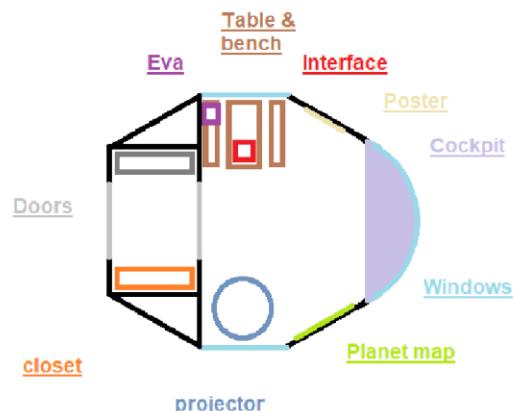
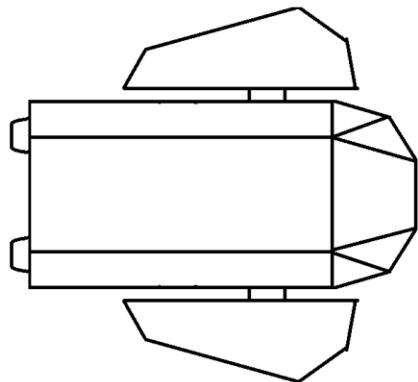
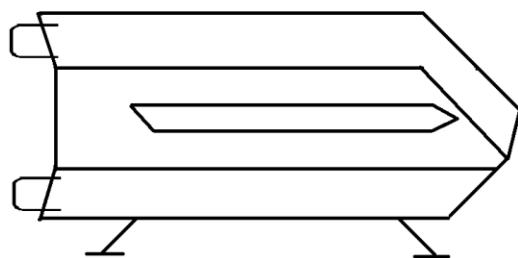
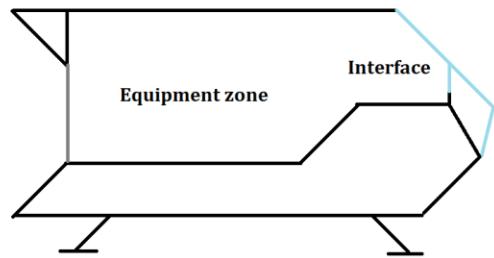
        prevRot = rot
    Next
End Function
```

Ship Design

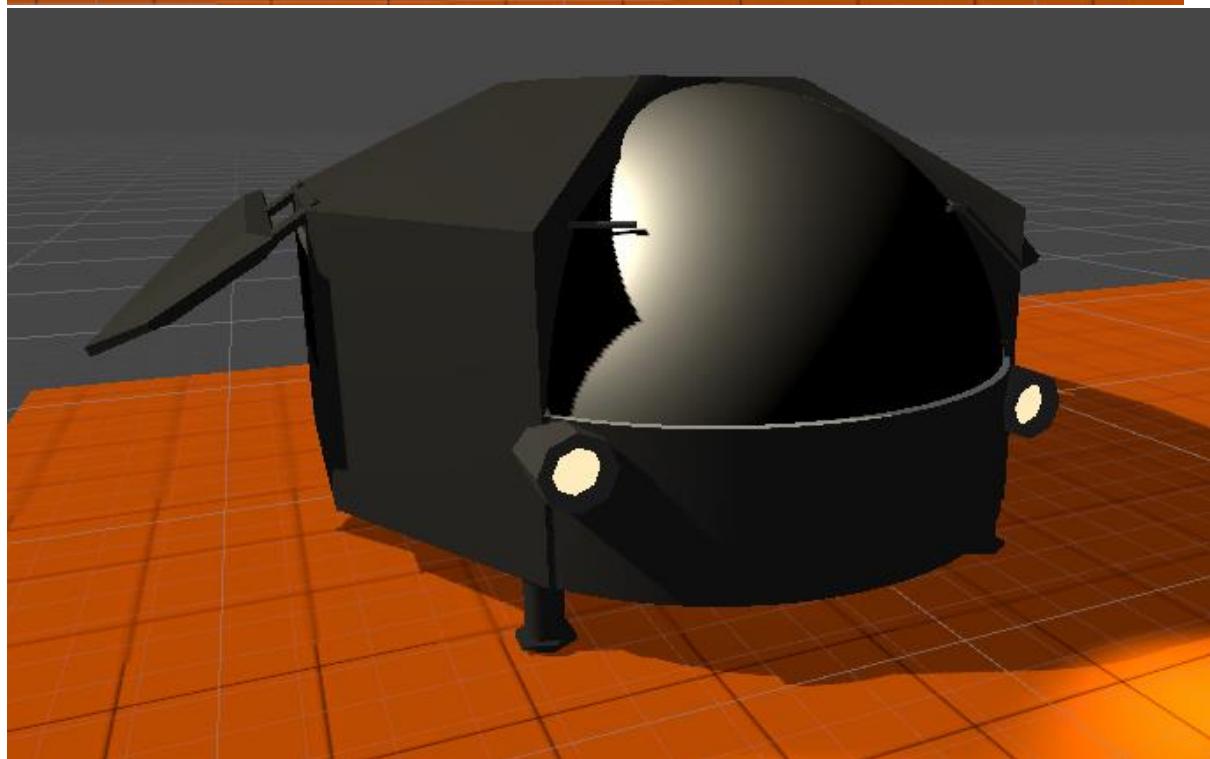
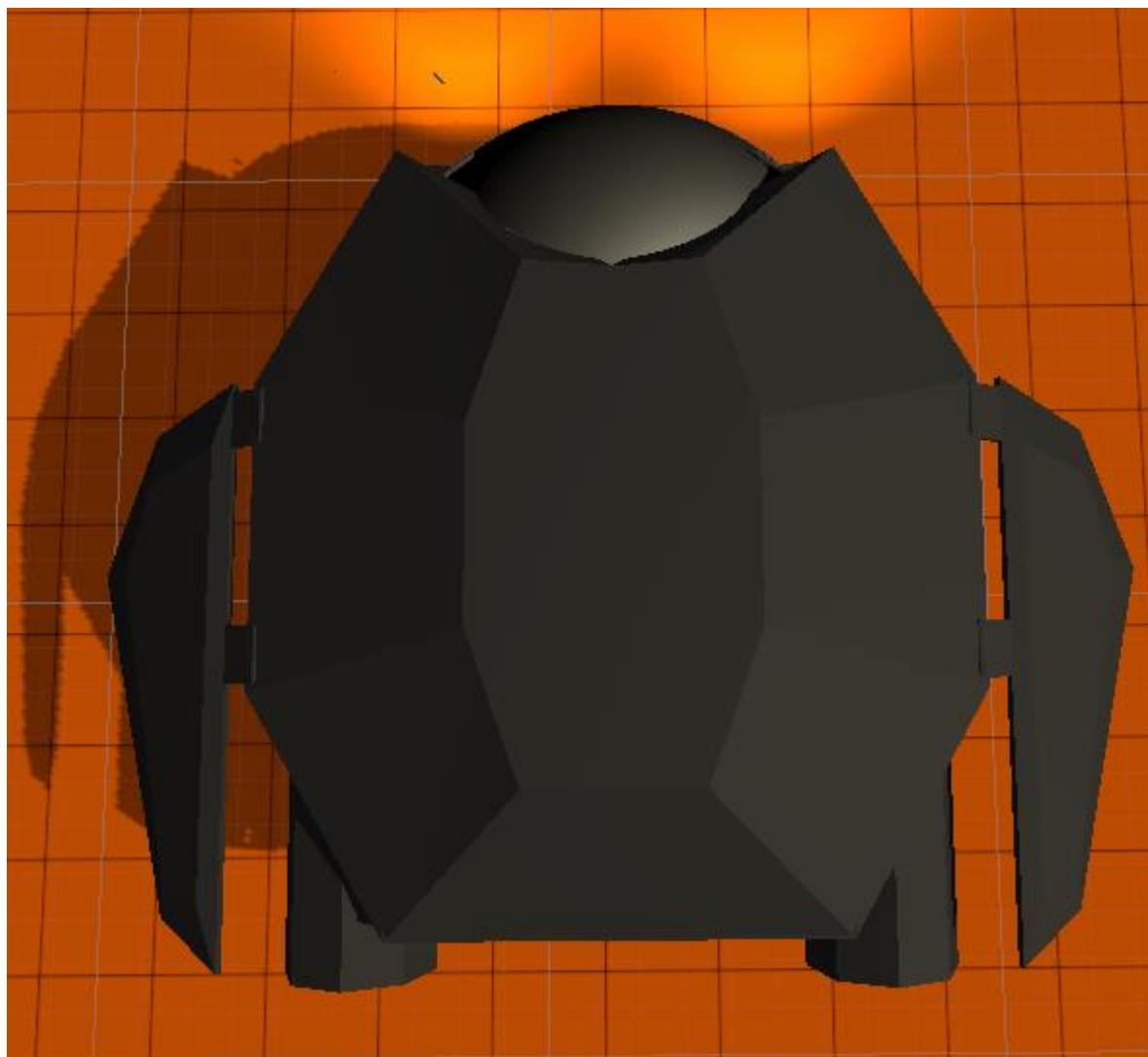
Visuals

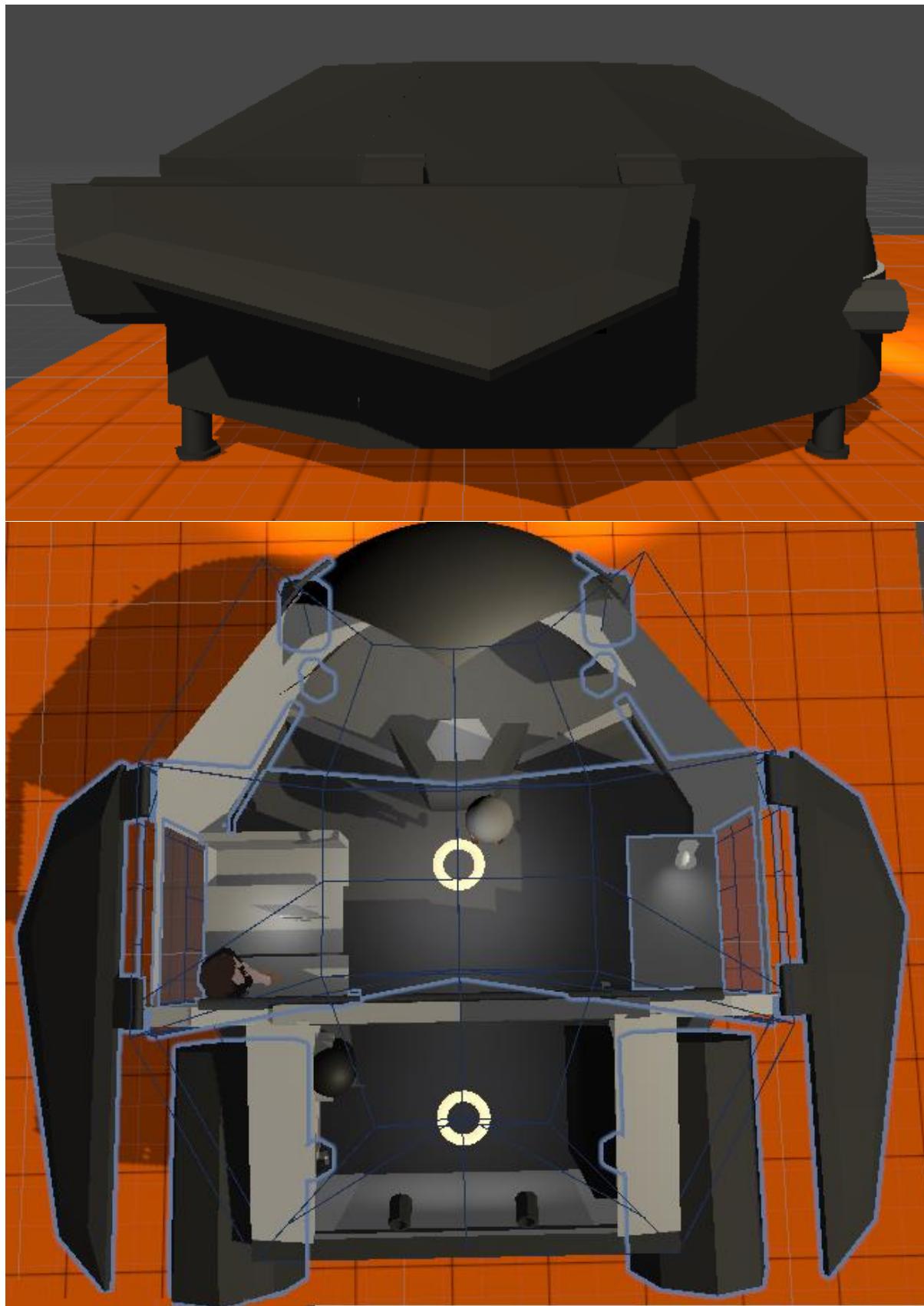
The player uses a ship to fly between planets and star systems.

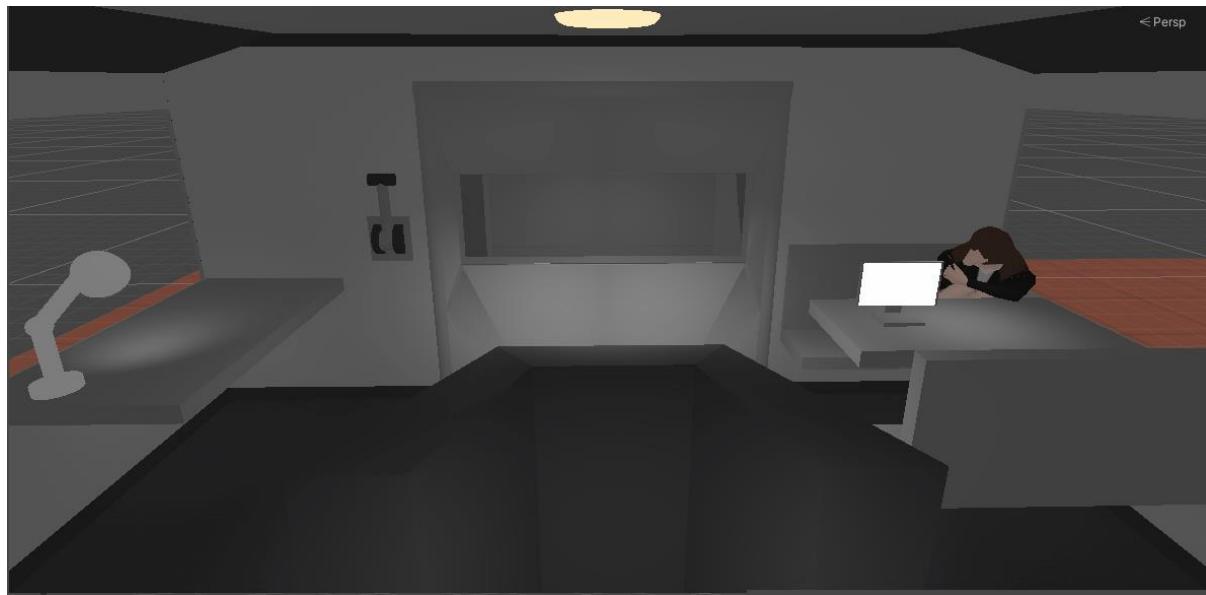
Below are two plans for how the ship will appear in game:



Below is the implemented ship model that will be used in the game (I think it ended up looking like a turtle):







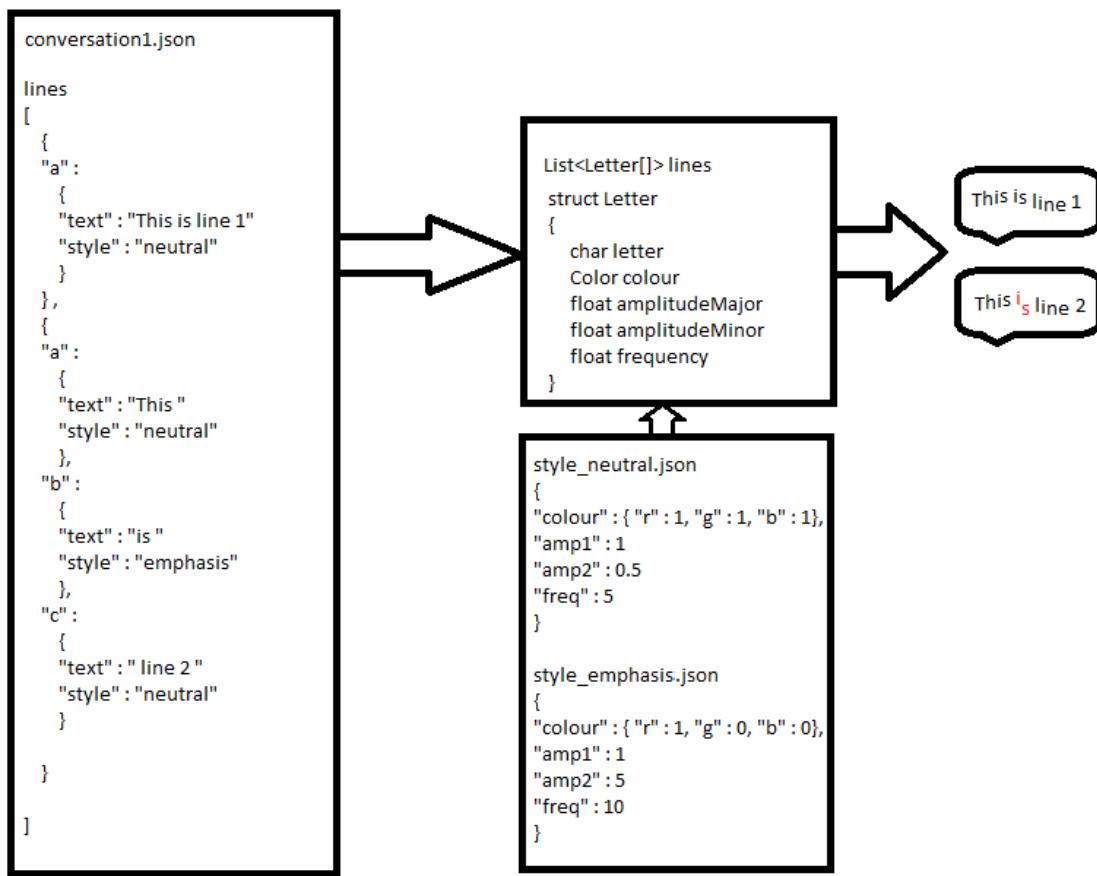
Dialogue System

Overview

Characters in my game will be able to communicate with the player via text rendered in world space. This text should be visually appealing, readable in all backgrounds and animated.

The player can have *conversations* with NPCs. These *conversations* consist of multiple *lines* of text. Text in each line fades in from below letter by letter. Once the *line* has been read, the entire line fades out above. And the next line appears as described. Each *line* consists of up to three consecutive *styles* of text. Letters of a *style* oscillate (up and down) at certain frequency with a certain amplitude and are a certain colour. For instance, using these styles will allow “Hello there” oscillate like a Mexican wave. This adds personality (hopefully) to the dialogue.

Information on each conversation is stored in its own json file. The file is loaded and processed before the conversation begins. This makes it very easy to add conversations to my game.

VisualsData structures required

Object (instance name if applicable)	Contents (if applicable)
conversation as <code>List<Letter[]></code>	A list of letters composing each line of dialogue in a conversation
Struct <code>Letter</code>	Contains information on how and when to render and animate a letter in a line. delay as <code>float</code> letter as <code>char</code> colour as <code>Color</code> amplitudeMajor as <code>float</code> amplitudeMinor as <code>float</code> fadeVelocity as <code>float</code> fadeTime as <code>float</code>
Struct <code>Text</code>	Contains a string of text and in which style it should be rendered. text as <code>string</code> style as <code>string</code>

Struct Line	Contains at most three unique Texts and specifies the startup delay length before rendering them. startup as float a as Text b as Text c as Text
--------------------	---

Pseudocode

Below is pseudocode for loading a conversation from a json file:

```

Public Class Dialogue
    Public lines As Line()

    Public Structure Line
        Public startup As Float
        Public a As Text
        Public b As Text
        Public c As Text
    End Structure

    Public Structure Text
        Public text As String
        Public style As String
    End Structure

    Public Structure Letter
        Public character As String
        Public delay As Float
        Public colour As Color
        Public freqSpeedAmpAmp As Vector4
        Public fadeVelTime As Vector2

        Public Sub New(textAbstract As Text)
            asset As TextAsset = TryCast(Resources.Load(textAbstract.style), TextAsset)
            json As String = asset.text
            detail As Letter = JsonUtility.FromJson(Of Letter)(json)
            character = Nothing
            delay = detail.delay
            colour = detail.colour
            freqSpeedAmpAmp = detail.freqSpeedAmpAmp
            fadeVelTime = detail.fadeVelTime
        End Sub

        Public Sub New(_character As Char, detail As Letter, addDelay As Float)
            character = _character.ToString()
            delay = detail.delay + addDelay
            colour = detail.colour
            freqSpeedAmpAmp = detail.freqSpeedAmpAmp
            fadeVelTime = detail.fadeVelTime
        End Sub
    End Structure

```

```
Public Shared Function Load(filename As String) As Dialogue
    Return OpenJson(Of Dialogue)(filename)
End Function

Public Function GetLetters() As List(Of Letter())
    letters As List(Of Letter()) = New List(Of Letter())()

    For Each line As Line In lines
        sentence As Letter() = New Letter(line.a.text.Length + line.b.text.Length
+ line.c.text.Length - 1)
        aDetail As Letter = New Letter(line.a)
        bDetail As Letter = New Letter(line.b)
        cDetail As Letter = New Letter(line.c)
        i As Integer = 0
        addDelay As Single = 0

        For Each character As Char In line.a.text
            sentence(i) = New Letter(character, aDetail, addDelay)
            i += 1
        Next

        addDelay += aDetail.delay

        If Not line.b.Equals(Nothing) Then

            For Each character As Char In line.b.text
                sentence(i) = New Letter(character, bDetail, addDelay)
                i += 1
            Next
        End If

        addDelay += bDetail.delay

        If Not line.c.Equals(Nothing) Then

            For Each character As Char In line.c.text
                sentence(i) = New Letter(character, cDetail, addDelay)
                i += 1
            Next
        End If

        letters.Add(sentence)
    Next

    Return letters
End Function
End Class
```


Technical solution

Each script that is used during gameplay is listed below, except shader graphs which have been omitted. You can (should you wish) access the shader graphs, editor scripts and more information about how the scripts work such as what each serialized class attribute is set to and how the transform hierarchy is organised via this link: [https://drive.google.com/file/d/1oI87QhJOhyi-VwbvzJz6Un_0N3zn8ami/view?usp=share_link] (the download link to the entire project in Unity 2020.3.18f1). Additionally, if you want to play Third Law for yourself, here is a link to download the latest build of the game: [https://drive.google.com/file/d/1EdFMmq5TrtnT8Pou6lw6uA4qJpOZqz9J/view?usp=share_link]. Finally, here is a link to specifically download the json files written for the dialogue system (referenced in the Testing objective 4 section): [https://drive.google.com/file/d/1I97mrFQgXG02r8xBt-Blfh2UHq8QZ6-x/view?usp=share_link].

Android.cs

```

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

public class Android : MonoBehaviour
{
    [SerializeField] private AndroidVision androidVision;
    [SerializeField] private FabrikLeg[] legs;
    [SerializeField] private Transform Chest;
    private RoboEyes roboEyes;
    public float height;

    [SerializeField] private float moveSpeed = 5;
    private Rigidbody rb;
    private List<Vector3> path;
    private HiveGen hiveGen;
    private Vector3 prevHivePos;
    private int framesDetected = 0;

    // Start is called before the first frame update
    void Start()
    {
        roboEyes = new RoboEyes(androidVision);

        hiveGen = FindObjectOfType<HiveGen>();
        rb = GetComponent<Rigidbody>();

        transform.parent = FindObjectOfType<HiveGen>().transform;

        androidVision.Eyes[0].GetChild(0).GetComponent<MeshRenderer>().sharedMaterial =
new Material(androidVision.Eyes[0].GetChild(0).GetComponent<MeshRenderer>().sharedMaterial);
        androidVision.Eyes[1].GetChild(0).GetComponent<MeshRenderer>().sharedMaterial =
new Material(androidVision.Eyes[1].GetChild(0).GetComponent<MeshRenderer>().sharedMaterial);
    }

    // Update is called once per frame
    void Update()
    {
        if (CameraState.isPaused)
            return;

        CameraState.isDead |= (Camera.main.transform.position -
transform.position).sqrMagnitude < 4;

        roboEyes.SetEyeColour(Color.yellow);

        Vector3 position = transform.position;

        //If the Android sees the player
        if (Physics.Raycast(position, Camera.main.transform.position - position, out RaycastHit hitInfo) && Camera.main.transform.IsChildOf(hitInfo.collider.transform))
        {
            framesDetected++;
            roboEyes.SetEyeColour(Color.Lerp(Color.yellow, Color.red, framesDetected *
Time.smoothDeltaTime));
        }
    }
}

```

```

        else
            framesDetected = 0;

        //If the Android has seen the player for more than one second, the Android
moves towards the Camera's position
        if (framesDetected * Time.smoothDeltaTime > 1)
        {
            MoveTowards(Camera.main.transform.position);

            Debug.DrawLine(position, Camera.main.transform.position -
Camera.main.transform.up, Color.green, Time.smoothDeltaTime);
        }
        //If there is a valid path from the Android to the player, the Android moves
towards the next point on the path
        else if (hiveGen.PathBetweenPoints(rb.position,
Camera.main.transform.position, out path) && path.Count > 1)
        {
            Vector3 nextPointOnPath = path[0];
            for (int i = 1; i < path.Count; i++)
            {
                if (Physics.Raycast(position, path[i] - position, out RaycastHit
groundInfo) && groundInfo.collider.CompareTag("Ground") &&
(groundInfo.collider.transform.position - path[i]).sqrMagnitude < 25)
                    nextPointOnPath = path[i];
            }

            MoveTowards(nextPointOnPath);

            Debug.DrawRay(rb.position, nextPointOnPath - position, Color.blue,
Time.smoothDeltaTime);
        }

        //Update the Android's legs based on the movement of the hive (which moves
relative to the player)
        Vector3 displacement = hiveGen.transform.position - prevHivePos;
        prevHivePos = hiveGen.transform.position;
        foreach (FabrikLeg leg in legs)
        {
            leg.deltaPos = transform.position - position - displacement;
            leg.prevTarget += displacement;
            leg.midTarget += displacement;
            leg.endTarget += displacement;
        }
    }

    void MoveTowards(Vector3 point)
    {
        androidVision.LookTowards(point);

        if (Physics.Raycast(transform.position, -hiveGen.transform.up, out RaycastHit
groundInfo) && groundInfo.collider.CompareTag("Ground"))
            transform.position = groundInfo.point + hiveGen.transform.up * height;
        transform.position -= Chest.forward * Time.deltaTime * (roboEyes.colour !=
Color.yellow ? moveSpeed : 5);
    }
}

```

AndroidVision.cs

```

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

public class AndroidVision : MonoBehaviour
{
    [SerializeField] private Transform Arms;
    [SerializeField] private Transform Chest;
    [SerializeField] private Transform Head;

    public Transform[] Eyes;

    [SerializeField] private float inertia = 100;

    private Vector3 up;

    private void Start()
    {
        up = FindObjectOfType<HiveGen>().transform.up;
    }

    public void LookTowards(Vector3 target)
    {
        //Calculate the rotation to face the target position.
        Quaternion rotation = Quaternion.LookRotation(-(target - transform.position).normalized, up);
        Quaternion flatRotation = Quaternion.LookRotation(Vector3.ProjectOnPlane(-(target - transform.position).normalized, up), up);

        Chest.rotation = Quaternion.Slerp(Chest.rotation, flatRotation, Time.deltaTime / inertia);

        Arms.rotation = Quaternion.Slerp(Arms.rotation, Chest.rotation, Time.deltaTime / inertia);

        Head.rotation = Quaternion.Slerp(Head.rotation, rotation, Time.deltaTime / inertia);

        Head.localRotation = Quaternion.Euler(Mathf.LerpAngle(0, Head.localEulerAngles.x, 0.1f), Head.localEulerAngles.y, Head.localEulerAngles.z);

        //Eyes[0].rotation = rotation * Quaternion.Euler(90, 0, 0);
        //Eyes[1].rotation = rotation * Quaternion.Euler(90, 0, 0);
    }
}

```

Atmosphere.shader

```

Shader "Custom/Atmosphere"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}

    }
    SubShader
    {
        // No culling or depth
        Cull Off ZWrite Off ZTest Always

        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"
            #include "Math.cginc"
            #include "PerlinNoise.cginc"
            //

            struct appdata {
                float4 vertex : POSITION;
                float4 uv : TEXCOORD0;
            };

            struct v2f {
                float4 pos : SV_POSITION;

                float2 uv : TEXCOORD0;
                float3 viewVector : TEXCOORD1;
            };

            v2f vert (appdata v) {
                v2f output;
                output.pos = UnityObjectToClipPos(v.vertex);

                output.uv = v.uv;
                // Camera space matches OpenGL convention where cam
forward is -z. In unity forward is positive z.
                // (https://docs.unity3d.com/ScriptReference/Camera-cameraToWorldMatrix.html)
                float3 viewVector = mul(unity_CameraInvProjection,
float4(v.uv.xy * 2 - 1, 0, -1));
                output.viewVector = mul(unity_CameraToWorld,
float4(viewVector,0));
            }

            float4 frag (v2f i) : SV_Target
            {
                float4 col = tex2D(_MainTex, i.uv);
                float3 viewDir = normalize(i.viewVector);
                float3 lightDir = normalize(-i.viewVector);
                float3 normal = normalize(i.normal);
                float3 ambient = col.rgb * 0.1;
                float3 diffuse = col.rgb * max(0, dot(normal, lightDir));
                float3 specular = pow(max(0, dot(reflect(-viewDir, normal), lightDir)), 10);
                float3 finalColor = ambient + diffuse + specular;
                return float4(finalColor, 1);
            }
        }
    }
}

```

```
        return output;
    }

    float2 squareUV(float2 uv) {
        float width = _ScreenParams.x;
        float height = _ScreenParams.y;
        //float minDim = min(width, height);
        float scale = 1000;
        float x = uv.x * width;
        float y = uv.y * height;
        return float2 (x/scale, y/scale);
    }

    sampler2D _MainTex;
    sampler2D _CameraDepthTexture;
    sampler2D _CameraDepthNormalsTexture;

    sampler2D fogColourRings;
    sampler2D BlueNoise;

    float3 sunCentre;
    float3 planetCentre;
    float atmosphereRadius;
    float oceanRadius;
    float planetRadius;

    float fogRange;

    float noiseFreq;
    float noiseAmp;

    float windSpeed;

    float ditherScale;
    float ditherStrength;

    float dispersionPower;
    float dispersionScale;
    float density;

    sampler2D waveNormalA;
    sampler2D waveNormalB;
    float4 oceanColour;
    float visibleDepth;
    float waveSpeed;
    float waveStrength;
    float waveNormalScale;
```

```

    float smoothness;

    float roughBed;

    float numDiscs;
    float rings;
    float2 iris1;
    float2 iris2;
    float3 disc1Normal;
    float3 disc2Normal;
    float4 disc1Colour;
    float4 disc2Colour;

    float4 discColour;

    float numInScatteringPoints;
    float atmosphereDensity;
    float opticalDensity;
    float lightIntensity;

    float4 DrawRings(float4 discNColour, float distance01, float
distanceToRing, float3 intersectPoint)
    {
        //Calculate the overall opacity of the ring based on the two
noise values that determine the ring spacing
        float coarseNoise =
Unity_GradientNoise_float(float2(distance01, distance01), rings / 10);
        float fineNoise = Unity_GradientNoise_float(float2(distance01,
distance01), rings);
        float alpha = clamp(pow(sin(1.571429 * coarseNoise), 5) *
fineNoise, 0, 1);
        alpha = alpha * clamp(remap01(5, 100, distanceToRing), 0, 1);

        float4 ringColour = discNColour * remap01(1, -1,
dot(normalize(intersectPoint - planetCentre), normalize(intersectPoint -
sunCentre)));
        ringColour.a = alpha;
        return ringColour;
    }

    float GenerateRings(float3 rayOrigin, float3 rayDir, float
surfaceDist, float dstToAtmosphere)
    {
        discColour = float4(0, 0, 0, 0);

        float3 intersectPoint;
        float distanceToRing = surfaceDist + 1;
        float ringDistToCentre = 0;

```

```

        float distance01 = -1;

        if (numDiscs > 0)
        {
            //Calculate the intersection time with the first disc
            float intersectTime = dot(disc1Normal, planetCentre -
rayOrigin) / dot(disc1Normal, rayDir);

                //If the intersection is in front of the camera, calculate
the intersect point and ring colour to draw
                if (intersectTime > 0)
                {
                    intersectPoint = rayOrigin + rayDir * intersectTime;

                    ringDistToCentre = length(intersectPoint -
planetCentre);

                    distance01 = ringDistToCentre / (planetRadius * 5);

                    if (distance01 > iris1.x && distance01 < iris1.y)
                    {
                        distanceToRing = length(intersectPoint -
rayOrigin);
                        if (distanceToRing < surfaceDist)
                            discColour = DrawRings(disc1Colour,
distance01, distanceToRing, intersectPoint);
                    }
                }
            }

            if (numDiscs > 1)
            {
                float intersectTime = dot(disc2Normal, planetCentre -
rayOrigin) / dot(disc2Normal, rayDir);

                if (intersectTime > 0)
                {
                    float3 newIntersectPoint = rayOrigin + rayDir *
intersectTime;
                    float newRingDist = length(newIntersectPoint -
planetCentre);
                    float newDist = length(newIntersectPoint - rayOrigin);
                    float newdist01 = newRingDist / (planetRadius * 5);

                    if (newDist < surfaceDist && newdist01 > iris2.x &&
newdist01 < iris2.y)
                    {
                        float4 newDiscColour = DrawRings(disc2Colour,
newdist01, newDist, newIntersectPoint);
                    }
                }
            }
        }
    }
}

```

```

                //If the disc is closer than the previously drawn
disc or there are no discs drawn yet, draw the new disc
                if (discColour.a == 0 || distanceToRing >
dstToAtmosphere)
                    discColour = newDiscColour;
                // Otherwise, blend the two discs together
                else if (newDist < dstToAtmosphere ||

dstToAtmosphere == 0)
{
                float geoMean = pow(newDiscColour.a *
discColour.a, 0.5);
                float t = 0.5;
                if (newDiscColour.a != discColour.a)
                    t = abs((newDiscColour.a - geoMean) /
(newDiscColour.a - discColour.a));

                discColour = lerp(discColour, newDiscColour,
t);
                discColour.a = max(newDiscColour.a,
discColour.a);
}
                if (newDist < distanceToRing)
{
                intersectPoint = newIntersectPoint;
                ringDistToCentre = newRingDist;
                distanceToRing = newDist;

                distance01 = ringDistToCentre / (planetRadius
* 5);
}
}
}
}

}
//Return 0 if no discs were drawn, 1 if a disc was drawn in
front of the atmosphere and 2 is discs were only drawn behind the atmosphere
if (discColour.a == 0)
    return 0;
else if (distanceToRing > dstToAtmosphere)
    return 1;
return 2;
}

float reflection01(float3 normal, float3 position)
{
    return dot(normal, normalize(sunCentre - position)) * 2 - 1;
}

```

```

    }

        float4 GenerateOcean(float4 originalCol, float3 clipPlanePos,
float3 rayDir, float sceneDepth, float dstToOcean, float dstThroughOcean,
float3 oceanPos, float oceanNoise, float4 fogColour)
    {
        if (dstToOcean < sceneDepth && dstThroughOcean > 0)
        {
            oceanColour = normalize(oceanColour) * 0.1;

            waveSpeed *= windSpeed;

                //Calculate the distance above water by subtracting the
ocean radius from the distance between the clip plane position and the planet
center
            float dstAboveWater = length(clipPlanePos - planetCentre)
- oceanRadius;

            float4 finalColour = lerp(originalCol, oceanColour,
clamp((sceneDepth - dstToOcean) / 10, 0, 1));
                //This adds our lip lining shore at depths of 5 units or
less
            finalColour = lerp(oceanColour, finalColour,
clamp((sceneDepth - dstToOcean) / (5 - oceanNoise), 0, 1));

            ///START OF NOT MY CODE

            float3 oceanSphereNormal = normalize(oceanPos);

                //Calculate two wave offsets based on the time and wave
speed, and use them to calculate the normal vector of the waves using
triplanar mapping
            float2 waveOffsetA = float2(_Time.x * waveSpeed, _Time.x *
waveSpeed * 0.8);
            float2 waveOffsetB = float2(_Time.x * waveSpeed * -0.8,
_Time.x * waveSpeed * -0.3);
            float3 waveNormal = triplanarNormal(oceanPos,
oceanSphereNormal, waveNormalScale / planetRadius, waveOffsetA, waveNormalA);
            waveNormal = triplanarNormal(oceanPos, waveNormal, 1 /
waveNormalScale, waveOffsetB, waveNormalB);
            waveNormal = normalize(lerp(oceanSphereNormal, waveNormal,
waveStrength));
            //return float4(oceanNormal * .5 + .5,1);
            float diffuseLighting = saturate(dot(oceanSphereNormal,
normalize(sunCentre - planetCentre)));
            float specularAngle = lerp(waveNormal,
acos(dot(normalize(normalize(sunCentre - planetCentre) - rayDir),
waveNormal)), 0.2);
        }
    }
}

```

```

        float specularExponent = specularAngle / (1 - smoothness);
        float specularHighlight = exp(-specularExponent * 
specularExponent);

                finalColour *= diffuseLighting;
                finalColour += specularHighlight * lerp(finalColour,
oceanColour, 0.5);

                finalColour = finalColour * (0.75 +
reflection01(waveNormal, oceanPos) * 0.25);

                finalColour = float4(min(finalColour.r, 1),
min(finalColour.g, 1), min(finalColour.b, 1), 1);

                ///END OF NOT MY CODE

            return finalColour;
        }
        return originalCol;
    }

    float densityAtPoint(float3 samplePoint)
{
    return remap01(0, atmosphereRadius - planetRadius,
length(samplePoint - planetCentre) - planetRadius);
}
    float opticalDepth(float3 samplePoint, float spread1, float
spread2)
{
    return remap01(0, 2 * planetRadius * spread1,
raySphere(planetCentre, planetRadius * spread2, samplePoint,
normalize(sunCentre - samplePoint)).y);
}

    //This function calculates the density and optical density of the
atmosphere in a given direction,
    //starting from a given point and stepping in that direction at a
given step size, with some added noise.
    void densityInDir(float3 enterPoint, float3 stepDir, float
stepSize, float blueNoise)
{
    float angle = 0;
    opticalDensity = 0;
    atmosphereDensity = 0;

    //Loop over a set number of scattering points to calculate the
density and optical density.
    for (int i = 0; i < numInScatteringPoints; i++)

```

```

    {
        float3 samplePoint = enterPoint + stepDir * stepSize * i;
        float3 sampleNormal = normalize(samplePoint - sunCentre);

        //Calculate the atmosphere density at the current
scattering point
        atmosphereDensity += pow(densityAtPoint(samplePoint),
density) + blueNoise;

        angle -= dot(stepDir, sampleNormal);
        opticalDensity += lerp(0, planetRadius * 3.54,
opticalDepth(samplePoint, 1.77, 1.77)) + blueNoise;
    }

    //Rescale variables to be between 0 and 1

    angle = remap01(-numInScatteringPoints, numInScatteringPoints,
angle);

    opticalDensity *= stepSize;
    opticalDensity = remap01(0, atmosphereRadius *
atmosphereRadius, opticalDensity) * angle * min(1, 2 / density);
    opticalDensity = clamp(pow(opticalDensity, dispersionPower) *
dispersionScale, 0, 1);

    atmosphereDensity = 1 - remap01(0, numInScatteringPoints,
atmosphereDensity);
    atmosphereDensity = clamp(atmosphereDensity + density / 100,
0, 1);
}

void lightInDir(float3 enterPoint, float3 stepDir, float stepSize,
float blueNoise)
{
    lightIntensity = 0;
    float enterIntensity = 0;

    for (int i = 0; i < numInScatteringPoints; i++)
    {
        float3 samplePoint = enterPoint + stepDir * stepSize * i;

        lightIntensity += opticalDepth(samplePoint, 1, 1.77) +
blueNoise;
        if (i == 0)
            enterIntensity = lightIntensity;
    }
    lightIntensity = remap01(0, numInScatteringPoints,
lightIntensity);
}

```

```

        lightIntensity = lerp(enterIntensity, lightIntensity,
abs(lightIntensity - enterIntensity));
    }

    float4 frag (v2f i) : SV_Target
{
    float p = 0.01;

    density = pow(density, 2);
    numInScatteringPoints = 20;

    float4 originalCol = tex2D(_MainTex, i.uv);
    float sceneDepthNonLinear =
SAMPLE_DEPTH_TEXTURE(_CameraDepthTexture, i.uv);
    float sceneDepth = LinearEyeDepth(sceneDepthNonLinear) *
length(i.viewVector);

    float3 rayOrigin = _WorldSpaceCameraPos;
    float3 rayDir = normalize(i.viewVector);

    //Calculate the distance to the default ocean level and the
position of the point on the ocean surface
    float dstToDefault = raySphere(planetCentre, oceanRadius,
rayOrigin, rayDir);
    float3 oceanPos = rayOrigin + rayDir * dstToDefault -
planetCentre;

    float oceanNoise = 0;
    if (dstToDefault < 100 && roughBed == 0)
        oceanNoise = LayeredNoise(oceanPos, _Time.x * 5 *
windSpeed, 10, 1);

    //Increase the radius of the ocean to make waves
    oceanRadius += oceanNoise * remap01(100, 90, dstToDefault);

    //Calculate the distance to and through the updated ocean
surface
    float2 oceanInfo = raySphere(planetCentre, oceanRadius,
rayOrigin, rayDir);
    float dstToOcean = oceanInfo.x;
    float dstThroughOcean = oceanInfo.y;

    float dstToSurface = min(sceneDepth, dstToOcean);

    float3 clipPlanePos = rayOrigin + i.viewVector *
_ProjectionParams.y;
}

```

```

        float2 atmosInfo = raySphere(planetCentre, atmosphereRadius,
rayOrigin, rayDir);
        float dstToAtmosphere = atmosInfo.x;

        float dstThroughAtmosphere = min(dstToSurface -
dstToAtmosphere, atmosInfo.y);

        float2 discInfo = raySphere(planetCentre, planetRadius * 5,
rayOrigin, rayDir);

        float drawRingMode = 0;
        if (min(dstToSurface - discInfo.x, discInfo.y) > 0)
            drawRingMode = GenerateRings(rayOrigin, rayDir,
dstToSurface, dstToAtmosphere);
        //If a ring is to be drawn in front of the atmosphere
        if (drawRingMode == 1)
            originalCol = lerp(originalCol, discColour, discColour.a);

        if (dstThroughAtmosphere > 0)
        {
            float3 enterPoint = rayOrigin + rayDir * dstToAtmosphere;
            float3 cEnterPoint;

            float correctedAtmDst = dstThroughAtmosphere;
            //If inside atmosphere
            if (dstToAtmosphere == 0)
            {
                //Get distance behind player out of atmosphere
                float newDist = raySphere(planetCentre,
atmosphereRadius, rayOrigin, -rayDir).y;
                if (newDist < atmosphereRadius - planetRadius &&
dot(rayDir, planetCentre - rayOrigin) < 0)
                {
                    cEnterPoint = rayOrigin - rayDir * newDist;
//actually exit point
                    correctedAtmDst += newDist; //add distance behind
player
                }
            }
            else
                cEnterPoint = enterPoint;

            float blueNoise = 0;
            if (sceneDepth > 9999)
            {
                blueNoise = tex2Dlod(BlueNoise, float4(squareUV(i.uv)
* ditherScale,0,0));
                blueNoise = (blueNoise - 0.5) * ditherStrength * p;
            }
        }
    }
}

```

```

        densityInDir(cEnterPoint, rayDir, correctedAtmDst /
numInScatteringPoints, blueNoise);
        lightInDir(enterPoint, rayDir, dstThroughAtmosphere /
numInScatteringPoints, blueNoise);

            float latitude = remap01(-atmosphereRadius,
atmosphereRadius, enterPoint.y - planetCentre.y) + LayeredNoise(enterPoint -
planetCentre, _Time.x * windSpeed, noiseFreq, noiseAmp) + blueNoise;

                //Get the color of the atmosphere based on the latitude
(colour bands) and density (thinner means sunset colours are used) at the
given point
            float4 fogColour = tex2D(fogColourRings,
float2(clamp(latitude, p, 1 - p), clamp(opticalDensity, p, 1 - p)));
            fogColour *= 1 - lightIntensity;

                originalCol = GenerateOcean(originalCol, clipPlanePos,
rayDir, sceneDepth, dstToOcean, dstThroughOcean, oceanPos, oceanNoise,
fogColour);

            float4 atmosphereColour = lerp(originalCol, fogColour,
atmosphereDensity);

                //Anything closer than the fogRange should be visible
if (dstThroughAtmosphere < fogRange)
            atmosphereColour = lerp(originalCol, atmosphereColour,
remap01(0, fogRange, dstThroughAtmosphere));

                //If a ring is to be drawn behind the the atmosphere
if (drawRingMode == 2)
            atmosphereColour = lerp(atmosphereColour, discColour,
discColour.a);

            return atmosphereColour;
        }
        //If a ring is to be drawn
if (drawRingMode != 0)
        originalCol = lerp(originalCol, discColour, discColour.a);

        return originalCol;
    }

    ENDCG
}
}
```

AudioHandler.cs

```

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

public class AudioHandler : MonoBehaviour
{
    [SerializeField] private RobotWeight robotWeight;
    [SerializeField] private ShipWeight shipWeight;

    [SerializeField] private AudioClip hiveAmbience;

    [SerializeField] private AudioClip pavaneForDeadPrincess;
    [SerializeField] private AudioClip gymnopiedieNo1;

    [SerializeField] private AudioClip click;

    private bool queueingMusic;
    public float speedClipT;

    private float engineDemand;
    private float pitch;

    private float sqrAtmRadius = -1;

    AudioSource musicSource;
    AudioSource windSource;
    [SerializeField] AudioSource shipEngineSource;
    AudioSource oneShotSource;

    // Start is called before the first frame update
    void Start()
    {
        musicSource = transform.GetChild(0).GetComponent<AudioSource>();
        windSource = transform.GetChild(1).GetComponent<AudioSource>();
        oneShotSource = transform.GetChild(2).GetComponent<AudioSource>();
    }

    // Update is called once per frame
    void Update()
    {
        if (CameraState.flyingShip && shipWeight.sigWeight == null && !queueingMusic)
            PlayPavane();

        if (CameraState.flyingShip && Input.GetKeyDown(KeyCode.X))
            oneShotSource.PlayOneShot(click);

        if (robotWeight.sigWeight != null)
        {
            if (sqrAtmRadius == -1)
            {
                sqrAtmRadius = robotWeight.sigWeight.planet.atmosphere.atmosRadius;
                sqrAtmRadius *= sqrAtmRadius;
            }
            if (!windSource.isPlaying && !CameraState.inHive && (robotWeight.position -
robotWeight.sigWeight.position).sqrMagnitude < sqrAtmRadius)
                PlayWind();
        }
        else
    }
}

```

```

void PlayPavane()
{
    StartCoroutine(CoPlayMusicInSpace(pavaneForDeadPrincess));
}
IEnumerator CoPlayMusicInSpace(AudioClip music)
{
    queueingMusic = true;

    float startVolume = musicSource.volume;
    musicSource.volume = 0;

    yield return new WaitForSeconds(Random.Range(50, 70));

    musicSource.clip = music;
    musicSource.Play();

    while (musicSource.volume < startVolume && shipWeight.sigWeight == null)
    {
        musicSource.volume += startVolume * Time.deltaTime * 0.2f;
        yield return new WaitForSeconds();
    }

    yield return new WaitUntil(new System.Func<bool>(() => shipWeight.sigWeight != null));

    while (musicSource.volume > 0)
    {
        musicSource.volume -= startVolume * Time.deltaTime;
        yield return new WaitForSeconds();
    }

    musicSource.Stop();
    musicSource.volume = startVolume;

    queueingMusic = false;
}

void PlayWind()
{
    StartCoroutine(CoPlayWind());
}
IEnumerator CoPlayWind()
{
    float startVolume = windSource.volume;

    windSource.Play();
    windSource.loop = true;

    Random.InitState(robotWeight.sigWeight.planet.planetValues.environmentSeed);
//so that its the same for all clouds
    Vector3 windDirSeed = Random.onUnitSphere;
    Random.InitState(System.Environment.TickCount);

    PlanetEffect windEffect = robotWeight.sigWeight.planet.atmosphere;

    while (robotWeight.sigWeight != null && !CameraState.inHive)
    {
        //Pitch and volume of wind are lower in the ship
        windSource.volume = Mathf.Lerp(windSource.volume, startVolume *
        Mathf.InverseLerp(0, 6, windEffect.density) * (CameraState.inShip ? 0.5f : 1),
        Time.deltaTime);
}

```

```
        windSource.pitch = (1 + InventoryUI.robotTemperature) *  
(CameraState.inShip ? 0.25f : 0.5f);  
        //Wind sounds like it comes from a direction  
        windSource.panStereo = 0.1f * Vector3.Dot(Camera.main.transform.right,  
Vector3.Cross(robotWeight.transform.up, windDirSeed).normalized);  
  
        yield return new WaitForEndOfFrame();  
    }  
  
    windSource.Stop();  
    windSource.volume = startVolume;  
}  
}
```

BlitMaterialFeature.cs

```

//Note from Adam Catley:
//The sections of this script that I wrote are commented to indicate this

// Copyright (C) 2020 Ned Makes Games

// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.

using UnityEngine;
using UnityEngine.Rendering;
using UnityEngine.Rendering.Universal;

public class BlitMaterialFeature : ScriptableRendererFeature
{
    class RenderPass : ScriptableRenderPass
    {

        private string profilingName;
        private int materialPassIndex;
        private RenderTargetIdentifier sourceID;
        private RenderTargetHandle[] tempDest;
        private EffectsHandler effects;

        public RenderPass(string profilingName, int passIndex, EffectsHandler effects)
        : base()
        {
            this.profilingName = profilingName;
            this.materialPassIndex = passIndex;
            this.effects = effects;
        }

        public void SetSource(RenderTargetIdentifier source)
        {
            this.sourceID = source;
        }

        ///START OF ADAM CATLEY'S CODE

        public override void Execute(ScriptableRenderContext context, ref
        RenderingData renderingData)
        {
            //renderPassEvent = Camera.main.transform.position.sqrMagnitude < 700 *
            700 ? RenderPassEvent.BeforeRenderingTransparents :
            RenderPassEvent.AfterRenderingTransparents;

            Material[] materials = effects.GetMaterials().ToArray();

            int L = materials.Length;

            if (L == 0)
                return;
    }
}

```

```

        CommandBuffer cmd = CommandBufferPool.Get(profilingName);

        RenderTextureDescriptor cameraTextureDesc =
renderingData.cameraData.cameraTargetDescriptor;
        cameraTextureDesc.depthBufferBits = 0;

        RenderTargetIdentifier[] sources = new RenderTargetIdentifier[L];
        tempDests = new RenderTargetHandle[L];

        //Loop through each material and blit the source to the temporary render
        target using the current material (ordered by distance to player)
        //Each material is generated either from the Atmosphere.shader,
Clouds.shader or ScreenEffects.shader scripts
        for (int i = 0; i < L; i++)
        {
            if (i == 0)
                sources[i] = sourceID;
            else
                sources[i] = tempDests[i - 1].Identifier();

            tempDests[i].Init(i.ToString());
            cmd.GetTemporaryRT(tempDests[i].id, cameraTextureDesc,
FilterMode.Bilinear);
            cmd.Blit(sources[i], tempDests[i].Identifier(), materials[i]);
        }

        //Loop through each material in reverse order and blit the temporary
        render target to the source.
        for (int i = 0; i < L; i++)
            cmd.Blit(tempDests[L - 1 - i].Identifier(), sources[L - 1 - i]);

        context.ExecuteCommandBuffer(cmd);
        cmd.Clear();
        CommandBufferPool.Release(cmd);
    }

    public override void FrameCleanup(CommandBuffer cmd)
    {
        if (tempDests == null)
            return;
        foreach (RenderTargetHandle _tempDest in tempDests)
            cmd.ReleaseTemporaryRT(_tempDest.id);
    }

    ///END OF ADAM CATLEY'S CODE
}

[System.Serializable]
public class Settings
{
    public int materialPassIndex = -2; // -1 means render all passes
    public RenderPassEvent renderEvent =
RenderPassEvent.BeforeRenderingTransparents; // ADAM CATLEY ADDED THIS LINE so that
transparent foliage is not coloured over
    public EffectsHandler effects;
}

[SerializeField]
private Settings settings = new Settings();

private RenderPass renderPass;

```

```
public override void Create()
{
    renderPass = new RenderPass(name, settings.materialPassIndex,
settings.effects);
    SetRenderPassEvent(settings.renderEvent);
}

public void SetRenderPassEvent(RenderPassEvent renderPassEvent)
{
    renderPass.renderPassEvent = renderPassEvent;
}

public override void AddRenderPasses(ScriptableRenderer renderer, ref
RenderingData renderingData)
{
    renderPass.SetSource(renderer.cameraColorTarget);
    renderer.EnqueuePass(renderPass);
}
}
```

Branch.cs

```

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

public class Branch
{
    public static TreeGen treeObj;
    public static float trunkLength;
    public static int leafDensity;
    public static Vector3 planetNormal;

    private bool created = false;
    private GameObject branch;
    public Vector3 startPoint { get; private set; }
    public Vector3 endPoint { get; private set; }
    public float branchLength { get; private set; }
    public float branchWidth { get; private set; }
    public Vector3 branchNormal { get; private set; }
    public int splitsInto { get; private set; }
    public int consecutiveSplits { get; private set; }

    public Branch(Vector3 _startPos, float _branchLength, float _branchWidth, Vector3 parentNormal, Vector2 directionRange, int _consecutiveSplits)
    {
        consecutiveSplits = _consecutiveSplits + 1;

        startPoint = _startPos;
        branchLength = _branchLength;
        branchWidth = _branchWidth;
        endPoint = GetBranchEndPoint(parentNormal, directionRange);
        branchNormal = (endPoint - startPoint).normalized;
        splitsInto = Random.Range(1, 3);

        branch = new GameObject("Branch of Layer " + consecutiveSplits);

        branch.layer = treeObj.gameObject.layer;
        branch.transform.parent = treeObj.root.transform;
        branch.transform.localPosition = Vector3.zero;
    }
    Vector3 GetBranchEndPoint(Vector3 parentNormal, Vector2 directionRange)
    {
        float z;
        //Root branch more aligned with planetNormal
        if (consecutiveSplits == 1)
            z = Random.Range(Mathf.Cos(Mathf.PI / 12), 1);
        else
            z = Random.Range(Mathf.Cos(Mathf.PI / 6), Mathf.Cos(Mathf.PI / 3));
        float a = Random.Range(directionRange.x, directionRange.y);
        //https://math.stackexchange.com/questions/56784/generate-a-random-direction-within-a-cone/205589#205589
        Vector3 normalDir = new Vector3(Mathf.Sqrt(1 - z * z) * Mathf.Cos(a),
        Mathf.Sqrt(1 - z * z) * Mathf.Sin(a), z);
        return startPoint + Quaternion.FromToRotation(Vector3.forward, (planetNormal +
        parentNormal) / 2) * normalDir * branchLength;
    }
    public void HideBranch()
    {
        branch.SetActive(false);
    }
    public void ShowBranch()
    {
        branch.SetActive(true);
    }
}

```

```

        if (!created)
            CreateBranch();
        created = true;
    }
    private void CreateBranch()
    {
        GameObject branchObj = Object.Instantiate(treeObj.BranchPrefab,
branch.transform);
        branchObj.transform.localPosition = (startPoint + endPoint) / 2;
        branchObj.transform.GetChild(0).rotation =
Quaternion.Euler(Quaternion.LookRotation(branchNormal).eulerAngles + new Vector3(90,
0, 0));
        branchObj.transform.GetChild(0).localScale = new Vector3(branchWidth,
branchLength / 2, branchWidth);

        if (consecutiveSplits <= 2)
            branchObj.transform.GetChild(0).GetComponent<CapsuleCollider>().enabled =
true;

        CreateLeaves();
    }
    public void CreateLeaves()
    {
        /*
        for (int i = 0; i < Random.Range(0, leafDensity * consecutiveSplits); i++)
        {
            GameObject leafObj = Object.Instantiate(gameObject.LeafPrefab,
branch.transform);
            leafObj.transform.position = Vector3.Lerp(startPoint, endPoint,
Random.Range(Random.Range(0.25f, 0.5f), 1));
            leafObj.transform.rotation = Random.rotation;
        }
        */
        //Add leaves only to branches that are higher up the tree
        if (consecutiveSplits > 2)
        {
            branch.transform.GetChild(0).GetChild(1).gameObject.SetActive(true);
            for (int i = 0; i < branch.transform.GetChild(0).GetChild(1).childCount;
i++)
            {
                GameObject leafObj =
branch.transform.GetChild(0).GetChild(1).GetChild(i).gameObject;
                leafObj.transform.position = branch.transform.position +
Vector3.Lerp(startPoint, endPoint, Random.Range(Random.Range(0.25f, 0.5f), 1));
                leafObj.transform.rotation = Random.rotation;
            }
        }
    }
}

```

CameraState.cs

```

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

public static class CameraState
{
    public static bool withinShip;
    public static bool inShip;
    public static bool flyingShip;
    public static bool playingTetris;
    public static bool inHive;
    public static bool isPaused;
    public static bool isDead;
    private static RoboVision roboVision;
    public enum LockState { locked, unlocked, changing }

    public static void Reset()
    {
        flyingShip = false;
        inHive = false;
        roboVision.Unlock();
        foreach (FlipSwitch flipSwitch in Object.FindObjectsOfType<FlipSwitch>())
            flipSwitch.Start();
        foreach (ShipDoorOpen shipDoor in Object.FindObjectsOfType<ShipDoorOpen>())
            shipDoor.Start();
        Object.FindObjectsOfType<Interface>().Where(computer =>
computer.shipComputer).First().StopInteracting();
    }

    public static bool CamIsInteractingW(Vector3 objectPos, Vector3 fwdNormal, float
maxDist, float maxAngle)
    {
        if (!Input.GetKeyDown(KeyCode.E))
            return false;

        return CanInteractW(objectPos, fwdNormal, maxDist, maxAngle);
    }

    public static bool CamIsInteractingW(Vector3 objectPos, float maxDist)
    {
        if (!Input.GetKeyDown(KeyCode.E))
            return false;

        return !((Camera.main.transform.position - objectPos).sqrMagnitude > maxDist *
maxDist);
    }

    public static bool CanInteractW(Vector3 objectPos, Vector3 fwdNormal, float maxDist,
float maxAngle)
    {
        if ((Camera.main.transform.position - objectPos).sqrMagnitude > maxDist *
maxDist)
            return false;

        Vector3 camDir = Camera.main.transform.forward;
        Vector3 objDir = (Camera.main.transform.position - objectPos).normalized;
        float angle = Mathf.Acos(Vector3.Dot(camDir, objDir)) * Mathf.Rad2Deg;
    }
}

```

```
        return !(Mathf.Abs(angle) < 180 - maxAngle || Vector3.Dot(camDir, fwdNormal) <
0);
    }

    public static bool LockCamera(Transform cameraPos)
{
    if (roboVision != null || Camera.main.TryGetComponent(out roboVision))
    {
        roboVision.Lock(cameraPos);
        return true;
    }
    return false;
}

public static bool UnlockCamera()
{
    if (roboVision != null || Camera.main.TryGetComponent(out roboVision))
    {
        roboVision.Unlock();
        return true;
    }
    return false;
}

public static bool InLockState(LockState lockState)
{
    if (roboVision != null || Camera.main.TryGetComponent(out roboVision))
        return roboVision.isLocked == lockState;

    return lockState == LockState.unlocked;
}
}
```

CeilLight.cs

```

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

public class CeilLight : MonoBehaviour
{
    [SerializeField] private Light lightSource;
    [SerializeField] private float speed;
    [SerializeField] private float intensity = 5;

    private Material lightMat;
    private float seed;

    private void Start()
    {
        GetComponentInChildren<MeshRenderer>().sharedMaterial = new
Material(GetComponentInChildren<MeshRenderer>().sharedMaterial);
        lightMat = GetComponentInChildren<MeshRenderer>().sharedMaterial;

        Vector3 T = transform.position + Vector3.one;
        seed = T.x * T.y + T.x * T.z + T.z * T.y;
    }

    // Update is called once per frame
    void Update()
    {
        //Only show the light if the camera is close enough and looking towards the light
        if ((transform.position - Camera.main.transform.position).sqrMagnitude < 100 ||
CameraState.CanInteractW(transform.position, (transform.position -
Camera.main.transform.position).normalized, 20, 160))
        {
            if (!lightSource.enabled)
                lightSource.enabled = true;

            //Flicker
            float noise = Mathf.PerlinNoise(seed + Time.realtimeSinceStartup * speed *
10, 0.1f);
            bool onOff = Mathf.RoundToInt(noise + 0.4f) == 1;

            //lightSource.intensity = 5 * noise;

            if (onOff)
            {
                lightMat.EnableKeyword("_ONOFF");
                lightSource.intensity = intensity;
            }
            else
            {
                lightMat.DisableKeyword("_ONOFF");
                lightSource.intensity = noise * intensity;
            }
        }
        else if (lightSource.enabled)
        {
            lightMat.EnableKeyword("_ONOFF");
            lightSource.enabled = false;
        }
    }
}

```

}
}
}

CloudFollower.cs

```

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

public class CloudFollower : MonoBehaviour
{
    [SerializeField] private Planet planet;

    [SerializeField] private Transform sun;
    [SerializeField] private float moveDist = 5;
    [SerializeField] private float moveTime = 5;
    [SerializeField] private float distFromPlayer = 60;
    [SerializeField] private float distRange = 10;
    private MeshRenderer meshRenderer;
    private Light sunLight;
    private Material dustMat;
    private float lerp;
    private float endLerp = 0.6f;
    public bool active = false;

    // Start is called before the first frame update
    void Start()
    {
        moveTime += (Random.value * 2 - 1) * (moveTime / 50);
        lerp = 1;

        sunLight = planet.transform.GetChild(4).GetComponent<Light>();
        meshRenderer = GetComponent<MeshRenderer>();
        dustMat = new Material(meshRenderer.sharedMaterial);
        GetComponent<MeshRenderer>().sharedMaterial = dustMat;

        transform.localScale *= Random.Range(40, 80f);
    }

    // Update is called once per frame
    void Update()
    {
        if (planet.atmosphere == null)
            return;

        lerp += Time.deltaTime / moveTime;

        if (!active && (planet.transform.position -
Camera.main.transform.position).sqrMagnitude < planet.atmosphere.atmosRadius *
planet.atmosphere.atmosRadius)
        {
            active = true;
            meshRenderer.enabled = active;
        }
        else if (active && (planet.transform.position -
Camera.main.transform.position).sqrMagnitude >= planet.atmosphere.atmosRadius *
planet.atmosphere.atmosRadius)
        {
            active = false;
        }
        if (!active && lerp > 1)
        {
            meshRenderer.enabled = false;
        }
    }
}

```

```

        return;
    }

    if (lerp > 1)
    {
        endlerp = 0.6f;
        lerp = 0;

        //Calculate the zone direction and position in which the cloud will be
generated

        Vector3 zoneDir = Vector3.Cross(Camera.main.transform.position -
planet.transform.position, Random.onUnitSphere).normalized * (distFromPlayer +
Random.Range(-distRange, distRange));

        Vector3 zonePos = zoneDir + Camera.main.transform.position;
        //zonePos += (zonePos - planet.transform.position).normalized *
transform.localScale.z / 2;
        Vector3 zoneNormal = (zonePos - planet.transform.position).normalized;

        //Physics.Raycast(zoneNormal * planet.planetMesh.elevationData.Max, -
zoneNormal, out RaycastHit hitInfo, planet.planetValues.radius);

        //Raycast to find the intersection point of the zone and the planet
surface
        Physics.Raycast(zoneNormal * planet.planetMesh.elevationData.Max, -
zoneNormal, out RaycastHit hitInfo, planet.planetValues.radius);

        //Physics.Raycast(zonePos, -zoneNormal, out RaycastHit posInfo,
planet.planetValues.radius);
        transform.position = zonePos + zoneNormal * 10;// + Random.value *
moveDist * transform.right;
        //Look towards player
        transform.rotation =
Quaternion.LookRotation(Vector3.ProjectOnPlane((Camera.main.transform.position -
transform.position).normalized, zoneNormal), zoneNormal);

        dustMat.SetFloat("_minElevation", planet.planetMesh.elevationData.Min);
        dustMat.SetFloat("_maxElevation", planet.planetMesh.elevationData.Max);
        dustMat.SetTexture("_planetTexture",
planet.terrainMat.GetTexture("_planetTexture"));
        dustMat.SetVector("_position", hitInfo.point - planet.transform.position);
        dustMat.SetVector("_oceanCentre", planet.transform.position);
        dustMat.SetFloat("_oceanRadius", planet.planetValues.radius * 0.99f);

        float shadeAngle = Vector3.Dot((sun.position -
planet.transform.position).normalized, (transform.position -
planet.transform.position).normalized);
        shadeAngle = Mathf.Pow(Mathf.Clamp01(shadeAngle), 0.2f);
        dustMat.SetFloat("_transparency", sunLight.intensity * shadeAngle);
    }

    RunCycle();
}

void RunCycle()
{
    //Fade in and out at the start and end of the cloud cycle
    if (endlerp < 0.2f)
        dustMat.SetFloat("_alphaFade", Mathf.Min(lerp / 0.2f, 1 - (lerp - endlerp) /
0.2f));
    else if (lerp < 0.2f)
}

```

```
        dustMat.SetFloat("_alphaFade", lerp / 0.2f);
    else if (lerp > endlerp)
        dustMat.SetFloat("_alphaFade", 1 - (lerp - endlerp) / 0.2f);

    transform.position += Time.deltaTime * moveDist / moveTime * transform.right;

    if (lerp > endlerp + 0.2f)
        lerp = 1;
}
}
```

Clouds.shader

```
//Note, this shader is similar to the Atmosphere.shader script in many ways,
but the way the atmosphere is generated is completely different

Shader "Custom/Clouds"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
    }
    SubShader
    {
        // No culling or depth
        Cull Off ZWrite Off ZTest Always

        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"
            #include "Math.cginc"
            #include "PerlinNoise.cginc"
            //

            struct appdata {
                float4 vertex : POSITION;
                float4 uv : TEXCOORD0;
            };

            struct v2f {
                float4 pos : SV_POSITION;

                float2 uv : TEXCOORD0;
                float3 viewVector : TEXCOORD1;
            };

            v2f vert (appdata v) {
                v2f output;
                output.pos = UnityObjectToClipPos(v.vertex);

                output.uv = v.uv;
                // Camera space matches OpenGL convention where cam
                forward is -z. In unity forward is positive z.
                // (https://docs.unity3d.com/ScriptReference/Camera-cameraToWorldMatrix.html)
            }
        }
    }
}
```

```
        float3 viewVector = mul(unity_CameraInvProjection,
float4(v.uv.xy * 2 - 1, 0, -1));
        output.viewVector = mul(unity_CameraToWorld,
float4(viewVector,0));
        return output;
    }

    float2 squareUV(float2 uv) {
        float width = _ScreenParams.x;
        float height = _ScreenParams.y;
        //float minDim = min(width, height);
        float scale = 1000;
        float x = uv.x * width;
        float y = uv.y * height;
        return float2 (x/scale, y/scale);
    }

    sampler2D _MainTex;
    sampler2D _CameraDepthTexture;

    sampler2D fogColourRings;
    sampler2D BlueNoise;

    float3 sunCentre;
    float3 planetCentre;
    float planetRadius;
    float oceanRadius;
    float fogRange;

    float atmosphereRadius;
    float cloudLyrDepth;

    float noiseFreq;
    float noiseAmp;

    float ditherScale;
    float ditherStrength;

    float lightIntensity;
    float numInScatteringPoints;

    float windSpeed;

    sampler2D waveNormalA;
    sampler2D waveNormalB;
    float4 oceanColour;
    float visibleDepth;
    float waveSpeed;
```

```

    float waveStrength;
    float waveNormalScale;
    float smoothness;

    float roughBed;

    float reflection01(float3 normal, float3 position)
    {
        return dot(normal, normalize(sunCentre - position)) * 2 - 1;
    }

    float4 GenerateOcean(float4 originalCol, float3 clipPlanePos,
float3 rayDir, float sceneDepth, float dstToOcean, float3 oceanPos, float
oceanNoise, float4 fogColour)
    {
        if (dstToOcean < sceneDepth)
        {
            oceanColour = normalize(oceanColour) * 0.1;

            waveSpeed *= windSpeed;

            float dstAboveWater = length(clipPlanePos - planetCentre)
- oceanRadius;

            float4 finalColour = lerp(originalCol, oceanColour,
clamp((sceneDepth - dstToOcean) / 10, 0, 1));
            finalColour = lerp(oceanColour, finalColour,
clamp((sceneDepth - dstToOcean) / (5 - oceanNoise), 0, 1));

            float3 oceanSphereNormal = normalize(oceanPos);

            float2 waveOffsetA = float2(_Time.x * waveSpeed, _Time.x *
waveSpeed * 0.8);
            float2 waveOffsetB = float2(_Time.x * waveSpeed * -0.8,
_Time.x * waveSpeed * -0.3);
            float3 waveNormal = triplanarNormal(oceanPos,
oceanSphereNormal, waveNormalScale / planetRadius, waveOffsetA, waveNormalA);
            waveNormal = triplanarNormal(oceanPos, waveNormal, 1 /
waveNormalScale, waveOffsetB, waveNormalB);
            waveNormal = normalize(lerp(oceanSphereNormal, waveNormal,
waveStrength));
            //return float4(oceanNormal * .5 + .5,1);
            float diffuseLighting = saturate(dot(oceanSphereNormal,
normalize(sunCentre - planetCentre)));
            float specularAngle = lerp(waveNormal,
acos(dot(normalize(normalize(sunCentre - planetCentre) - rayDir),
waveNormal)), 0.2);
            float specularExponent = specularAngle / (1 - smoothness);
        }
    }
}

```

```

        float specularHighlight = exp(-specularExponent *
specularExponent);

        finalColour *= diffuseLighting;
        finalColour += specularHighlight * lerp(finalColour,
float4(1,1,1,1), 0.1);

        finalColour = finalColour * (0.75 +
reflection01(waveNormal, oceanPos) * 0.25);

        finalColour = float4(min(finalColour.r, 1),
min(finalColour.g, 1), min(finalColour.b, 1), 1);

        return finalColour;
    }
    return originalCol;
}
float opticalDepth(float3 samplePoint, float spread1, float
spread2)
{
    return remap01(0, 2 * planetRadius * spread1,
raySphere(planetCentre, planetRadius * spread2, samplePoint,
normalize(sunCentre - samplePoint)).y);
}
void lightInDir(float3 enterPoint, float3 stepDir, float stepSize,
float blueNoise)
{
    lightIntensity = 0;
    float enterIntensity = 0;

    for (int i = 0; i < numInScatteringPoints; i++)
    {
        float3 samplePoint = enterPoint + stepDir * stepSize * i;

        lightIntensity += opticalDepth(samplePoint, 1, 1.25) +
blueNoise;
        if (i == 0)
            enterIntensity = lightIntensity;
    }
    lightIntensity = remap01(0, numInScatteringPoints,
lightIntensity);
    lightIntensity = lerp(enterIntensity, lightIntensity,
abs(lightIntensity - enterIntensity));
}

float4 frag (v2f i) : SV_Target
{
    float p = 0.01;

```

```

        numInScatteringPoints = 20;

        float4 originalCol = tex2D(_MainTex, i.uv);
        float sceneDepthNonLinear =
SAMPLE_DEPTH_TEXTURE(_CameraDepthTexture, i.uv);
        float sceneDepth = LinearEyeDepth(sceneDepthNonLinear) *
length(i.viewVector);

        float3 rayOrigin = _WorldSpaceCameraPos;
        float3 rayDir = normalize(i.viewVector);

        float dstToDefault = raySphere(planetCentre, oceanRadius,
rayOrigin, rayDir);//
        float3 oceanPos = rayOrigin + rayDir * dstToDefault -
planetCentre;

        float oceanNoise = 0;
        if (dstToDefault < 100 && roughBed == 0)
            oceanNoise = LayeredNoise(oceanPos, _Time.x * 5 *
windSpeed, 10, 1);

        oceanRadius += oceanNoise * remap01(100, 90, dstToDefault);

        float2 oceanInfo = raySphere(planetCentre, oceanRadius,
rayOrigin, rayDir);
        float dstToOcean = oceanInfo.x;
        float dstThroughOcean = oceanInfo.y;

        float dstToSurface = min(sceneDepth, dstToOcean);

        float3 clipPlanePos = rayOrigin + i.viewVector *
_ProjectionParams.y;

        float2 btmHitInfo = raySphere(planetCentre, atmosphereRadius -
cloudLyrDepth, rayOrigin, rayDir);
        float dstToBtmCloudLyr = btmHitInfo.x;
        float dstThroughBtmCloudLyr = min(btmHitInfo.y, (dstToSurface -
dstToBtmCloudLyr));

        float2 topHitInfo = raySphere(planetCentre, atmosphereRadius,
rayOrigin, rayDir);
        float dstToTopCloudLyr = topHitInfo.x;
        float dstThroughTopCloudLyr = min(topHitInfo.y, (dstToSurface -
dstToTopCloudLyr));

        oceanRadius += oceanNoise * remap01(100, 90, dstToDefault);
    }
}

```

```

        if (dstThroughTopCloudLyr > 0) //&& (dstToCloudLyr == 0 &&
dstToSurface > dstThroughCloudLyr || dstToCloudLyr != 0))
{
    //2 if the player is between the top and bottom
    layers, 1 if the player is within the bottom layer, 0 if the player is out of
    the atmosphere
    int playerAlt = dstToTopCloudLyr > 0 ? 2 :
dstToBtmCloudLyr > 0 ? 1 : 0;
    float playerAltLerp = remap01(atmosphereRadius -
cloudLyrDepth, atmosphereRadius, length(rayOrigin - planetCentre));

    //Calculate the entry and exit points of the ray through
    the cloud layers (they match if playerAlt is 0)
    float3 enter1;
    if (playerAlt == 2)
        enter1 = rayOrigin + rayDir * dstToTopCloudLyr;
    else if (playerAlt == 1)
        enter1 = rayOrigin;
    else
        enter1 = rayOrigin + rayDir * dstThroughBtmCloudLyr;

    float3 exit1;
    if (playerAlt == 0)
        exit1 = rayOrigin + rayDir * dstThroughTopCloudLyr;
    else if (dstThroughBtmCloudLyr > 0)
        exit1 = rayOrigin + rayDir * dstToBtmCloudLyr;
    else if (dstThroughBtmCloudLyr == 0)
        exit1 = rayOrigin + rayDir * (dstToTopCloudLyr +
dstThroughTopCloudLyr);

    float3 enter2;
    if (playerAlt != 0 && dstThroughBtmCloudLyr > 0 &&
(dstToBtmCloudLyr + dstThroughBtmCloudLyr) < dstToSurface)
        enter2 = rayOrigin + rayDir * (dstToBtmCloudLyr +
dstThroughBtmCloudLyr);
    else
        enter2 = enter1;

    float3 exit2;
    if (playerAlt != 0 && dstThroughBtmCloudLyr > 0 &&
(dstToBtmCloudLyr + dstThroughBtmCloudLyr) < dstToSurface)
        exit2 = rayOrigin + rayDir * (dstToTopCloudLyr +
dstThroughTopCloudLyr);
    else
        exit2 = enter1;

    float blueNoise = 0;
}

```

```

        if (sceneDepth > 9999)
        {
            blueNoise = tex2Dlod(BlueNoise, float4(squareUV(i.uv)
* ditherScale,0,0));
            blueNoise = (blueNoise - 0.5) * ditherStrength * p;
        }

            float dstThroughCloudLyr = max(length(enter1 - exit1),
length(enter2 - exit2));
            float cloudStrength = remap01(0, cloudLyrDepth,
dstThroughCloudLyr);
            float frenzel = lerp(remap01(0, atmosphereRadius,
topHitInfo.y), 1, 1 - playerAltLerp);

                //Calculate the latitude of the entry and exit points of
the ray through the cloud layers
                float outLat = remap01(-atmosphereRadius - cloudLyrDepth,
atmosphereRadius, enter1.y - planetCentre.y) + LayeredNoise(enter1 -
planetCentre, _Time.x, noiseFreq, noiseAmp) + blueNoise;
                float inLat = remap01(-atmosphereRadius - cloudLyrDepth,
atmosphereRadius, enter2.y) + LayeredNoise(enter2 - planetCentre, _Time.x,
noiseFreq, noiseAmp) + blueNoise;
                float platerLat = remap01(-atmosphereRadius -
cloudLyrDepth, atmosphereRadius, rayOrigin.y) + LayeredNoise(rayOrigin -
planetCentre, _Time.x, noiseFreq, noiseAmp) + blueNoise;

                //Not interested in sunsets which are the y component of
fogColourRings at atmosphere is opaque at a distance
                float4 outColour = tex2D(fogColourRings,
float2(clamp(outLat, p, 1 - p), p));
                float4 inColour = lerp(tex2D(fogColourRings,
float2(clamp(inLat, p, 1 - p), p)), LayeredNoise(enter2 - planetCentre,
_Time.x * windSpeed, 5, 1), 0.05);
                float4 cloudColour = lerp(outColour, inColour, 1 -
playerAltLerp);

                float fog = lerp(1, remap01(0, planetRadius,
dstThroughTopCloudLyr), 1 - playerAltLerp);
                float4 fogColour = tex2D(fogColourRings,
float2(clamp(platerLat, p, 1 - p), p));

        if (roughBed == 0)
            originalCol = GenerateOcean(originalCol, clipPlanePos,
rayDir, sceneDepth, dstToOcean, oceanPos, oceanNoise, fogColour);

        if (playerAlt != 2)
            cloudColour = lerp(cloudColour, fogColour, fog);
    }
}

```

```
        lightInDir(enter1, rayDir, dstThroughTopCloudLyr /  
numInScatteringPoints, blueNoise);  
  
        float4 finalColour;  
        if (dstThroughBtmCloudLyr < dstToSurface &&  
!(dstThroughBtmCloudLyr + dstToBtmCloudLyr == dstToSurface && playerAlt == 1))  
            finalColour = lerp(originalCol, cloudColour,  
cloudStrength * frenzel);  
        else  
            finalColour = lerp(originalCol, fogColour, fog);  
  
        finalColour *= clamp(1.2 - lightIntensity, 0, 1);  
        finalColour.a = 1;  
        return finalColour;  
    }  
  
    return originalCol;  
}  
  
ENDCG  
}  
}  
}
```

CloudSurround.cs

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

public class CloudSurround : MonoBehaviour
{
    [SerializeField] private Planet planet;
    [SerializeField] private Transform sun;
    [SerializeField] private float moveDist = 5;
    [SerializeField] private float moveTime = 5;
    [SerializeField] private float timeRange = 2;
    [SerializeField] private float distFromPlayer = 150;
    [SerializeField] private float distRange = 10;
    private Vector3 ogScale;

    private Vector3 windDirSeed;
    private float angle01;

    public bool active = false;

    private MeshRenderer meshRenderer;
    private Light sunLight;
    private Material dustMat;

    private float lerp;
    private Vector3 windDir;
    private float newMoveTime;

    private float endlerp = 0.6f;

    // Start is called before the first frame update
    void Start()
    {
        angle01 = (float)transform.GetSiblingIndex() / (transform.parent.childCount - 1);

        newMoveTime = moveTime;
        lerp = 1;

        sunLight = planet.transform.GetChild(4).GetComponent<Light>();
        meshRenderer = GetComponent<MeshRenderer>();
        dustMat = new Material(meshRenderer.sharedMaterial);
        meshRenderer.sharedMaterial = dustMat;

        ogScale = transform.localScale;

        Random.InitState(planet.planetValues.environmentSeed); //so that its the same for
        all clouds
        windDirSeed = Random.onUnitSphere;
        Random.InitState(System.Environment.TickCount);
    }

    // Update is called once per frame
    void Update()
    {
        if (planet.atmosphere == null)
            return;

        lerp += Time.deltaTime / newMoveTime;
```

```

        if (!active && (planet.transform.position -
Camera.main.transform.position).sqrMagnitude < planet.atmosphere.atmosRadius *
planet.atmosphere.atmosRadius)
{
    active = true;
    meshRenderer.enabled = active;
}
else if (active && (planet.transform.position -
Camera.main.transform.position).sqrMagnitude >= planet.atmosphere.atmosRadius *
planet.atmosphere.atmosRadius)
{
    active = false;
}
if (!active && lerp > 1)
{
    meshRenderer.enabled = false;
    return;
}

if (lerp > 1)
{
    endlerp = 0.6f;
    lerp = 0;

newMoveTime = moveTime + Random.Range(-timeRange, timeRange);

transform.localScale = ogScale * Random.Range(40, 80f);

Vector3 planetNormal = (Camera.main.transform.position -
planet.transform.position).normalized;

//Calculate the zone direction and position in which the cloud will be
generated

Vector3 zoneDir = Quaternion.AngleAxis(angle01 * 360, planetNormal) *
Vector3.forward;

windDir = Vector3.Cross(planetNormal, windDirSeed + Random.onUnitSphere /
2).normalized;

Vector3 zonePos = zoneDir * (distFromPlayer + Random.Range(-distRange,
distRange)) + Camera.main.transform.position - windDir;
//zonePos += (zonePos - planet.transform.position).normalized *
transform.localScale.z / 2;
Vector3 zoneNormal = (zonePos - planet.transform.position).normalized;

//Physics.Raycast(zoneNormal * planet.planetMesh.elevationData.Max, -
zoneNormal, out RaycastHit hitInfo, planet.planetValues.radius);

//Raycast to find the intersection point of the zone and the planet
surface
Physics.Raycast(planet.transform.position + zoneNormal *
planet.planetMesh.elevationData.Max, -zoneNormal, out RaycastHit hitInfo,
planet.planetValues.radius);

Vector3 hitPoint = hitInfo.point;
if ((hitPoint - planet.transform.position).magnitude <
planet.planetValues.radius)
    hitPoint = planet.transform.position + (hitPoint -
planet.transform.position).normalized * planet.planetValues.radius;

```

```

        transform.position = hitPoint;// + Random.value * moveDist *
transform.right;
        //Look towards player
        transform.rotation =
Quaternion.LookRotation(Vector3.ProjectOnPlane((Camera.main.transform.position -
transform.position).normalized, zoneNormal), zoneNormal);

        dustMat.SetFloat("_minElevation", planet.planetMesh.elevationData.Min);
        dustMat.SetFloat("_maxElevation", planet.planetMesh.elevationData.Max);
        dustMat.SetTexture("_planetTexture",
planet.terrainMat.GetTexture("_planetTexture"));
        dustMat.SetVector("_position", hitInfo.point - planet.transform.position);
        dustMat.SetVector("_oceanCentre", planet.transform.position);
        dustMat.SetFloat("_oceanRadius", planet.planetValues.radius * 0.99f);

        float shadeAngle = Vector3.Dot((sun.position -
planet.transform.position).normalized, (hitPoint -
planet.transform.position).normalized);
        shadeAngle = Mathf.Clamp01(Mathf.Pow(Mathf.Clamp01(shadeAngle), 0.2f) +
0.1f);
        dustMat.SetFloat("_transparency", sunLight.intensity * shadeAngle);
    }

    RunCycle();
}

void RunCycle()
{
    if (endlerp < 0.2f)
        dustMat.SetFloat("_alphaFade", Mathf.Min(lerp / 0.2f, 1 - (lerp - endlerp) /
0.2f));
    else if (lerp < 0.2f)
        dustMat.SetFloat("_alphaFade", lerp / 0.2f);
    else if (lerp > endlerp)
        dustMat.SetFloat("_alphaFade", 1 - (lerp - endlerp) / 0.2f);

    if ((Camera.main.transform.position - transform.position).sqrMagnitude < 2000)
        endlerp = Mathf.Min(endlerp, lerp);

    transform.position += Time.deltaTime * moveDist / newMoveTime * windDir;

    Debug.DrawLine(transform.position, transform.position + (transform.position -
planet.transform.position).normalized * 50, Color.red);

    if (lerp > endlerp + 0.2f)
        lerp = 1;
}
}

```

ColourGenerator.cs

```

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

public class ColourGenerator
{
    private Planet planet;
    private Texture2D planetTexture;

    private const int textureRes = 50;

    public ColourGenerator(Planet _planet)
    {
        planet = _planet;
        if (planetTexture == null)
            planetTexture = new Texture2D(textureRes, textureRes);
    }
    public void UpdateElevation(ElevationData elevationData)
    {
        planet.terrainMat.SetFloat("_minElevation", elevationData.Min);
        planet.terrainMat.SetFloat("_maxElevation", elevationData.Max);
    }
    public void UpdateColours()
    {
        float lerpRange = 0.05f;

        for (int lat = 0; lat < textureRes; lat++)
        {
            float lat01 = (float)lat / textureRes;
            KeyValuePair<float, Gradient> biomeBelow = new KeyValuePair<float, Gradient>(0, null);
            KeyValuePair<float, Gradient> biomeAbove = new KeyValuePair<float, Gradient>(2, null);

            //Get the biome gradients below and above the current latitude
            foreach (KeyValuePair<float, Gradient> biome in
planet.planetValues.biomeGradients)
            {
                if (biome.Key <= lat01 && lat01 - biome.Key <= lat01 - biomeBelow.Key)
                    biomeBelow = biome;
                else if (biome.Key > lat01 && biome.Key - lat01 <= biomeAbove.Key - lat01)
                    biomeAbove = biome;
            }

            for (int alt = 0; alt < textureRes; alt++)
            {
                float alt01 = (float)alt / textureRes;
                Color pixel;
                //Lerp between the biome gradients based on the distance to the biome
                above
                if (biomeAbove.Key != 2 && biomeAbove.Key - lerpRange < lat01)
                    pixel = Color.Lerp(biomeBelow.Value.Evaluate(alt01),
biomeAbove.Value.Evaluate(alt01), Mathf.InverseLerp(biomeAbove.Key - lerpRange,
biomeAbove.Key, lat01));
                else
                    pixel = biomeBelow.Value.Evaluate(alt01);
            }
        }
    }
}

```

```
        planetTexture.SetPixel(alt, lat, pixel);
    }
}

planetTexture.Apply();

planet.terrainMat.SetTexture("_planetTexture", planetTexture);
planet.setPlanetValues.readonlyPlanetTexture =
planet.terrainMat.GetTexture("_planetTexture");

Random.initState(planet.planetValues.colourSeed);

//Choose a random pixel from the lowest latitude on the planet's texture to
use as the sea color
Color seaColour = planetTexture.GetPixel(Random.Range(0, textureRes), 0);

Color.RGBToHSV(seaColour, out float H, out float _, out float _);

H = Mathf.Lerp(Mathx.EndsCommon(Random.value), H, 0.75f);
float S = Random.value;

if (planet.TryGetComponent(out Weight planetWeight))
    planetWeight.colour = Color.HSVToRGB(H, S, 1);

planet.atmosphere.atmosColour = Color.HSVToRGB(H, S, Random.Range(0.4f,
0.6f));
    planet.atmosphere.ringColour = Color.HSVToRGB(H, S, Mathf.Clamp01(0.2f +
Random.value));

Random.initState(System.Environment.TickCount);
}
public void UpdatePlanetInfo()
{
    planet.oceanMat.SetFloat("_radius", planet.planetValues.radius);
    planet.terrainMat.SetVector("_planetCentre", planet.transform.position);

    planet.atmosphere.planetRadius = planet.planetValues.radius;
    planet.atmosphere.maxFogRange = planet.maxNatureDist;
}

}
```

CompassAligner.cs

```

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

public class CompassAligner : MonoBehaviour
{
    [SerializeField] private RobotWeight robot;
    [SerializeField] private ShipWeight ship;
    [SerializeField] private Material streaksMat;
    private Material compass;
    private float lerp;
    // Start is called before the first frame update
    void Start()
    {
        compass = new Material(GetComponent<MeshRenderer>().sharedMaterial);
        GetComponent<MeshRenderer>().sharedMaterial = compass;
    }

    // Update is called once per frame
    void Update()
    {

        compass.SetFloat("_turnOnLerp", lerp);
        compass.SetFloat("_engineOn", InventoryUI.shipEngineOn01);
        if (ShipWeight.hyperOn)
            compass.EnableKeyword("_HYPERON");
        else
            compass.DisableKeyword("_HYPERON");

        //Show the velocity meter when not close to a planet
        if (robot.sigWeight == null)
        {
            float shipVelocity = ship.velocity.magnitude;
            float speed0to1_5 = Mathf.InverseLerp(0, ship.maxVelocity, shipVelocity) +
0.5f * Mathf.InverseLerp(ship.maxVelocity, ship.maxHyperVelocity, shipVelocity);
            float revs01 = 1 - (1 - Mathf.InverseLerp(0, ship.maxHyperVelocity + 1,
shipVelocity)) % 0.125f / 0.125f;
            compass.SetFloat("_speed0to1_5", speed0to1_5);
            compass.SetFloat("_revs01", revs01);
            if (streaksMat != null)
                streaksMat.SetFloat("_speed0to1_5", speed0to1_5);

            lerp = Mathf.Clamp01(lerp - Time.deltaTime * 3);
            return;
        }
        lerp = Mathf.Clamp01(lerp + Time.deltaTime * 3);

        Vector3 northPole = robot.sigWeight.position + robot.sigWeight.transform.up *
robot.sigWeight.planet.planetValues.radius;
        Vector3 northNeedle = Vector3.ProjectOnPlane(northPole - transform.position,
transform.up).normalized;
        Vector3 localNeedle = transform.InverseTransformDirection(northNeedle);

        compass.SetFloat("_rotationRad", Mathf.Atan2(localNeedle.z, localNeedle.x) +
Mathf.PI / 2);

        Debug.DrawLine(transform.position, northPole);
        Debug.DrawRay(transform.position, northNeedle, Color.red);
    }
}

```

```
        Debug.DrawRay(transform.position, transform.forward, Color.blue);  
    }  
}
```

ConstructionHandler.cs

```

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

public class ConstructionHandler : MonoBehaviour
{
    [SerializeField] private Transform[] robots;
    private Transform robot;
    private TextRender textRender;
    private int conversation;
    private int upgradeIndex;

    // Start is called before the first frame update
    void Start()
    {

        Random.InitState(transform.root.GetComponent<Planet>().planetValues.environmentSeed);
        robot = robots[Random.Range(0, robots.Length)];
        robot.gameObject.SetActive(true);
        textRender = robot.GetChild(0).GetComponent<TextRender>();
        textRender.LoadConversation("construction/greeting/" + Random.Range(0,
4).ToString());
        conversation = 0;
    }

    // Update is called once per frame
    void Update()
    {
        //If the player requests the upgrade conversation and they have sufficient metal
        //to buy an upgrade and the greeting conversation has no more sentences to render
        if (CameraState.CamIsInteractingW(robot.position, 7) &&
!textRender.NextSentence() && InventoryUI.robotMetalCount >= 100)
        {
            if (conversation == 0)
            {
                conversation = 1;
                List<int> upgradeIndexes = new List<int>();

                //Cannot upgrade beyond level 3
                if (InventoryUI.player.robotFuelCapacityLvl != 3)
                    upgradeIndexes.Add(0);
                if (InventoryUI.player.robotFuelBurnRateLvl != 3)
                    upgradeIndexes.Add(1);
                if (InventoryUI.player.robotFuelPerOreLvl != 3)
                    upgradeIndexes.Add(2);
                if (InventoryUI.player.robotOxygenCapacityLvl != 3)
                    upgradeIndexes.Add(3);
                if (InventoryUI.player.robotOxygenTemperatureScaleLvl != 3)
                    upgradeIndexes.Add(4);
                if (InventoryUI.player.robotOxygenSafeScaleLvl != 3)
                    upgradeIndexes.Add(5);
                if (InventoryUI.player.shipFuelCapacityLvl != 3)
                    upgradeIndexes.Add(6);
                if (InventoryUI.player.shipFuelBurnRateLvl != 3)
                    upgradeIndexes.Add(7);
                if (InventoryUI.player.shipSolarChargeRateLvl != 3)
                    upgradeIndexes.Add(8);
            }
        }
    }
}

```

```
upgradeIndex = upgradeIndexes.Count == 0 ? 9 :  
upgradeIndexes[Random.Range(0, upgradeIndexes.Count)];  
    textRender.LoadConversation("construction/upgrade/" +  
upgradeIndex.ToString());  
}  
else if (conversation == 1 && upgradeIndex != 9)  
{  
    conversation = 2;  
    textRender.LoadConversation("construction/upgrade/done");  
}  
}  
//Added to take metal from player *after* robot announces confirmation.  
if (CameraState.CamIsInteractingW(robot.position, 7) && conversation == 2)  
{  
    conversation = 3;  
    InventoryUI.robotMetalCount -= 100;  
  
    switch (upgradeIndex)  
{  
        case 0: InventoryUI.player.robotFuelCapacityLvl++; break;  
        case 1: InventoryUI.player.robotFuelBurnRateLvl++; break;  
        case 2: InventoryUI.player.robotFuelPerOreLvl++; break;  
        case 3: InventoryUI.player.robotOxygenCapacityLvl++; break;  
        case 4: InventoryUI.player.robotOxygenTemperatureScaleLvl++; break;  
        case 5: InventoryUI.player.robotOxygenSafeScaleLvl++; break;  
        case 6: InventoryUI.player.shipFuelCapacityLvl++; break;  
        case 7: InventoryUI.player.shipFuelBurnRateLvl++; break;  
        case 8: InventoryUI.player.shipSolarChargeRateLvl++; break;  
        default: break;  
    };  
    JsonSaver.SaveData("Player_Stats", InventoryUI.player);  
}  
}  
}
```

ControlHelp.cs

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

public class ControlHelp : MonoBehaviour
{
    private bool showControls;

    private enum ControlMode { robot, ship, tetris, none };

    [SerializeField] private TMPro.TextMeshProUGUI robotText;
    [SerializeField] private TMPro.TextMeshProUGUI shipText;
    [SerializeField] private TMPro.TextMeshProUGUI tetrisText;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        showControls ^= PauseMenu.selectedPauseMenuOption == 3;

        ControlMode controlMode = ControlMode.none;
        if (CameraState.flyingShip)
            controlMode = ControlMode.ship;
        else if (CameraState.playingTetris)
            controlMode = ControlMode.tetris;
        else if (CameraState.InLockState(CameraState.LockState.unlocked))
            controlMode = ControlMode.robot;

        robotText.enabled = showControls && controlMode == ControlMode.robot;
        shipText.enabled = showControls && controlMode == ControlMode.ship;
        tetrisText.enabled = showControls && controlMode == ControlMode.tetris;
    }
}
```

CustomTile.cs

```

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

//This tile is prefabricated. There are two custom tiles, the stairs and the hall.
public class CustomTile : MonoBehaviour
{
    public Vector3Int parentKey { get; private set; }

    [SerializeField] private Transform Exit;
    [SerializeField] private Transform Start;
    [SerializeField] private Transform End;
    //List of points describing the path an android can take through the custom tile
    //during pathfinding
    [SerializeField] private Transform[] ExitToStartPath;

    [HideInInspector] public Vector3Int localExitPos;

    private Vector3Int startTilePos;
    private Vector3Int endTilePos;

    public void Init(Vector3Int pos, Vector3Int _parentKey, Vector2 tileSize, Vector3Int dir)
    {
        parentKey = _parentKey;

        localExitPos = new Vector3Int(Mathf.RoundToInt(Exit.localPosition.x),
        Mathf.RoundToInt(Exit.localPosition.y), Mathf.RoundToInt(Exit.localPosition.z));
        startTilePos = new Vector3Int(Mathf.RoundToInt(Start.localPosition.x),
        Mathf.RoundToInt(Start.localPosition.y), Mathf.RoundToInt(Start.localPosition.z));
        endTilePos = new Vector3Int(Mathf.RoundToInt(End.localPosition.x),
        Mathf.RoundToInt(End.localPosition.y), Mathf.RoundToInt(End.localPosition.z));

        localExitPos /= 5;
        startTilePos /= 5;
        endTilePos /= 5;

        ChangeDir(dir);

        startTilePos += pos;
        endTilePos += pos;

        transform.localPosition = new Vector3(pos.x * tileSize.x, pos.y * tileSize.y,
        pos.z * tileSize.x);
        transform.localScale = new Vector3(tileSize.x, tileSize.y, tileSize.x) / 5;
    }

    void ChangeDir(Vector3Int new_dir)
    {
        transform.rotation = Quaternion.Euler(0, 0, 0);

        if (new_dir == Vector3Int.back)
            transform.rotation = Quaternion.Euler(0, 180, 0);
        else if (new_dir == Vector3Int.left)
            transform.rotation = Quaternion.Euler(0, -90, 0);
    }
}

```

```

        else if (new_dir == Vector3Int.right)
            transform.rotation = Quaternion.Euler(0, 90, 0);

        RotateAroundOrigin(ref localExitPos, new_dir);
        RotateAroundOrigin(ref startTilePos, new_dir);
        RotateAroundOrigin(ref endTilePos, new_dir);
    }

    void RotateAroundOrigin(ref Vector3Int point, Vector3Int dir)
    {
        if (dir == Vector3Int.back)
            point = new Vector3Int(-point.x, point.y, -point.z);
        else if (dir == Vector3Int.left)
            point = new Vector3Int(-point.z, point.y, point.x);
        else if (dir == Vector3Int.right)
            point = new Vector3Int(point.z, point.y, -point.x);
    }

    public void AddPlaceholders(ref Dictionary<Vector3Int, Tile> TileMap)
    {
        //Every coordinate tile within the bounding box of the custom tile should be
        overwritten with null as a placeholder
        for (int x = Mathf.Min(startTilePos.x, endTilePos.x); x <=
        Mathf.Max(startTilePos.x, endTilePos.x); x++)
        {
            for (int y = Mathf.Min(startTilePos.y, endTilePos.y); y <=
            Mathf.Max(startTilePos.y, endTilePos.y); y++)
            {
                for (int z = Mathf.Min(startTilePos.z, endTilePos.z); z <=
                Mathf.Max(startTilePos.z, endTilePos.z); z++)
                {
                    if (TileMap.ContainsKey(new Vector3Int(x, y, z)))
                    {
                        if (TileMap[new Vector3Int(x, y, z)] != null)
                        {
                            //Remove all walls from any tiles in the way so they don't
                            render
                            foreach (Wall wall in TileMap[new Vector3Int(x, y,
                            z)].Walls.Values)
                                wall.wallState.SetState(Wall.State.removed);

                            TileMap[new Vector3Int(x, y, z)] = null;
                        }
                    }
                    else
                        TileMap.Add(new Vector3Int(x, y, z), null);
                }
            }
        }
    }

    public Vector3[] GetPath()
    {
        List<Vector3> path = new List<Vector3>();
        foreach (Transform point in ExitToStartPath)
            path.Add(point.position);
        return path.ToArray();
    }
}

```

Dialogue.cs

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

[System.Serializable]
public class Dialogue
{
    public Line[] lines;

    [System.Serializable]
    public struct Line
    {
        public float startup;

        public Text a;
        public Text b;
        public Text c;
    }

    [System.Serializable]
    public struct Text
    {
        public string text;
        public string style;
    }

    [System.Serializable]
    public struct Letter
    {
        public string character;
        public float delay;
        public Color colour;
        public Vector4 freqSpeedAmpAmp;
        public Vector2 fadeVelTime;

        public Letter(Text textAbstract)
        {
            TextAsset asset = Resources.Load(textAbstract.style) as TextAsset;
            string json = asset.text;

            Letter detail = JsonUtility.FromJson<Letter>(json);

            character = null;
            delay = detail.delay;
            colour = detail.colour;
            freqSpeedAmpAmp = detail.freqSpeedAmpAmp;
            fadeVelTime = detail.fadeVelTime;
        }
        public Letter(char _character, Letter detail, float addDelay)
        {
            character = _character.ToString();
            delay = detail.delay + addDelay;
            colour = detail.colour;
            freqSpeedAmpAmp = detail.freqSpeedAmpAmp;
            fadeVelTime = detail.fadeVelTime;
        }
    }
}

public static Dialogue Load(string filename)
{
    TextAsset asset = Resources.Load(filename) as TextAsset;
    string json = asset.text;
```

```

        return JsonUtility.FromJson<Dialogue>(json);
    }

    public List<Letter[]> GetLetters() //line<sentence[letter]>
    {
        List<Letter[]> letters = new List<Letter[]>();

        //Each line is composed of three phrases / sentences each with their own mood
[a,b,c]
        foreach (Line line in lines)
        {
            Letter[] sentence = new Letter[line.a.text.Length +
(line.b.Equals(default(Text)) ? 0 : line.b.text.Length) +
(line.c.Equals(default(Text)) ? 0 : line.c.text.Length)];

            Letter aDetail = new Letter(line.a);
            Letter bDetail = line.b.Equals(default(Text)) ? new Letter() : new
Letter(line.b);
            Letter cDetail = line.c.Equals(default(Text)) ? new Letter() : new
Letter(line.c);

            int i = 0;
            float addDelay = 0;
            foreach (char character in line.a.text)
            {
                sentence[i] = new Letter(character, aDetail, addDelay);
                i++;
            }
            addDelay += aDetail.delay;
            if (!line.b.Equals(default(Text)))
            {
                foreach (char character in line.b.text)
                {
                    sentence[i] = new Letter(character, bDetail, addDelay);
                    i++;
                }
            }
            addDelay += bDetail.delay;
            if (!line.c.Equals(default(Text)))
            {
                foreach (char character in line.c.text)
                {
                    sentence[i] = new Letter(character, cDetail, addDelay);
                    i++;
                }
            }

            letters.Add(sentence);
        }
        return letters;
    }
}

```

Dijkstras.cs

```

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

public class Dijkstras
{
    private int recursionLimit = 9999;
    private int recursionCount = 0;
    struct NodeInfo
    {
        public bool visited;
        public float shortest;
        public int parent;
    }

    int numNodes;
    float[,] adjMatrix;

    public Dijkstras(Vector3[] nodes, float maxDist)
    {
        numNodes = nodes.Length;
        CreateAdjMatrix(nodes, maxDist);
    }
    void CreateAdjMatrix(Vector3[] nodes, float maxDist)
    {
        float maxSqrDist = maxDist * maxDist;
        adjMatrix = new float[numNodes, numNodes];
        for (int i = 0; i < numNodes; i++)
        {
            for (int j = 0; j < numNodes; j++)
            {
                if (j > i)
                {
                    //Incentivise shorter distances with square distance
                    float sqrDist = (nodes[i] - nodes[j]).sqrMagnitude;
                    if (sqrDist < maxSqrDist)
                    {
                        adjMatrix[i, j] = sqrDist;
                    }
                    else
                        adjMatrix[i, j] = float.MaxValue;
                }
                else if (j == i)
                    adjMatrix[i, j] = float.MaxValue;
                else
                    adjMatrix[i, j] = adjMatrix[j, i];
            }
        }
    }
    public void ShortestPath(int from, int to, out List<int> path, out float pathLength)
    {
        //Initialise infoTable
        NodeInfo[] infoTable = new NodeInfo[numNodes];
        for (int i = 0; i < numNodes; i++)
        {
            infoTable[i] = new NodeInfo();
            infoTable[i].visited = false;
            infoTable[i].shortest = float.MaxValue;
        }
    }
}

```

```

    }

    infoTable[from].shortest = 0;
    infoTable[from].parent = -1;

    recursionCount = 0;
    CaluclatePaths(from, ref infoTable);

    path = new List<int>(numNodes);

    recursionCount = 0;
    PathTo(to, infoTable, ref path);
    pathLength = infoTable[to].shortest;
}

void CaluclatePaths(int from, ref NodeInfo[] infoTable)
{
    if (recursionCount == recursionLimit)
    {
        Debug.LogError("Error: recursion limit reached");
        return;
    }
    else if (from == -1)
    {
        Debug.LogError("Error: Graph is not connected");
        return;
    }

    infoTable[from].visited = true;

    bool allVisited = true;
    for (int i = 0; i < numNodes; i++)
    {
        if (infoTable[i].visited)
            continue;

        allVisited = false;

        //Calculate the weight of the path from the starting node to the current
node
        float arcWeight = infoTable[from].shortest + adjMatrix[from, i];
        //If this path is shorter than the current shortest path to the node,
update shortest path and parent node
        if (arcWeight < infoTable[i].shortest)
        {
            infoTable[i].shortest = arcWeight;
            infoTable[i].parent = from;
        }
    }
    if (allVisited)
        return;

    //Find the unvisited node with the shortest path from the starting node
    (int next, float minWeight) = (-1, float.MaxValue);
    for (int i = 0; i < infoTable.Length; i++)
    {
        if (!infoTable[i].visited && infoTable[i].shortest < minWeight)
            (next, minWeight) = (i, infoTable[i].shortest);
    }

    //Recursively calculate shortest paths starting from the next node
    recursionCount++;
    CaluclatePaths(next, ref infoTable);
    recursionCount--;
}

```

```

    }
    void PathTo(int child, NodeInfo[] infoTable, ref List<int> path)
    {
        if (recursionCount == recursionLimit)
        {
            Debug.LogError("Error: recursion limit reached");
            return;
        }
        //If child node is the starting node
        else if (child == -1)
            return;

        //Add current node to the path list and recursively call PathTo on the parent
        node

        path.Add(child);

        recursionCount++;
        PathTo(infoTable[child].parent, infoTable, ref path);
        recursionCount--;
    }
}

```

DoorCollide.cs

```

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

//Helper class for OpenDoor.cs
public class DoorCollide : MonoBehaviour
{
    [SerializeField] private OpenDoor doorParent;
    private bool stuck = true;
    private float timer = float.MaxValue;
    private Collider stuckWith = null;

    private void OnTriggerStay(Collider collider)
    {
        stuck = true;
        doorParent.Stuck(collider);
    }
    private void OnTriggerExit(Collider collider)
    {
        stuck = false;
        timer = Time.realtimeSinceStartup;
        stuckWith = collider;
    }

    private void FixedUpdate()
    {
        if (!stuck && Time.realtimeSinceStartup > timer + 1f)
        {
            doorParent.UnStuck(stuckWith);
            stuck = true;
        }
    }
}

```

DustFollower.cs

```

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

public class DustFollower : MonoBehaviour
{
    [SerializeField] private Planet planet;
    [SerializeField] private Transform sun;
    private float minMoveTime = 1;
    private float minMoveDist = 1;
    private float moveTimeInstance = 1;
    private float moveDistInstance = 1;
    private float distFromPlayer = 60;
    private Material dustMat;
    private MeshRenderer meshRenderer;
    private float lerp;
    private float shadeAngle;
    private bool active = false;

    // Start is called before the first frame update
    void Start()
    {
        lerp = 1;

        meshRenderer = GetComponent<MeshRenderer>();
        dustMat = new Material(meshRenderer.sharedMaterial);
        meshRenderer.sharedMaterial = dustMat;
    }

    // Update is called once per frame
    void Update()
    {
        lerp += Time.deltaTime / moveTimeInstance;

        if (!active && (planet.transform.position -
Camera.main.transform.position).sqrMagnitude < planet.planetMesh.elevationData.Max *
planet.planetMesh.elevationData.Max)
        {
            active = true;
            meshRenderer.enabled = active;
        }
        else if (active && (planet.transform.position -
Camera.main.transform.position).sqrMagnitude >= planet.planetMesh.elevationData.Max *
planet.planetMesh.elevationData.Max)
        {
            active = false;
        }
        if (!active && lerp > 1)
        {
            meshRenderer.enabled = false;
            return;
        }

        if (lerp > 1)
        {
            lerp = 0;

            moveTimeInstance = Random.Range(minMoveTime, 2 * minMoveTime);
        }
    }
}

```

```

        moveDistInstance = Random.Range(minMoveDist, 2 * minMoveDist);

        //Calculate the zone direction and position in which the dust will be
generated

        Vector3 zonePos = 0.33f * distFromPlayer * (2 *
Camera.main.transform.forward + Random.onUnitSphere) + Camera.main.transform.position;

        Vector3 zoneNormal = (zonePos - planet.transform.position).normalized;

        //Raycast to find the intersection point of the zone and the planet
surface
        Physics.Raycast(zoneNormal * planet.planetMesh.elevationData.Max +
planet.transform.position, -zoneNormal, out RaycastHit hitInfo,
planet.planetValues.radius);

        //Look towards player
        transform.rotation = Quaternion.LookRotation(hitInfo.point -
Camera.main.transform.position, zoneNormal);
        transform.position = hitInfo.point + zoneNormal * 0.2f;// + Random.value *
moveDist * transform.right;

        dustMat.SetFloat("_minElevation", planet.planetMesh.elevationData.Min);
        dustMat.SetFloat("_maxElevation", planet.planetMesh.elevationData.Max);
        dustMat.SetTexture("_planetTexture",
planet.terrainMat.GetTexture("_planetTexture"));
        dustMat.SetVector("_position", hitInfo.point - planet.transform.position);

        shadeAngle = Vector3.Dot((sun.position -
planet.transform.position).normalized, (transform.position -
planet.transform.position).normalized);
        shadeAngle = Mathf.Clamp01(Mathf.Pow(Mathf.Clamp01(shadeAngle), 0.2f) +
0.1f);
    }

    RunCycle();
}

void RunCycle()
{
    if (lerp < 0.2f)
        dustMat.SetFloat("_alphaFade", shadeAngle * 5 * lerp);
    else if (lerp > 0.8f)
        dustMat.SetFloat("_alphaFade", shadeAngle * 5 * (1 - lerp));

    transform.position += Time.deltaTime * moveDistInstance / moveTimeInstance *
transform.right;

    Debug.DrawLine(transform.position, transform.position + (transform.position -
planet.transform.position).normalized * 50, Color.blue);
}
}

```

EffectsHandler.cs

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

[CreateAssetMenu(menuName = "PostProcessing/EffectsHandler")]
public class EffectsHandler : ScriptableObject
{
    PlanetEffect[] allEffects;

    List<PlanetEffect> GetActiveEffectsByDist()
    {
        //Should contain all four planet effects
        if (allEffects == null || allEffects.Length < 4)
            allEffects = FindObjectsOfType<PlanetEffect>(true);

        List<PlanetEffect> effects = new List<PlanetEffect>();
        foreach (PlanetEffect effect in allEffects)
        {
            if (effect == null)
            {
                allEffects = FindObjectsOfType<PlanetEffect>(true);
                continue;
            }
            if (effect.active && effect.transform.parent.gameObject.activeSelf)
            {
                effect.UpdateInfo();
                effects.Add(effect);
            }
        }
        effects = effects.OrderBy(x => x.cameraSqrDist).ToList();
        effects.Reverse();
        return effects;
    }

    public virtual List<Material> GetMaterials()
    {
        List<PlanetEffect> effects = GetActiveEffectsByDist();

        List<Material> materials = new List<Material>();

        foreach (PlanetEffect _effect in effects)
            materials.Add(_effect.GetMaterial());

        return materials;
    }
}
```

ElevationData.cs

```

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

public class ElevationData
{
    public float Max { get; private set; }
    public float Min { get; private set; }

    public float Mean01 { get; private set; }
    public float lowerMean01 { get; private set; }
    public float upperMean01 { get; private set; }
    public float interMeanRange01 { get; private set; }

    private int meanCount;
    private int lowCount;
    private int upCount;

    private List<Vector3[]> vertSegments;

    public ElevationData()
    {
        Reset(float.MaxValue);
    }
    public void Reset(float _radius)
    {
        Min = _radius;
        Max = float.MinValue;
        Mean01 = 0;
        lowerMean01 = 0;
        upperMean01 = 0;
        meanCount = 0;
        lowCount = 0;
        upCount = 0;
        vertSegments = new List<Vector3[]>();
    }
    public void Add(float v)
    {
        Max = Mathf.Max(Max, v);
        if (v >= Min)
        {
            Mean01 += v;
            meanCount++;

            if (v < Mean01 / meanCount)
            {
                //Since the mean will change as adding values, this only approximately
                the mean of all the values below the mean
                lowerMean01 += v;
                lowCount++;
            }
            else
            {
                //See above) this method is better because it is significantly less
                expensive for only a slight reduction in accuracy.
                upperMean01 += v;
                upCount++;
            }
        }
    }
}

```

```
        }
    public void CalcStats()
    {
        if (Min == float.MaxValue)
            Debug.LogError("Error: Elevation Data needs to be reset");

        Mean01 = Mathf.InverseLerp(Min, Max, Mean01 / meanCount);
        lowerMean01 = Mathf.InverseLerp(Min, Max, lowerMean01 / lowCount);
        upperMean01 = Mathf.InverseLerp(Min, Max, upperMean01 / upCount);
        //If values are very similar the range can be negative so absolute value taken
below
        interMeanRange01 = Mathf.Abs(upperMean01 - lowerMean01);

        //Debug.Log("Mean: " + Mean01 + "\nL: " + lowerMean01 + "\nU: " + upperMean01
+ "\nI: " + interMeanRange01);
    }
    public void AddMap(Vector3[] verts)
    {
        vertSegments.Add(verts);
    }
    public Vector3[] GetMapTexture(int index)
    {
        return vertSegments[index];
    }
}
```

EvilBaseDoorOpen.cs

```

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

//This is the opening and closing sequence of the great trap door in the Conrol Tower
public class EvilBaseDoorOpen : MonoBehaviour
{
    [SerializeField] private FlipSwitch trigger;
    [SerializeField] private float lerpSpeed = 0.1f;
    private float lerp = 0;

    [SerializeField] private Transform[] safetyBarriers;

    [SerializeField] private Vector2 openCloseTDoorZ;
    [SerializeField] private Vector2 openCloseTDoorX;
    [SerializeField] private Vector2 topBottomStandY;
    [SerializeField] private Vector2 topBottomPillarY;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (trigger.switchState == FlipSwitch.State.top && lerp < 2.25f)
            lerp += lerpSpeed * Time.deltaTime;
        else if (trigger.switchState == FlipSwitch.State.bottom && lerp > 0)
            lerp -= lerpSpeed * Time.deltaTime;

        //start interpolates from 0 to 1 to move the lift down first, then gates
        //interpolates from 0 to 1 to open the gates, then end interpolates from 0 to 1 to close
        //the trap door
        //this process is reversable
        lerp = Mathf.Clamp(lerp, 0, 2.25f);
        float start = Mathx.EndsCommon(Mathf.Clamp01(lerp));
        float gates = Mathx.EndsCommon((Mathf.Clamp(lerp, 1, 1.25f) - 1) * 4);
        float end = Mathx.EndsCommon(Mathf.Clamp(lerp, 1.25f, 2.25f) - 1.25f);

        transform.GetChild(0).transform.localScale = new Vector3(1,
        Mathf.Lerp(topBottomPillarY.x, topBottomPillarY.y, start), 1);
        transform.GetChild(1).transform.localPosition = new Vector3(0,
        Mathf.Lerp(topBottomStandY.x, topBottomStandY.y, start), 0);

        for (int i = 0; i < safetyBarriers.Length; i++)
            safetyBarriers[i].localEulerAngles = new Vector3(90 * (i == 0 ? gates : 1 -
gates), -90, i == 0 ? 90 : -90);

        transform.GetChild(2).GetChild(0).transform.localScale = new
        Vector3(Mathf.Lerp(openCloseTDoorX.x, openCloseTDoorX.y, end), 1,
        Mathf.Lerp(openCloseTDoorZ.x, openCloseTDoorZ.y, end));
        transform.GetChild(2).GetChild(1).transform.localScale = new
        Vector3(Mathf.Lerp(openCloseTDoorX.x, openCloseTDoorX.y, end), 1, -
        Mathf.Lerp(openCloseTDoorZ.x, openCloseTDoorZ.y, end));
    }
}

```

FabrikLeg.cs

```

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

public class FabrikLeg : MonoBehaviour
{
    [SerializeField] private AudioClip carpetSound;
    [SerializeField] private Transform Chest;
    private bool stepped = false;

    [SerializeField] private Transform[] Limbs;
    [SerializeField] private Transform[] Ligaments;

    [SerializeField] private float limbLength;
    private float usedLimbLength;

    [SerializeField] private bool inferior; //we all know right is inferior to left :)
    private bool usedInferior;
    [SerializeField] private FabrikLeg superior;

    public Vector3 prevTarget;
    public Vector3 midTarget;
    public Vector3 endTarget;

    private Vector3 lerpTarget;

    private readonly float syncCycle = 1;

    public float walkCycle { get; private set; }

    [SerializeField] private float lerpSpeed;

    private int limbNum;
    public Vector3 deltaPos;

    void Start()
    {
        usedInferior = inferior;
        usedLimbLength = limbLength;

        limbNum = Limbs.Length;

        endTarget = Limbs[0].position;
        midTarget = endTarget;
        prevTarget = endTarget;
        walkCycle = 0;
    }

    void FixedUpdate()
    {
        //prev,mid,end Target updated again in CollisionDetection.GetSafeMotion()

        //Re-synchronise inferior leg with superior leg when then superior leg is
        directly below the hip stepping forwards
    }
}

```

```

        //Set the inferior leg to directly below the hip moving backwards as the
player walks
        if (usedInferior && Mathf.Abs(superior.walkCycle - syncCycle) < 0.1f)
        {
            walkCycle = 2;
            Physics.Raycast(Limbs[limbNum - 1].position, -transform.up, out RaycastHit
belowHip);

            endTarget = belowHip.point;
        }

        if (walkCycle > 2 && !stepped)
        {
            stepped = true;
            if (carpetSound != null)
                GetComponent< AudioSource >().PlayOneShot(carpetSound);
        }
        else if (walkCycle <= 2)
            stepped = false;

        TargetUpdate();

        walkCycle += Time.fixedDeltaTime * 100 / lerpSpeed;

        if (walkCycle < 1)
            lerpTarget = Vector3.Lerp(prevTarget, midTarget, walkCycle);
        else
            lerpTarget = Vector3.Lerp(midTarget, endTarget, walkCycle - 1);

        FabrikUpdate(ref Limbs, lerpTarget);
        LigamentUpdate();
    }

    void TargetUpdate()
    {
        Vector3 hipPos = Limbs[limbNum - 1].position;

        Physics.Raycast(hipPos, endTarget - hipPos, out RaycastHit newHit);
        endTarget = newHit.point;

        // If a ray cast downwards from the hip doesn't hit anything, reset the
targets to below the hip so the robot has straight legs in the air
        if (!Physics.Raycast(hipPos, -transform.up, out RaycastHit belowHip,
usedLimbLength * (limbNum - 1)))
        {
            midTarget = hipPos - transform.up * usedLimbLength * (limbNum - 1);
            endTarget = midTarget;
            usedInferior = false;
        }
        //If the end target is too far behind the hip, adjust the target to step
forward
        else if ((endTarget - hipPos).sqrMagnitude > Mathf.Pow(usedLimbLength *
(limbNum - 1) + deltaPos.magnitude, 2))
        {
            //Debug.DrawRay(hipPos, -transform.up, Color.white, 1);

            Vector3 idealTarget = endTarget + 2 * (belowHip.point - endTarget);

            RaycastHit targetInfo = new RaycastHit();

            bool flag = false;
            //Try different limb lengths until a valid target is found

```

```

        for (usedLimbLength = limbLength; usedLimbLength < limbLength * 1.25f;
usedLimbLength += limbLength * 0.05f)
    {
        //Lerp between the ideal target and the hit point below the hip to
find a target position
        for (int lerp = 0; lerp < 100; lerp++)
        {
            Vector3 targetPos = Vector3.Lerp(idealTarget, belowHip.point, lerp
/ 100f);
            //Create a target direction that points towards the target and is
biased towards the forward-down direction of the character to decrease the distance to
the floor
            Vector3 targetDir = (targetPos - hipPos).normalized * 0.75f +
(transform.forward - transform.up) * 0.25f;
            if (Physics.Raycast(hipPos, targetDir, out targetInfo,
usedLimbLength * (limbNum - 1)))
            {
                flag = true;
                //Debug.Log(lerp);
                //Debug.DrawRay(hipPos, targetDir, Color.red, 1);
                break;
            }
        }
        if (flag)
            break;
    }

    prevTarget = endTarget;
    midTarget = (belowHip.point + hipPos) / 2;
    endTarget = targetInfo.point;

    Limbs[0].transform.rotation =
Quaternion.LookRotation(Vector3.ProjectOnPlane(Chest.forward, targetInfo.normal),
targetInfo.normal);

    walkCycle = 0;
    usedInferior = inferior;
}

}

void FabrikUpdate(ref Transform[] limbs, Vector3 target)
{
    Vector3 hipPos = limbs[limbNum - 1].position;

    //Set the position of the end node to the target position
    limbs[0].transform.position = target;
    //Update all intermediate joints from end node to hip to move towards target
    for (int i = 1; i < limbNum; i++)
    {
        Vector3 u = (limbs[i].position - limbs[i - 1].position).normalized;
        limbs[i].position = limbs[i - 1].position + u * usedLimbLength;
    }
    //Set the position of the hip joint to its original position
    limbs[limbNum - 1].position = hipPos;
    //Update all intermediate joints from hip to end node to move back to hip
    for (int i = limbNum - 2; i >= 0; i--)
    {
        Vector3 u = (limbs[i].position - limbs[i + 1].position).normalized;
        limbs[i].position = limbs[i + 1].transform.position + u * usedLimbLength;
    }
}

```

```
}

void LigamentUpdate()
{
    for (int i = 0; i < Ligaments.Length; i++)
    {
        Ligaments[i].position = (Limbs[i].position + Limbs[i + 1].position) / 2;
        Ligaments[i].rotation = Quaternion.LookRotation(Limbs[i].position -
Limbs[i + 1].position) * Quaternion.Euler(90, 90, 90);
        Ligaments[i].localScale = new Vector3(Ligaments[i].localScale.x,
usedLimbLength / 2, Ligaments[i].localScale.z);
    }
}
```

Flat.cs

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

public class Flat : MonoBehaviour
{
    private static Dictionary<int, int> script2shader = new Dictionary<int, int>(20)
    {
        { 14, 0 },
        { 5, 1 },
        { 3, 2 },
        { 4, 3 },
        { 13, 4 },
        { 2, 5 },
        { 0, 6 },
        { 7, 7 },
        { 11, 8 },
        { 1, 9 },
        { 19, 10 },
        { 17, 11 },
        { 15, 12 },
        { 16, 13 },
        { 18, 14 },
        { 12, 15 },
        { 6, 16 },
        { 8, 17 },
        { 9, 18 },
        { 10, 19 }
    };

    public Transform nextGapPos;

    [SerializeField] private GameObject building;
    [SerializeField] private GameObject shelfPrefab;
    [SerializeField] private GameObject ceilLightPrefab;
    [SerializeField] private GameObject[] otherObjects;

    [SerializeField] private float generateRadius = 20;

    private bool debugMake;

    private Dictionary<Vector3, Vector3Int> objectWalls;

    private float sqrRadius;
    private bool generated = false;
    private bool init = false;
    private Vector3Int localGapPos;
    private float rot;
    private float seed;

    public void Init(int _seed, float _rot, Transform transformGapPos)
    {
        init = true;

        sqrRadius = generateRadius * generateRadius;

        rot = _rot;
```

```

objectWalls = new Dictionary<Vector3, Vector3Int>();

transformGapPos.parent = transform;
localGapPos = RoundV3(transformGapPos.transform.localPosition);

//Placeholder for flat when player is far away
Material placeholder =
transform.GetChild(0).GetChild(0).GetComponent<MeshRenderer>().sharedMaterial;
placeholder = new Material(placeholder);
seed = _seed;
placeholder.SetFloat("_seed", seed);
transform.GetChild(0).GetChild(0).GetComponent<MeshRenderer>().sharedMaterial
= placeholder;

foreach (Transform child in GetComponentsInChildren<Transform>(true))
    child.gameObject.layer = gameObject.layer;
}

public void DebugMake()
{
    debugMake = true;
    Update();
    Update();
    debugMake = false;
}

private void Update()
{
    if (!init)
    {
        Debug.LogWarning("Warning: flat not initialised");
        return;
    }

    if (debugMake || (Camera.main.transform.position -
transform.position).sqrMagnitude < sqrRadius + 100)
    {
        if (!generated)
        {
            generated = true;
            Instantiate(building, transform.position, transform.rotation,
transform).transform.SetAsFirstSibling();
            GenerateBuilding();
            GenerateProps();

            foreach (Transform child in GetComponentsInChildren<Transform>(true))
                child.gameObject.layer = gameObject.layer;
        }
        //If placeholder active and camera is close, swap placeholder for the
actual flat
        else if (transform.GetChild(1).gameObject.activeSelf)
        {
            transform.GetChild(0).gameObject.SetActive(true);
            transform.GetChild(1).gameObject.SetActive(false);
        }
    }
    //If placeholder active and camera is far, swap actual flat for the
placeholder
    else if (generated && transform.GetChild(0).gameObject.activeSelf)
    {
        transform.GetChild(0).gameObject.SetActive(false);
        transform.GetChild(1).gameObject.SetActive(true);
    }
}

```

```

        }

    }

    void GenerateBuilding()
    {
        Transform floorHolder = transform.GetChild(0).GetChild(0);
        Transform wallHolder = transform.GetChild(0).GetChild(1);

        for (int i = 0; i < floorHolder.childCount; i++)
        {
            //Remove floor tile in flat blocking enterance from stairs below
            if (RoundV3(floorHolder.GetChild(i).localPosition) == localGapPos)
                floorHolder.GetChild(i).GetChild(0).gameObject.SetActive(false);
        }
        int c = 0;
        for (int i = 0; i < wallHolder.childCount; i++)
        {
            Vector3 wallPos = RoundV3(wallHolder.GetChild(i).localPosition);
            Vector3Int wallType = GetWallType(wallHolder.GetChild(i), rot);

            //Replace wall with window if wall on outside
            if (wallHolder.GetChild(i).GetChild(0).GetChild(0).gameObject.activeSelf)
            {
                if (Mathf.Sin(seed + 10 * script2shader[c]) > 0.5f)
                {
                    wallHolder.GetChild(i).GetChild(0).gameObject.SetActive(false);
                    wallHolder.GetChild(i).GetChild(1).gameObject.SetActive(true);
                }
                wallHolder.GetChild(i).GetChild(1).name = i.ToString();
                c++;
            }

            //Decide which wall a prop will generate from onto a floor tile
            if (objectWalls.ContainsKey(wallPos))
            {
                if (Random.value < 0.5f)
                    objectWalls[wallPos] = wallType;
            }
            else
                objectWalls.Add(wallPos, wallType);
        }
    }

    void GenerateProps()
    {
        Transform floorHolder = transform.GetChild(0).GetChild(0);

        for (int i = 0; i < floorHolder.childCount; i++)
        {
            if (Random.value < 0.2f)
                Instantiate(ceilLightPrefab, floorHolder.GetChild(i).position,
floorHolder.GetChild(i).rotation, transform.GetChild(0).GetChild(2));
        }

        foreach (Vector3 pos in objectWalls.Keys)
        {
            //80% of positions on the flat shouln't have a prop, nor the position
            where a gap will be made in the floor
            if (pos == localGapPos || Random.value < 0.8f)
                continue;
        }
    }
}

```

```
//Temporary wall to spawn prop
Wall wall = new Wall(objectWalls[pos], null, shelfPrefab, otherObjects);

Transform holder = new GameObject("Object Holder").transform;

holder.parent = transform.GetChild(0).GetChild(2);
holder.localPosition = pos;
wall.SpawnObject(holder);
holder.rotation = transform.rotation * Quaternion.Euler(0, -rot, 0);
}
}

Vector3Int RoundV3(Vector3 vector3)
{
    return new Vector3Int(Mathf.RoundToInt(vector3.x / 5) * 5,
Mathf.RoundToInt(vector3.y / 5) * 5, Mathf.RoundToInt(vector3.z / 5) * 5);
}

Vector3Int GetWallType(Transform wall, float yRot)
{
    Vector3Int wallType = Vector3Int.zero;

    if (wall.name.Contains("forward"))
        wallType = Vector3Int.forward;
    else if (wall.name.Contains("back"))
        wallType = Vector3Int.back;
    else if (wall.name.Contains("left"))
        wallType = Vector3Int.left;
    else if (wall.name.Contains("right"))
        wallType = Vector3Int.right;
    else
        Debug.LogError("Error wall has invalid name: " + wall.name);

    if (yRot % 180 == 90)
    {
        int dir = yRot == 90 ? 1 : -1;

        if (wallType == Vector3Int.forward)
            wallType = Vector3Int.right * dir;
        else if (wallType == Vector3Int.right)
            wallType = Vector3Int.back * dir;
        else if (wallType == Vector3Int.back)
            wallType = Vector3Int.left * dir;
        else if (wallType == Vector3Int.left)
            wallType = Vector3Int.forward * dir;
    }
    else if (yRot == 180)
        wallType = -wallType;
    return wallType;
}
}
```

FlipSwitch.cs

```

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

public class FlipSwitch : MonoBehaviour
{
    private AudioSource liftSource;
    public enum State { top, bottom, rising, falling }
    private enum Type { innerDoor, outerDoor, wing, light, seat }
    public State switchState { get; private set; }
    private float lerp = 1;
    [SerializeField] State initialSwitchState = State.bottom;
    [SerializeField] private Material ShipLightMat;
    [SerializeField] private Light[] lights;
    [SerializeField] private ParticleSystem[] smokeEmitters;
    [SerializeField] private AudioSource smokeSource;
    [SerializeField] private FlipSwitch innerSwitch;
    [SerializeField] private float lerpSpeed = 1;
    [SerializeField] private Type switchType;

    [HideInInspector] public bool overRide = false;
    [HideInInspector] public float doorLerpOffset;
    private float smokeTimer = 6;
    private float softlockTimer = 0;
    private bool softLocked = false;

    // Start is called before the first frame update
    public void Start()
    {
        liftSource = GetComponent<AudioSource>();

        lerp = 0;
        switchState = initialSwitchState;
        transform.localRotation = switchState == State.bottom ?
        Quaternion.Euler(Vector3.zero) : Quaternion.Euler(Vector3.up * -180);

        if (switchType == Type.light)
        {
            for (int i = 0; i < lights.Length; i++)
            {
                lights[i].color = switchState == State.bottom ? Color.white :
                Color.black;
                ShipLightMat.SetFloat("_brightness", switchState == State.bottom ? 1 :
                0);
            }
        }
    }

    // Update is called once per frame
    void Update()
    {
        if (CameraState.isPaused)
            return;

        if (!liftSource.isPlaying && lerp < 1 && lerp > 0)
            PlayLift();

        if (switchType == Type.outerDoor)
    
```

```

    {
        smokeTimer += Time.deltaTime;
        smokeSource.volume = (7 - smokeTimer) / 14;
        if (smokeTimer > 6)
        {
            if (smokeEmitters[0].isPlaying)
            {
                smokeEmitters[0].Stop();
                smokeEmitters[1].Stop();
            }
        }
        else if (smokeTimer > 3)
        {
            if (smokeEmitters[0].isStopped)
            {
                smokeEmitters[0].Play();
                smokeEmitters[1].Play();
                smokeSource.Play();
            }
        }
    }

    if (lerp < 1 && (switchState == State.rising || switchState == State.falling))
    {
        transform.localRotation = Quaternion.Euler(Vector3.Slerp(Vector3.zero,
Vector3.up * -180, switchState == State.rising ? lerp : (1 - lerp)));
        lerp += Time.deltaTime * lerpSpeed;
    }
    //If switch has finished moving, set its state to top or bottom
    else
    {
        lerp = 1;
        if (switchState == State.rising)
            switchState = State.top;
        else if (switchState == State.falling)
            switchState = State.bottom;
    }

    //If seat is in the way of the switch, extra range is needed
    int maxDist = switchType == Type.seat && switchState == State.top ? 3 : 2;

    //If player is not inside the ship and switch is in bottom state, check if
softlock timer has exceeded 1 second, if so set softLocked flag to true
    if (!CameraState.withinShip && switchState == State.bottom)
    {
        if (Time.realtimeSinceStartup > softlockTimer + 1)
            softLocked = true;
    }
    else
        softlockTimer = Time.realtimeSinceStartup;

    if (switchType != Type.outerDoor &&
CameraState.CamIsInteractingW(transform.position, -transform.parent.right, maxDist,
60) || overRide)
    {
        //If the switch is Type wing then it is to be synchronised with another
switch, override this other switch so both are in the same state
        if (switchType == Type.wing && !overRide)
        {
            innerSwitch.overRide = true;
        }
    }
}

```

```

overRide = false;
if (switchState == State.bottom)
{
    lerp = 0;
    switchState = State.rising;
}
else if (switchState == State.top)
{
    lerp = 0;
    switchState = State.falling;
}
}
else if (switchType == Type.outerDoor &&
(CameraState.CamIsInteractingW(transform.position, -transform.parent.right, 2, 30) ||
softLocked))
{
    overRide = true;

    //No smoke if softlocked, since its visible through walls and you're
trapped outside
    smokeTimer = softLocked ? 6 : 0;
    softLocked = false;

    //Make sure the inner switch for the inner door is in the opposite state
to the outer switch for the outer door, so at most one door is open (otherwise there
is no air lock)
    if (switchState == State.bottom)
    {
        if (innerSwitch.switchState == State.top)
            innerSwitch.overRide = true;

        //Wait 5 seconds before beginning to open outer door, start to close
inner door
        doorLerpOffset = -5;
        innerSwitch.doorLerpOffset = 0;
    }
    else if (switchState == State.top)
    {
        if (innerSwitch.switchState == State.bottom)
            innerSwitch.overRide = true;

        //Wait 5 seconds before beginning to open inner door, start to close
outer door
        doorLerpOffset = 0;
        innerSwitch.doorLerpOffset = -5;
    }
}

//If the switch is a light type, update the light color and brightness based
on the fuel remaining and the switch state
if (switchType == Type.light)
{
    for (int i = 0; i < lights.Length; i++)
    {
        if (InventoryUI.shipFuelRemaining == 0)
        {
            lights[i].color = Color.black;
            ShipLightMat.SetFloat("_brightness", 0);
        }
        else if (switchState == State.falling)
        {
            lights[i].color = Color.Lerp(Color.black, Color.white, lerp);
        }
    }
}

```

```
        ShipLightMat.SetFloat("_brightness", lerp);
        InventoryUI.shipEngineOn01 = lerp;
    }
    else if (switchState == State.rising)
    {
        lights[i].color = Color.Lerp(Color.white, Color.black, lerp);
        ShipLightMat.SetFloat("_brightness", 1 - lerp);
        InventoryUI.shipEngineOn01 = 1 - lerp;
    }
}
}

void PlayLift()
{
    StartCoroutine(CoPlayLift());
}
IEnumerator CoPlayLift()
{
    float startVolume = liftSource.volume;
    float startPitch = liftSource.pitch;

    liftSource.volume = 0;
    liftSource.pitch = startPitch + (lerp > 0.5f ? 0.1f : -0.1f);
    liftSource.Play();

    while (liftSource.volume < startVolume)
    {
        liftSource.volume += startVolume * Time.deltaTime * 5;
        yield return new WaitForEndOfFrame();
    }

    //Wait until switch has finished rotating before turning "lift" noise off
    yield return new WaitUntil(new System.Func<bool>(() => lerp <= 0 || lerp >=
1));

    while (liftSource.volume > 0)
    {
        liftSource.volume -= startVolume * Time.deltaTime * 5;
        yield return new WaitForEndOfFrame();
    }

    liftSource.Stop();
    liftSource.volume = startVolume;
    liftSource.pitch = startPitch;
}
}
```

Helipad.cs

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

public class Helipad : MonoBehaviour
{
    private ShipWeight shipWeight;

    // Start is called before the first frame update
    void Start()
    {
        shipWeight = FindObjectOfType<ShipWeight>();
    }

    // Update is called once per frame
    void Update()
    {
        //If this class has been instantiated then the ship is either in the air or on
        //the helipad
        //If this is the case we want the ship to perform collision detection even if the
        player is not in the ship as the evilBase lift may move down beneath the ship
        shipWeight.onHelipad = true;
    }
}
```

HiveGen.cs

```

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

public class HiveGen : MonoBehaviour
{
    [Range(0,1)]
    public float corridorStraight = 0.8f;
    public int corridorLength = 25;
    public int corridors = 5;

    private int corridorCountForName;

    public Vector2 tileSize;
    public Tile tilePrefab;
    public CustomTile[] customTilePrefabs;
    public CustomTile enteranceTile;
    public float[] customSpawnProb;

    public Dictionary<Vector3Int, Tile> TileMap;
    public Dictionary<Vector3Int, CustomTile> CustomTileMap;

    [SerializeField] private ShipDoorOpen trigger;
    private bool comeFromTower = true;

    void Update()
    {
        if (!CameraState.inHive)
            return;

        if (!comeFromTower && trigger.lerp < 0)
            CameraState.inHive = false;
        else if (trigger.lerp > 0)
            comeFromTower = false;
    }

    // Start is called before the first frame update
    public void Start()
    {
        Random.initState((transform.root.TryGetComponent(out Planet planet) ?
planet.planetValues.environmentSeed : System.Environment.TickCount) +
Mathf.RoundToInt(transform.eulerAngles.sqrMagnitude));

        Quaternion rotation = transform.rotation;
        transform.rotation = Quaternion.identity;

        Clear();

        if (customSpawnProb.Length != customTilePrefabs.Length)
            Debug.LogError("Error: Each custom prefab must have an associated spawn probability");

        TileMap = new Dictionary<Vector3Int, Tile>();
        CustomTileMap = new Dictionary<Vector3Int, CustomTile>();

        Tile first = NewTile(Vector3Int.left, Vector3Int.left);
        trigger = NewCustomTile(Vector3Int.zero, Vector3Int.zero,
enteranceTile.transform.GetChild(1).GetChild(0).GetComponent<ShipDoorOpen>());
    }
}

```

```

first.Walls[-Vector3Int.left].wallState.SetState(Wall.State.removed);

//Generate the corridors
for (int i = 0; i < corridors; i++)
{
    corridorCountForName = i;
    Vector3Int[] allPos = new Vector3Int[TileMap.Count];
    TileMap.Keys.CopyTo(allPos, 0);

    Vector3Int randPos = Vector3Int.zero;
    Tile randTile = null;
    Vector3Int randDir = Vector3Int.zero;

    //Choose a random tile and direction
    while (randDir == Vector3Int.zero)
    {
        randPos = allPos[Random.Range(0, TileMap.Count)];

        randTile = TileMap[randPos];

        if (randTile == null)
            continue;

        List<Vector3Int> removed = new List<Vector3Int>(4);

        if (!randTile.Walls[Vector3Int.forward].wallState.removed)
            removed.Add(Vector3Int.forward);
        else if (!randTile.Walls[Vector3Int.back].wallState.removed)
            removed.Add(Vector3Int.back);
        else if (!randTile.Walls[Vector3Int.left].wallState.removed)
            removed.Add(Vector3Int.left);
        else if (!randTile.Walls[Vector3Int.right].wallState.removed)
            removed.Add(Vector3Int.right);

        if (removed.Count > 0)
            randDir = removed[Random.Range(0, removed.Count)];
    }

    Tile starterTile = NewTileInDir(randDir, ref randPos, randTile);

    //Add tiles to the new corridor
    AddCorridor(randPos, randDir, starterTile, corridorLength);
}

Cleanup();
Render();

transform.rotation = rotation;

Random.InitState(System.Environment.TickCount);
}

public void Clear()
{
    if (Application.isEditor)
        DestroyImmediate(transform.GetChild(0).gameObject);
    else
        Destroy(transform.GetChild(0).gameObject);

    GameObject newHolder = new GameObject("Holder");
    newHolder.transform.parent = transform;
    newHolder.transform.SetAsFirstSibling();
}

```

```

        newHolder.transform.localRotation = Quaternion.identity;
        newHolder.transform.localPosition = Vector3.zero;
    }
    Tile NewTile(Vector3Int pos, Vector3Int dir)
    {
        if (TileMap.TryGetValue(pos + dir, out Tile tile))
            return tile;

        GameObject tileObject = Instantiate(tilePrefab.gameObject,
transform.GetChild(0));
        tileObject.name = "Tile of corridor: " + corridorCountForName;

        tile = tileObject.GetComponent<Tile>();

        tile.Init(pos + dir, pos, tileSize);

        TileMap.Add(pos + dir, tile);

        return tile;
    }
    CustomTile NewCustomTile(Vector3Int pos, Vector3Int dir, CustomTile
customTilePrefab)
    {
        GameObject customTileObject = Instantiate(customTilePrefab.gameObject,
transform.GetChild(0));
        customTileObject.name = "Custom Tile of corridor: " + corridorCountForName;

        CustomTile customTile = customTileObject.GetComponent<CustomTile>();

        customTile.Init(pos + dir, pos, tileSize, dir);

        return customTile;
    }
    Tile NewTileInDir(Vector3Int dir, ref Vector3Int pos, Tile tile)
    {
        Tile new_tile = NewTile(pos, dir);

        pos += dir;

        //If the tile at that position is a "custom" tile, it cannot be overwritten
        if (new_tile == null)
            return null;

        //If the previous tile is not a placeholder, remove the wall of it towards the
        new tile in the specified direction
        if (tile != null)
            tile.Walls[dir].wallState.SetState(Wall.State.removed);

        //Remove the wall of the new tile towards the previous tile in the opposite
        direction
        new_tile.Walls[-dir].wallState.SetState(Wall.State.removed);

        return new_tile;
    }

    void NewCustomTileInDir(Vector3Int dir, ref Vector3Int pos, Tile tile, CustomTile
customTilePrefab)
    {
        CustomTile customTile = NewCustomTile(pos, dir, customTilePrefab);

        pos += dir;
    }
}

```

```

        if (tile != null)
            tile.Walls[dir].wallState.SetState(Wall.State.removed);

        //Add placeholders over the custom tile to occupy the space on the TileMap
        //that the customTile occupies in the game
        customTile.AddPlaceholders(ref TileMap);

        pos += customTile.localExitPos;

        if (!CustomTileMap.ContainsKey(pos))
            CustomTileMap.Add(pos, customTile);
    }

    void AddCorridor(Vector3Int pos, Vector3Int dir, Tile tile, int length)
    {
        //If the desired length is reached or there is no tile to connect to, return
        if (length == 0 || tile == null)
            return;

        Tile new_tile = null;

        bool custom = false;

        for (int i = 0; i < customTilePrefabs.Length; i++)
        {
            if (Random.value < customSpawnProb[i])
            {
                //If a custom tile is spawned, add it to the corridor and set the new
                //tile to the next regular tile in the corridor
                NewCustomTileInDir(dir, ref pos, tile,
customTilePrefabs[Random.Range(0, customTilePrefabs.Length)]);
                new_tile = NewTileInDir(dir, ref pos, null);
                custom = true;
                break;
            }
        }
        if (!custom)
        {
            dir = RandomBiasDir(dir, corridorStraight);
            new_tile = NewTileInDir(dir, ref pos, tile);
        }

        //The function is called recursively until the desired length is reached
        AddCorridor(pos, dir, new_tile, length - 1);
    }

    Vector3Int RandomBiasDir(Vector3Int biasDir, float strength)
    {
        if (biasDir != Vector3Int.forward && biasDir != Vector3Int.back && biasDir != Vector3Int.left && biasDir != Vector3Int.right)
        {
            Debug.LogError("Error: invalid direction");
            return Vector3Int.zero;
        }
        else if (Random.value < strength)
            return biasDir;
        //Return a random vector perpendicular to the input direction vector if the
        //random value is greater than or equal to the strength
        else if (biasDir == Vector3Int.forward || biasDir == Vector3Int.back)
            return Random.value < 0.5f ? Vector3Int.right : Vector3Int.left;
        else
            return Random.value < 0.5f ? Vector3Int.forward : Vector3Int.back;
    }
}

```

```

    }

    void Cleanup()
    {
        Vector3Int[] dirs = new Vector3Int[4] { Vector3Int.forward, Vector3Int.back,
Vector3Int.left, Vector3Int.right };

        //For each tile in the map, set the wall state between the current tile all
        tiles adjacent to "removed"
        foreach (Vector3Int pos in TileMap.Keys)
        {
            foreach (Vector3Int dir in dirs)
            {
                if (TileMap.ContainsKey(pos + dir) && TileMap[pos] != null)
                    TileMap[pos].Walls[dir].wallState.SetState(Wall.State.removed);
            }
        }
    }

    void Render()
    {
        foreach (Tile tile in transform.GetChild(0).GetComponentsInChildren<Tile>())
            tile.Render();
    }

    private List<Vector3> PathToOrigin(Vector3Int child)
    {

        List<Vector3> path = new List<Vector3>();
        path.Add(TileMap[child].transform.position);

        while (child != Vector3Int.left + Vector3Int.left)
        {
            if (!TileMap.ContainsKey(child))
                break;

            if (TileMap[child] == null)
            {
                if (!CustomTileMap.ContainsKey(child))
                    break;

                foreach (Vector3 position in CustomTileMap[child].GetPath())
                    path.Add(position);
            }

            path.Add(CustomTileMap[child].transform.position);
            child = CustomTileMap[child].parentKey;
        }
        else
        {
            path.Add(TileMap[child].transform.position);
            child = TileMap[child].parentKey;
        }
    }

    path.Add(TileMap[Vector3Int.left + Vector3Int.left].transform.position);

    return path;
}

public bool PathBetweenPoints(Vector3 start, Vector3 end, out List<Vector3>
worldPath)

```

```

{
    Vector3Int tileClosestToStart = Vector3Int.zero;
    Vector3Int tileClosestToEnd = Vector3Int.zero;
    float minStartSqrDist = float.MaxValue;
    float minEndSqrDist = float.MaxValue;
    foreach (Vector3Int key in TileMap.Keys)
    {
        if (TileMap[key] == null)
            continue;

        float sqrStartDist = (TileMap[key].transform.position -
start).sqrMagnitude;
        if (sqrStartDist < minStartSqrDist)
        {
            minStartSqrDist = sqrStartDist;
            tileClosestToStart = key;
        }

        float sqrEndDist = (TileMap[key].transform.position - end).sqrMagnitude;
        if (sqrEndDist < minEndSqrDist)
        {
            minEndSqrDist = sqrEndDist;
            tileClosestToEnd = key;
        }
    }

    List<Vector3> startToOrigin = PathToOrigin(tileClosestToStart);
    List<Vector3> endToOrigin = PathToOrigin(tileClosestToEnd);
    worldPath = new List<Vector3>(startToOrigin.Count);

    bool success = true;
    int index = -1;

    //Add each point from the start point until the common point to the path
    foreach (Vector3 point in startToOrigin)
    {
        if (endToOrigin.Contains(point))
        {
            index = endToOrigin.FindIndex(x => x == point);
            break;
        }

        worldPath.Add(point);
    }

    //If there is no common point between the startToOrigin and endToOrigin lists
    then there is no path between the start and end
    if (index == -1)
    {
        success = false;
        index = 0;
    }

    //Add each point from the common point to the end point to the path
    for (int i = index; i >= 0; i--)
        worldPath.Add(endToOrigin[i]);

    for (int i = 1; i < worldPath.Count; i++)
        Debug.DrawLine(worldPath[i - 1], worldPath[i], Color.red,
Time.smoothDeltaTime);
}

```

```
        return success;
    }
}
```

InteractDesktop.cs

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

//Props generated on the shelf desktop
public class InteractDesktop : MonoBehaviour
{
    private void Update()
    {
        if (!CameraState.CamIsInteractingW(transform.position, 10))
            return;

        for (int i = 0; i < transform.childCount; i++)
        {
            if (transform.GetChild(i).gameObject.activeSelf &&
CameraState.CamIsInteractingW(transform.GetChild(i).position, transform.forward, 2.5f,
60))
            {
                InventoryUI.robotMetalCount += int.Parse(transform.GetChild(i).name);
                transform.GetChild(i).gameObject.SetActive(false);
                break;
            }
        }
    }
}
```

Interface.cs

```

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

public class Interface : MonoBehaviour
{
    private List<string> commands = new List<string>() { "HELP", "EXIT", "CLEAR",
"OPEN *FILENAME*", "LIST" };
    private Dictionary<string, Color> filenames = new Dictionary<string, Color>() { {
"HISTORY.LOG", Color.white } };

    private Queue<string> commandHistory = new Queue<string>(10);

    private Tetris tetris;

    [SerializeField] private Material ScreenMat;
    [SerializeField] private Material TetrisMat;
    [SerializeField] private Texture2D font;
    private string row0 = "0123456789 ";
    private string row1 = "abcdefghijklmnopqrstuvwxyz";
    private string row2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    private string row3 = "!*()-_=+;:'@\\|,<.>/?";
    [SerializeField] private int pixelSize = 1;
    public bool shipComputer = false;
    private Texture2D screen;

    private float timeSinceKeyPress;
    private bool executing = false;

    private float deltaTime;
    private int upBack;

    private int lineNo = 0;
    private int maxLines;
    private string sentence = "";

    [SerializeField] private bool interacting = false; //need this as otherwise
unlocking script here will run even if not camera interacted with this interface

    Texture2D SentenceTexture(string word)
    {
        Texture2D wordTexture = new Texture2D(6 * pixelSize * word.Length, 6 *
pixelSize);

        int i = 0;

        if (word.Length == 0)
            return null;

        foreach (char character in word)
        {
            //Get the pixel coordinates in the font texture for the current character
            CharIndex(out int fontX, out int fontY, character);

            int xOffset = i * 6 * pixelSize;

            //Copy the pixels for the character from the font texture to the output
            texture
            for (int x = 0; x < 6; x++)
            {
                for (int y = 0; y < 6; y++)

```

```

        {
            for (int x0 = 0; x0 < pixelSize; x0++)
            {
                for (int y0 = 0; y0 < pixelSize; y0++)
                {
                    wordTexture.SetPixel(xOffset + x * pixelSize + x0, y *
pixelSize + y0, font.GetPixel(fontX + x, fontY + y));
                }
            }
        }

        i++;
    }
    wordTexture.Apply();
    return wordTexture;
}

bool CharIndex(out int fontX, out int fontY, char character)
{
    if (row0.Contains(character.ToString()))
    {
        fontY = 0;
        fontX = row0.IndexOf(character) * 6;
    }
    else if (row1.Contains(character.ToString()))
    {
        fontY = 6;
        fontX = row1.IndexOf(character) * 6;
    }
    else if (row2.Contains(character.ToString()))
    {
        fontY = 12;
        fontX = row2.IndexOf(character) * 6;
    }
    else if (row3.Contains(character.ToString()))
    {
        fontY = 18;
        fontX = row3.IndexOf(character) * 6;
    }
    else
    {
        fontY = 0;
        fontX = row0.IndexOf(' ') * 6;
        return false;
    }
    return true;
}

void Write(string sentence)
{
    Write(sentence, Color.white);
}

void Write(string sentence, Color textColour)
{
    //If the number of lines displayed is already at the maximum allowed, the
    screen is scrolled down to make room for the new line.
    if (lineNo >= maxLines)
    {
        int scrollDown = (lineNo - maxLines + 1) * 6 * pixelSize;
        for (int x = 0; x < screen.width; x++)

```

```

        {
            for (int y = screen.height - 1; y >= 0; y--)
            {
                if (y - scrollDown > 0)
                    screen.SetPixel(x, y, screen.GetPixel(x, y - scrollDown));
                else
                    screen.SetPixel(x, y, Color.black);
            }
        }
        lineNo = maxLines - 1;
    }

    Texture2D addition = SentenceTexture(sentence);

    //Write the sentence line at the current lineNo
    for (int x = 0; x < addition.width; x++)
    {
        for (int y = 0; y < addition.height; y++)
        {
            screen.SetPixel(x, screen.height - (lineNo + 1) * addition.height + y
+ 1, Color.Lerp(textColour, Color.black, addition.GetPixel(x, y).grayscale));
        }
    }
}

private void ResetScreen()
{
    screen = new Texture2D(240, 135);
    for (int x = 0; x < screen.width; x++)
    {
        for (int y = 0; y < screen.height; y++)
        {
            screen.SetPixel(x, y, Color.black);
        }
    }
}

private bool ExecuteCommand(ref string sentence)
{
    if (!executing)
        lineNo++;

    if (sentence == "")
        return false;
    //HELP
    else if (sentence == "?" || sentence.ToUpper() == "HELP")
    {
        lineNo++; Write("COMMANDS:", Color.yellow);
        lineNo++;
        foreach (string command in commands)
        {
            lineNo++; Write(command);
        }
        lineNo += 2;
        return false;
    }
    //EXIT
    else if (sentence.ToUpper() == "EXIT")
    {
        lineNo++; Write("GOODBYE", Color.yellow);
        lineNo++;
        interacting = false;
    }
}

```

```

        CameraState.UnlockCamera();
        return false;
    }
    //CLEAR
    else if (sentence.ToUpper() == "CLEAR")
    {
        lineno = 0;
        ResetScreen();
        return false;
    }
    //LIST
    else if (sentence.ToUpper() == "LIST")
    {
        lineno++; Write("FILES ON COMPUTER:", Color.yellow);
        lineno++;
        foreach (string filename in filenames.Keys)
        {
            lineno++; Write(filename, filenames[filename]);
        }
        lineno += 2;
        return false;
    }
    //OPEN
    else if (sentence.Length >= 4 && sentence.Substring(0, 4).ToUpper() == "OPEN")
    {
        if (!(sentence.Length > 5 && sentence.Substring(0, 5).ToUpper() == "OPEN"))
        {
            lineno++; Write("ERROR: PLEASE PROVIDE PARAMETER", Color.red);
            lineno++; Write("*FILENAME* AFTER A SPACE", Color.red);
            lineno++;
            return false;
        }

        string[] _files = new string[filenames.Count];
        filenames.Keys.CopyTo(_files, 0);
        List<string> files = new List<string>(_files.Length);
        foreach (string _file in _files)
            files.Add(_file);

        string filename = sentence.Substring(5).ToUpper();
        if (!files.Contains(filename))
        {
            lineno++; Write("ERROR: FILE DOES NOT EXIST", Color.red);
            lineno++;
            return false;
        }
        lineno++; Write("FILE OPENED", Color.green);

        if (filename == "HISTORY.LOG")
        {
            lineno++; Write("PREVIOUS COMMANDS:", Color.yellow);
            lineno++;
            foreach (string command in commandHistory)
            {
                lineno++; Write(command);
            }
        }
        else if (filename == "BLUEPRINTS.PDF")
        {
            lineno++; Write("BLUEPRINT EXTRACTED", Color.blue);
            lineno++; Write("UPGRADE POTENTIAL ACQUIRED", Color.blue);
        }
    }
}

```

```

        InventoryUI.robotMetalCount += 100;
        filenames.Remove("BLUEPRINTS.PDF");
    }
    else if (filename == "TETRIS.EXE")
    {
        lineNumber += 2;
        Write("CONTROLS:", Color.yellow);
        lineNumber++; Write("W = HARD DROP");
        lineNumber++; Write("S = SOFT FALL");
        lineNumber++; Write("A/D = LEFT/RIGHT");
        lineNumber++; Write("ARROW KEYS = ROTATE");
        lineNumber++; Write("SHIFT = HOLD PIECE");
        lineNumber += 2;
        tetris = new Tetris();
        tetris.GameStart(TetrisMat, font);
        CameraState.playingTetris = true;
        sentence = "$TETRIS_INIT";
        return true;
    }
    lineNumber += 2;
    return false;
}
else if (sentence == "$TETRIS" || sentence == "$TETRIS_INIT")
{
    if (sentence == "$TETRIS_INIT")
    {
        Write("PRESS ENTER TO START:" + (Mathf.FloorToInt(deltaT * 2) % 2 == 0
? "_" : " "), Color.yellow);

        if (Input.GetKeyDown(KeyCode.Return))
        {
            Write("PRESS ENTER TO START: ", Color.yellow);
            sentence = "$TETRIS";
            GetComponent<MeshRenderer>().sharedMaterial = tetris.displayMat;
        }
        else
            return true;
    }
    tetris.GameFixedUpdate();
    tetris.GameUpdate();

    if (tetris.gameOver)
    {
        GetComponent<MeshRenderer>().sharedMaterial = ScreenMat;
        lineNumber += 2;
        Write("GAMEOVER. FINAL SCORE:" + tetris.scoreValue, Color.magenta);
        lineNumber += 2;
        CameraState.playingTetris = false;
        return false;
    }
    return true;
}
else if (sentence.ToUpper() == "MAP_PLANET")
{
    Weight sigWeight = FindObjectOfType<RobotWeight>().sigWeight;
    if (sigWeight == null)
    {
        lineNumber += 2;
        Write("ERROR: NO PLANET IN RANGE", Color.red);
        lineNumber += 2;
        return false;
    }
}

```

```

    {
        lineNo++; Write("ERROR: PLEASE PROVIDE PARAMETERS", Color.red);
        lineNo++; Write("*TYPE* *AMOUNT* EACH AFTER A SPACE", Color.red);
        lineNo++; Write("TYPES AVAILABLE: METAL, FUEL");
        lineNo++;
        return false;
    }
    string type = sentence.Substring(8).Split(' ')[0];
    if (!int.TryParse(sentence.Substring(8).Split(' ')[1], out int amount) ||
amount <= 0)
    {
        lineNo++; Write("ERROR: *AMOUNT* MUST BE WHOLE + POSITIVE",
Color.red);
        lineNo++;
        return false;
    }
    if (type == "METAL")
    {
        //Clamp metal amount so that values do not go below zero
        amount = Mathf.Min(amount, Mathf.FloorToInt(withdraw ?
InventoryUI.shipMetalCount : InventoryUI.robotMetalCount));
        InventoryUI.robotMetalCount += amount * (withdraw ? 1 : -1);
        InventoryUI.shipMetalCount += amount * (withdraw ? -1 : 1);
        JsonSaver.SaveData("Ship_Metal_Count", new
InventoryUI.ShipMetalCountWrap(Mathf.FloorToInt(InventoryUI.shipMetalCount)));
        lineNo++; Write("ADVICE: KEEP METAL SAFE ON SHIP", withdraw ?
Color.red : Color.green);
        lineNo++;
    }
    else if (type == "FUEL")
    {
        //Clamp fuel amount so that values do not go below zero or above
capacities
        amount = Mathf.Min(amount, Mathf.FloorToInt(withdraw ?
Mathf.Min(InventoryUI.player.Stat("robotFuelCapacity"), InventoryUI.shipFuelRemaining) :
InventoryUI.fuelRemaining));
        amount = Mathf.Min(amount, Mathf.FloorToInt(withdraw ?
InventoryUI.player.Stat("robotFuelCapacity") - InventoryUI.fuelRemaining :
InventoryUI.player.Stat("shipFuelCapacity") - InventoryUI.shipFuelRemaining));
        InventoryUI.fuelRemaining += amount * (withdraw ? 1 : -1);
        InventoryUI.shipFuelRemaining += amount * (withdraw ? -1 : 1);
    }
    else
    {
        lineNo++; Write("ERROR: TYPE DOES NOT EXIST", Color.red);
        lineNo++;
        return false;
    }
    lineNo++; Write((withdraw ? "WITHDRAWN" : "DEPOSITED") +
amount.ToString() + " " + type, Color.yellow);
    lineNo++;
    return false;
}
else
{
    Write("ERROR: COMMAND NOT FOUND", Color.red);
    lineNo++; Write("ENTER *HELP* TO LIST AVAILABLE COMMANDS", Color.yellow);
    lineNo++;
    return false;
}

```

```

    }

    private void UserInput()
    {
        deltaT = Time.realtimeSinceStartup - timeSinceKeyPress;

        string ticker = (Mathf.RoundToInt(2 * deltaT) % 2 == 0 && (deltaT > 0.5f) &&
!Input.GetKeyDown(KeyCode.Backspace) ? " " : "_");

        //Handle Return key press or command execution
        if (Input.GetKeyDown(KeyCode.Return) || executing)
        {
            upBack = 0;
            if (Input.GetKeyDown(KeyCode.Return) && !executing)
            {
                Write(" " + sentence + " "); Write(">>>", Color.green); //no ticker
                timeSinceKeyPress = Time.realtimeSinceStartup;
            }

            if (sentence.Length != 0 && sentence[0] != '$')
                commandHistory.Enqueue(sentence);
            if (commandHistory.Count > 10)
                commandHistory.Dequeue();

            executing = ExecuteCommand(ref sentence);

            if (!executing)
                sentence = "";
        }
        //Handle Backspace key press
        else if (Input.inputString.Length > 0 && (KeyCode)Input.inputString[0] ==
KeyCode.Backspace) //rather than Input.GetKeyDown(KeyCode.Backspace) which would remove 1
letter a frame
        {
            Write(" " + sentence + " "); Write(">>>", Color.green); //no ticker
            if (sentence.Length > 0)
                sentence = sentence.Substring(0, sentence.Length - 1);
        }
        //Handle Up/Down Arrow key press for command history navigation
        else if (Input.GetKeyDown(KeyCode.UpArrow) ||
Input.GetKeyDown(KeyCode.DownArrow))
        {
            upBack += Input.GetKeyDown(KeyCode.UpArrow) ? 1 : -1;
            upBack = Mathf.Clamp(upBack, 0, commandHistory.Count);

            string[] history = commandHistory.ToArray();
            if (history.Length >= upBack)
            {
                Write(" ");
            }
            Write(" " + history[history.Length - upBack] + " "); Write(">>>",
Color.green); //no ticker
            sentence = history[history.Length - upBack];
            timeSinceKeyPress = Time.realtimeSinceStartup;
        }
    }
    //Handle regular key press for typing
    else if (Input.inputString.Length > 0 && CharIndex(out _, out _,
Input.inputString.ToCharArray()[0]) && (sentence.Length + 4) * 6 * pixelSize <
screen.width) // + 4 for the ">>>" + ticker
    {
        upBack = 0;
    }
}

```

```

        sentence += Input.inputString.ToUpper();

        timeSinceKeyPress = Time.realtimeSinceStartup;
    }

    //Print the current sentence with the typing ticker
    if (!executing)
    {
        Write("    " + sentence + ticker); Write(">>>", Color.green);
    }
}

//For hive computers we are pretending have already been used
void AddHistory()
{
    for (int i = 0; i < 9; i++)
    {
        string command = commands[Random.Range(0, commands.Count)];
        if (command.Contains("OPEN"))
        {
            string[] files = new string[filenames.Count];
            filenames.Keys.CopyTo(files, 0);
            command = "OPEN " + files[Random.Range(0, files.Length)];
        }
        if (!commandHistory.Contains(command) && command != "EXIT")
            commandHistory.Enqueue(command);
    }
    commandHistory.Enqueue("EXIT");
}

private void Start()
{
    ScreenMat = new Material(ScreenMat);
    GetComponent<MeshRenderer>().sharedMaterial = ScreenMat;

    if (!shipComputer)
    {
        if (Random.value < 0.5f)
            filenames.Add("TETRIS.EXE", Color.white);
        if (Random.value < 0.5f)
            filenames.Add("BLUEPRINTS.PDF", Color.blue);

        AddHistory();
    }
    else
    {
        filenames.Add("TETRIS.EXE", Color.white);

        commands.Add("MAP_PLANET");
        commands.Add("PATH_TO *STAR_SYSTEM*");
        commands.Add("DEPOSIT *TYPE* *AMOUNT*");
        commands.Add("WITHDRAW *TYPE* *AMOUNT*");
    }

    ResetScreen();
    maxLines = screen.height / (pixelSize * 6);
    timeSinceKeyPress = Time.realtimeSinceStartup;

    Write("ENTER *HELP* TO LIST AVAILABLE COMMANDS", Color.yellow);
    lineNo++;
    Write("AWAITING USER INPUT...", Color.yellow);
    lineNo++;
}

```

```
        Write(">>> ", Color.green);
        screen.Apply();
        ScreenMat.SetTexture("_screen", screen);
    }

    // Update is called once per frame
    void Update()
    {
        if (!interacting && CameraState.InLockState(CameraState.LockState.unlocked) &&
CameraState.CamIsInteractingW(transform.position, -transform.up, 2, 60))
        {
            interacting = true;
            CameraState.LockCamera(transform.GetChild(0));
        }
        else if (interacting && CameraState.InLockState(CameraState.LockState.locked))
        {
            UserInput();

            screen.Apply();
            ScreenMat.SetTexture("_screen", screen);
        }
    }

    //Added as otherwise cannot interact with cli again
    public void StopInteracting()
    {
        Debug.Log("fixed");
        interacting = false;
    }
}
```

```

sigWeight.GetComponent<Planet>().GenerateMiniMap(FindObjectOfType<ShipRouter>().Map);
    lineNumber += 2;
    Write("MAP GENERATED!", Color.green);
    lineNumber += 2;
    lineNumber++; Write("                                     RED", Color.red);
Write("CURRENT POSITION MARKED IN ", Color.white);
    lineNumber++; Write("ACTIVE REGIONS HIGHLIGHTED", Color.white);
    lineNumber++; Write("RE-RUN COMMAND TO UPDATE MAP", Color.white);
    lineNumber++; Write("VIEW MAP ON SECOND MONITOR", Color.white);
    lineNumber++;
    return false;
}
else if (sentence.Length >= 7 && sentence.Substring(0, 7).ToUpper() ==
"PATH_TO")
{
    if (!(sentence.Length > 8 && sentence.Substring(0, 8).ToUpper() ==
"PATH_TO "))
    {
        lineNumber++; Write("ERROR: PLEASE PROVIDE PARAMETER", Color.red);
        lineNumber++; Write("*STAR_SYSTEM* AFTER A SPACE", Color.red);
        lineNumber++;
        return false;
    }
    else if (sentence.Length != 11)
    {
        lineNumber++; Write("ERROR: *STAR_SYSTEM* MUST BE 3 LETTERS", Color.red);
        lineNumber++;
        return false;
    }

    StarGenSystem starGenSystem = FindObjectOfType<StarGenSystem>();
    string key = sentence.Substring(8, 3);

    if (!starGenSystem.starCodes.ContainsKey(key))
    {
        lineNumber++; Write("ERROR: STAR SYSTEM " + key + " NOT FOUND",
Color.red);
        lineNumber++;
        return false;
    }

    starGenSystem.PathBetween(starGenSystem.closestStar.GetSiblingIndex(),
starGenSystem.starCodes[key], Color.green);
    lineNumber += 2;
    Write("PATH DRAWN!", Color.green);
    lineNumber++; Write("                                     RED", Color.red); Write("CUR-
RENT POSITION MARKED IN ", Color.white);
    lineNumber++; Write("PATH INCLUDES FUEL STOPS", Color.white);
    lineNumber++; Write("RE-RUN COMMAND TO UPDATE PATH", Color.white);
    lineNumber++; Write("VIEW MAP ON THIRD MONITOR", Color.white);
    lineNumber++;

    return false;
}
else if (sentence.ToUpper().Replace("WITHDRAW", "DEPOSIT").Length >= 7 && sen-
tence.ToUpper().Replace("WITHDRAW", "DEPOSIT").Substring(0, 7) == "DEPOSIT")
{
    bool withdraw = sentence.ToUpper().Contains("WITHDRAW");
    sentence = sentence.ToUpper().Replace("WITHDRAW", "DEPOSIT");
    if (!(sentence.Length > 8 && sentence.Substring(0, 8) == "DEPOSIT " &&
sentence.Substring(8).Contains(" ") && sentence.Substring(8).Split(' ').Length == 2))

```

InventoryUI.cs

```

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

[System.Serializable]
public struct PlayerLevel
{
    //All levels are 0,1,2,3
    public int robotFuelCapacityLvl;
    public int robotFuelBurnRateLvl;
    public int robotFuelPerOreLvl;

    public int robotOxygenCapacityLvl;
    public int robotOxygenTemperatureScaleLvl;
    public int robotOxygenSafeScaleLvl;

    public int shipFuelCapacityLvl;
    public int shipFuelBurnRateLvl;
    public int shipSolarChargeRateLvl;

    private Dictionary<string, float> Stats;

    private float Val(int lvl, float zero, float one, float two, float three)
    {
        return lvl == 0 ? zero : lvl == 1 ? one : lvl == 2 ? two : lvl == 3 ? three : float.NaN;
    }

    public float Stat(string stat)
    {
        if (Stats == null)
        {
            Stats = new Dictionary<string, float>(9)
            {
                { "robotFuelCapacity", Val(robotFuelCapacityLvl, 100, 110, 120, 130) },
                { "robotFuelBurnRate", Val(robotFuelBurnRateLvl, 0.25f, 0.22f, 0.2f, 0.18f) },
                { "robotFuelPerOre", Val(robotFuelPerOreLvl, 5, 10, 15, 20) },
                { "robotOxygenCapacity", Val(robotOxygenCapacityLvl, 600, 720, 840, 960) },
                { "robotOxygenTemperatureScale", Val(robotOxygenTemperatureScaleLvl, 10, 8, 6, 4) },
                { "robotOxygenSafeScale", Val(robotOxygenSafeScaleLvl, -2, -4, -5, -7) },
                { "shipFuelCapacity", Val(shipFuelCapacityLvl, 1000, 1100, 1200, 1300) },
                { "shipFuelBurnRate", Val(shipFuelBurnRateLvl, 1.67f, 1.33f, 1.17f, 1) },
                { "shipSolarChargeRate", Val(shipSolarChargeRateLvl, 4, 4.68f, 5.32f, 6.68f) }
            };
        }
        return Stats[stat];
    }
}

public class InventoryUI : MonoBehaviour

```

```

{
    [System.Serializable]
    public struct ShipMetalCountWrap
    {
        public int shipMetalCount;
        public ShipMetalCountWrap(int shipMetalCount)
        {
            this.shipMetalCount = shipMetalCount;
        }
    }

    public static float fuelRemaining;
    public static float oxygenRemaining;
    public static float shipFuelRemaining;
    public static float shipMetalCount;
    public static float robotMetalCount;
    public static float robotTemperature;
    public static PlayerLevel player;
    public static Weight shipTargetX;
    public static float shipEngineOn01;

    [SerializeField] private Slider fuel;
    [SerializeField] private Slider oxygen;
    [SerializeField] private Material fuelMat;
    [SerializeField] private Material oxygenMat;
    [SerializeField] private Material shipFuelMat;
    [SerializeField] private Material shipTempMat;

    [SerializeField] private TMPro.TextMeshProUGUI metalText;
    [SerializeField] private TextMesh shipMetalText;
    [SerializeField] private TMPro.TextMeshProUGUI warningText;
    [SerializeField] private TMPro.TextMeshProUGUI metalCounter;

    [SerializeField] private RobotWeight playerRobot;
    [SerializeField] private ShipWeight playerShip;
    [SerializeField] private StarGenSystem starGenSystem;

    private float barTurnOnLerp;
    private float chargingLerp;

    private float prevRobotMetalCount;
    private readonly string aquiredMetalMessage = "AQUIRED $ METAL";
    private float metalMessageTime;

    private readonly string oxygenDepletedMessage = "WARNING: OXYGEN DEPLETED. BURNING
FUEL FASTER";
    private float oxygenMessageTime;
    private readonly string fuelDepletedMessage = "WARNING: FUEL DEPLETED. BURNING
METAL INSTEAD";
    private float fuelMessageTime;
    private readonly string lockOnMessage = "HOLD SPACE-BAR TO MATCH VELOCITY";
    private float lockOnMessageTime;
    private readonly string liftoffMessage = "HOLD SHIFT TO LIFT-OFF";
    private float liftoffMessageTime;

    public static void Wipe()
    {
        JsonSaver.SaveData("Player_Stats", new PlayerLevel());
        JsonSaver.SaveData("Ship_Metal_Count", new ShipMetalCountWrap(0));
    }
}

```

```

// Start is called before the first frame update
public void Start()
{
    player = JsonSaver.LoadData<PlayerLevel>("Player_Stats", out bool success);
    if (!success)
        JsonSaver.SaveData("Player_Stats", player);

    shipEngineOn01 = 1;

    robotMetalCount = 0;

    shipMetalCount = JsonSaver.LoadData<ShipMetalCountWrap>("Ship_Metal_Count",
out success).shipMetalCount;
    if (!success)
        JsonSaver.SaveData("Ship_Metal_Count", new ShipMetalCountWrap(0));

    fuel.MaxValue = player.Stat("robotFuelCapacity");
    oxygen.MaxValue = player.Stat("robotOxygenCapacity");

    shipFuelMat.SetFloat("_maxShipFuel", player.Stat("shipFuelCapacity"));

    fuelMat.SetFloat("_maxSliderValue", fuel.MaxValue);
    oxygenMat.SetFloat("_maxSliderValue", oxygen.MaxValue);

    fuelRemaining = fuel.MaxValue;
    oxygenRemaining = oxygen.MaxValue;
    shipFuelRemaining = shipFuelMat.GetFloat("_maxShipFuel");
    UpdateSliders();
}

// Update is called once per frame
void Update()
{
    if (CameraState.isPaused)
        return;

    CameraState.isDead |= fuelRemaining <= 0 && robotMetalCount <= 0;

    float prevShipFuelRemaining = shipFuelRemaining;

    float sunDist10 = Mathf.InverseLerp(starGenSystem.starcoordLength,
starGenSystem.sun.minMaxDist.x, starGenSystem.sun.transform.position.magnitude);
    float solarDot01 = Mathf.Clamp01(Vector3.Dot(playerShip.transform.up *
(ShipWeight.xRotatesYaw ? -1 : 1), (starGenSystem.sun.transform.position -
playerShip.position).normalized));

    robotTemperature = 0;

    if (playerRobot.sigWeight != null)
    {
        float altitude = (playerRobot.position -
playerRobot.sigWeight.position).magnitude;

        float altitude01 =
Mathf.InverseLerp(Mathf.Max(playerRobot.sigWeight.planet.planetValues.radius,
playerRobot.sigWeight.planet.planetMesh.elevationData.Min),
playerRobot.sigWeight.planet.atmosphere.atmosRadius, altitude);

        float sunPermittivity01 = Mathf.InverseLerp(6, 0.5f,
playerRobot.sigWeight.planet.atmosphere.density) *

```

```

        Mathf.Clamp01(Vector3.Dot((playerRobot.position -
playerRobot.sigWeight.position) / altitude, (starGenSystem.sun.transform.position -
playerRobot.sigWeight.position).normalized));

        if (starGenSystem.generatedAtStar)
            shipFuelRemaining += player.Stat("shipSolarChargeRate") * solarDot01 *
sunDist10 * Mathf.Lerp(0.75f, 1, Mathf.Lerp(sunPermittivity01, 1, altitude01)) *
Time.deltaTime;

        //temperature given by planet (one minus so that the thermometer has cold
= 0)
        robotTemperature = 1 -
playerRobot.sigWeight.planet.planetValues.temperature;
        //temperature given by sunlight (varies based on atmosphere density)
        robotTemperature += 0.5f * sunPermittivity01;
        robotTemperature = Mathf.Clamp01(robotTemperature);
        //lerp temperature by altitude
        robotTemperature *= 1 - altitude01;
    }
    else
        shipFuelRemaining += player.Stat("shipSolarChargeRate") * solarDot01 *
sunDist10 * Time.deltaTime;

        float temperatureScale = Mathf.Abs(robotTemperature - 0.5f) * 2 *
player.Stat("robotOxygenTemperatureScale");
        float safeScale = CameraState.inShip || CameraState.inHive ?
player.Stat("robotOxygenSafeScale") : 1;

        RobotBreath(Time.deltaTime * temperatureScale * safeScale);

        if (!CameraState.inShip)
            RobotBurn(player.Stat("robotFuelBurnRate") * Time.deltaTime *
(Input.anyKey ? 1 : 0.9f));

        shipFuelRemaining -= player.Stat("shipFuelBurnRate") *
(!CameraState.flyingShip || !Input.anyKey ? 0.5f : ShipWeight.hyperOn ? 10 : 1) *
shipEngineOn01 * Time.deltaTime;

        shipFuelRemaining = Mathf.Clamp(shipFuelRemaining, 0,
player.Stat("shipFuelCapacity"));

        if (shipFuelRemaining == 0)
            shipEngineOn01 = 0;

        chargingLerp = Mathf.Clamp01(chargingLerp + Time.deltaTime *
(shipFuelRemaining > prevShipFuelRemaining ? 2 : -5));
        shipFuelMat.SetFloat("_charging", Mathf.Pow(chargingLerp, 2));
        //Debug.Log(shipFuelRemaining - prevShipFuelRemaining);

        UpdateSliders();
        UpdateText();
    }

    private void OnValidate()
    {
        Start();
    }

    private void UpdateSliders()
    {
        fuel.value = fuelRemaining;
        oxygen.value = oxygenRemaining;
    }
}

```

```

        shipFuelMat.SetFloat("_engineOn", shipEngineOn01);
        shipTempMat.SetFloat("_engineOn", shipEngineOn01);

        shipFuelMat.SetFloat("_shipFuel", shipFuelRemaining);
        fuelMat.SetFloat("_sliderValue", fuel.value);

        oxygenMat.SetFloat("_sliderValue", oxygen.value);

        //Turn on the oxygen and fuel bars when the player is not flying the ship
        barTurnOnLerp = Mathf.Clamp01(barTurnOnLerp + Time.deltaTime *
        (CameraState.flyingShip ? -1 : 1));
        fuelMat.SetFloat("_turnOnLerp", barTurnOnLerp);
        oxygenMat.SetFloat("_turnOnLerp", barTurnOnLerp);

        metalCounter.color = new Color(RoboVision.visionColour.r,
        RoboVision.visionColour.g, RoboVision.visionColour.b, 0.4f * barTurnOnLerp);
        warningText.color = new Color(RoboVision.visionColour.r,
        RoboVision.visionColour.g, RoboVision.visionColour.b, 0.5f);

        Color rgb = fuel.transform.GetChild(0).GetComponent<Image>().color;
        fuel.transform.GetChild(0).GetComponent<Image>().color = new Color(rgb.r,
        rgb.g, rgb.b, 0.5f * barTurnOnLerp);
        rgb = oxygen.transform.GetChild(0).GetComponent<Image>().color;
        oxygen.transform.GetChild(0).GetComponent<Image>().color = new Color(rgb.r,
        rgb.g, rgb.b, 0.5f * barTurnOnLerp);

        shipTempMat.SetFloat("_shipTemp", robotTemperature);
    }

    private void UpdateText()
    {
        UpdateAquiredMetalMessage();
        UpdateWarnings();
    }
    private void UpdateAquiredMetalMessage()
    {
        shipMetalText.text = "Metal:\n\n" +
        Mathf.FloorToInt(shipMetalCount).ToString();
        shipMetalText.color = Color.Lerp(Color.red, Color.cyan, Mathf.InverseLerp(0,
        100, shipMetalCount));
        shipMetalText.color *= Mathf.Clamp(shipEngineOn01, 0.05f, 1);

        //If no metal count has not increased this frame, do not display message
        if (prevRobotMetalCount >= robotMetalCount)
        {
            //robotMetalCount = Mathf.Max(0, robotMetalCount);
            prevRobotMetalCount = robotMetalCount;
            metalText.text = Mathf.FloorToInt(robotMetalCount).ToString();
            return;
        }

        //robotMetalCount can be updated here and the message on screen will be
        updated accordingly
        metalText.text = aquiredMetalMessage.Substring(0,
        Mathf.Min(Mathf.FloorToInt(metalMessageTime * 20),
        aquiredMetalMessage.Length)).Replace("$", Mathf.FloorToInt(robotMetalCount -
        prevRobotMetalCount).ToString());

        metalMessageTime += Time.deltaTime;
    }
}

```

```

        //If message has been displayed for more than four seconds (and message has
        been written in this time) hide the message
        if (Mathf.FloorToInt(metalMessageTime * 20) >= aquiredMetalMessage.Length &&
metalMessageTime > 4)
        {
            prevRobotMetalCount = robotMetalCount;
            metalMessageTime = 0;
        }
    }
    private void UpdateWarnings()
    {
        warningText.text = "";

        if (oxygenRemaining != 0)
            oxygenMessageTime = 0;
        else
        {
            oxygenMessageTime += Time.deltaTime;
            warningText.text += "[" + oxygenDepletedMessage.Substring(0,
Mathf.Min(Mathf.FloorToInt(oxygenMessageTime * 20), oxygenDepletedMessage.Length)) +
"]\n";
        }

        if (fuelRemaining != 0)
            fuelMessageTime = 0;
        else
        {
            fuelMessageTime += Time.deltaTime;
            warningText.text += "[" + fuelDepletedMessage.Substring(0,
Mathf.Min(Mathf.FloorToInt(fuelMessageTime * 20), fuelDepletedMessage.Length)) +
"]\n";
        }

        if (CameraState.flyingShip)
        {
            shipTargetX = playerRobot.sigWeight;

            float chosenSqrDist = 3_600_000_000; //60_000^2

            foreach (Planet planet in starGenSystem.sun.celestialBodies)
            {
                //Check if the planet is active, is closer than the current chosen
                planet, and is in front of the camera
                if (planet.gameObject.activeSelf &&
planet.transform.position.sqrMagnitude < chosenSqrDist &&
Vector3.Dot((planet.transform.position - Camera.main.transform.position).normalized,
Camera.main.transform.forward) > 0.85f)
                {
                    shipTargetX = planet.GetComponent<Weight>();
                    chosenSqrDist = planet.transform.position.sqrMagnitude;
                }
            }

            //If a valid target was found, highlight it unless the player is on the
            planet and update the lock-on message
            if (shipTargetX != null)
            {
                if (shipTargetX == playerRobot.sigWeight)
                    RoboVision.highlightBounds = new
RoboVision.TargetBounds(float.MaxValue);
                else

```

```
        RoboVision.highlightBounds = new
RoboVision.TargetBounds(shipTargetX.transform.GetChild(3));

        lockOnMessageTime += Time.deltaTime;
        warningText.text += "[" + lockOnMessage.Substring(0,
Mathf.Min(Mathf.FloorToInt(lockOnMessageTime * 20), lockOnMessage.Length)) + "]\n";
    }
    else
        lockOnMessageTime = 0;

    if (playerShip.kinematicBody.isGrounded)
    {
        liftoffMessageTime += Time.deltaTime;
        warningText.text += "[" + liftoffMessage.Substring(0,
Mathf.Min(Mathf.FloorToInt(liftoffMessageTime * 20), liftoffMessage.Length)) + "]\n";
    }
    else
        liftoffMessageTime = 0;
}
else
{
    lockOnMessageTime = 0;
    liftoffMessageTime = 0;
}

private void RobotBurn(float amount)
{
    if (fuelRemaining == 0)
        robotMetalCount -= amount / 2;
    else
        fuelRemaining = Mathf.Clamp(fuelRemaining - amount, 0, fuel.MaxValue);
// + playerRobot.forceMeter / 100; // [needs balancing first] like foxy
}
private void RobotBreath(float amount)
{
    if (oxygenRemaining == 0 && amount > 0)
        RobotBurn(amount);
    else
        oxygenRemaining = Mathf.Clamp(oxygenRemaining - amount, 0,
oxygen.MaxValue);
}
```

LetterEffect.cs

```

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

public class LetterEffect : MonoBehaviour
{
    Vector3 ogLocalPos;

    float distToStart; //Vector3.Dot(letter.right, ogLocalPos[i] -
transform.localPosition)
    float startTime = 0;

    Dialogue.Letter letter;

    bool active = false;
    bool fadein = true;
    bool fadeout = false;

    public void Init(Dialogue.Letter _letter, float _distToStart, float addDelay)
    {
        ogLocalPos = transform.localPosition;
        distToStart = _distToStart;

        letter = _letter;
        letter.delay += addDelay;

        startTime = Time.realtimeSinceStartup;

        GetComponent<MeshRenderer>().sharedMaterial = new
Material(GetComponent<MeshRenderer>().sharedMaterial);
        GetComponent<MeshRenderer>().enabled = false;
    }

    private void LateUpdate()
    {
        transform.localPosition = ogLocalPos;

        if (!active)
        {
            if (Time.realtimeSinceStartup < startTime + letter.delay)
                return;
            active = true;
            GetComponent<MeshRenderer>().enabled = true;
        }

        if (fadein || fadeout)
        {
            Color bgColour =
GetComponent<MeshRenderer>().sharedMaterial.GetColor("_bgColour");
            float now = Time.realtimeSinceStartup;
            //If fading
            if (now < startTime + letter.delay + letter.fadeVelTime.y)
            {
                float t = Mathf.InverseLerp(startTime + letter.delay, startTime +
letter.delay + letter.fadeVelTime.y, now);

                if (fadeout)
                    t = 1 - t;
            }
        }
    }
}

```

```

        //Any power to add "personality" to text fade effect
        t = Mathf.Pow(t, 0.3f);

        bgColour.a = t * 0.1f;

        //Move the letter up or down based on fade in or out
        transform.position -= letter.fadeVelTime.x * letter.fadeVelTime.y * (1
- t) * (fadeout ? -1 : 1) * transform.up; //dist = vel * time
        GetComponent<MeshRenderer>().sharedMaterial.SetColor("_Color",
letter.colour * new Color(1, 1, 1, t));
        GetComponent<MeshRenderer>().sharedMaterial.SetColor("_bgColour",
bgColour);
    }
    else
    {
        //If finished fading out
        if (fadeout)
            Destroy(gameObject); //good idea?

        bgColour.a = 0.1f;

        GetComponent<MeshRenderer>().sharedMaterial.SetColor("_Color",
letter.colour);
        GetComponent<MeshRenderer>().sharedMaterial.SetColor("_bgColour",
bgColour);

        fadein = false;
    }
}

//Move the letter up and down based on a global sine wave multiplied by the
global amplitude of the letter
transform.position += Mathf.Sin(distToStart + Time.realtimeSinceStartup) *
letter.freqSpeedAmpAmp.w * transform.up;

//Move the letter up and down based on the frequency, speed, and amplitude of
the letter
transform.position += Mathf.Sin(distToStart * letter.freqSpeedAmpAmp.x +
Time.realtimeSinceStartup * letter.freqSpeedAmpAmp.y) * letter.freqSpeedAmpAmp.z *
transform.up;
}

public void FadeOut()
{
    fadeout = true;
    startTime = Time.realtimeSinceStartup - letter.delay;
}
}

```

MeetingHandler.cs

```

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

public class MeetingHandler : MonoBehaviour
{
    [SerializeField]
    public struct PreviousMeetings
    {
        public int count;
        public int[] prevMeetSystemSeeds;

        public bool DoneAlready(int seed)
        {
            if (prevMeetSystemSeeds == null)
                WipeSave();

            foreach (int prevSeed in prevMeetSystemSeeds)
            {
                if (seed == prevSeed)
                    return true;
            }
            return false;
        }

        public void AddDone(int seed)
        {
            if (prevMeetSystemSeeds == null)
                WipeSave();

            foreach (int prevSeed in prevMeetSystemSeeds)
            {
                if (seed == prevSeed)
                    return;
            }
            prevMeetSystemSeeds = prevMeetSystemSeeds.Concat(new int[1] { seed }).ToArray();
            count++;
        }

        public void WipeSave()
        {
            prevMeetSystemSeeds = new int[0];
            count = 0;
        }
    }
    public static void Wipe()
    {
        previousMeetings.WipeSave();
    }

    public static void Save()
    {
        JsonSaver.SaveData("Previous_Meetings", previousMeetings);
    }

    private static PreviousMeetings previousMeetings = new PreviousMeetings();
}

```

MeshGen.cs

```

using UnityEditor;
using UnityEngine;

public class MeshGen
{
    private Planet planet;

    private Mesh[,] faceSegs;
    private int numSegs;
    private Vector2Int[,] segCoords;

    public ElevationData elevationData;

    public MeshGen(Planet _planet)
    {
        elevationData = new ElevationData();
        planet = _planet;
        numSegs = 6 * planet.splitFaces * planet.splitFaces;
    }
    private void CalculateQuadSphere()
    {
        int resolution = planet.resolution;
        int splitFaces = planet.splitFaces;

        if (resolution % splitFaces != 0)
        {
            Debug.LogError("Error: Bad Split Faces number");
            return;
        }

        int gridsize = resolution / splitFaces;
        faceSegs = new Mesh[6, numSegs / 6];
        segCoords = new Vector2Int[6, numSegs / 6];

        //For each face on the quadsphere
        for (int i = 0; i < 2; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                //For each segment on the quasphere face
                for (int x = 0; x < resolution; x += gridsize)
                {
                    for (int y = 0; y < resolution; y += gridsize)
                    {
                        Vector2Int coord = new Vector2Int(x, y);

                        CalcSegMesh(coord, resolution, gridsize, i, j);

                        segCoords[i * 3 + j, (x / gridsize * resolution + y) / gridsize]
= coord;
                    }
                }
            }
        }
    }
    public void RecalcSegMesh(int segIndex, int resolution, int prevRes, int splitFaces)
    {
        if (resolution % splitFaces != 0)
    }
}

```

```

    {
        Debug.LogError("Error: Bad Split Faces number");
        return;
    }

    int gridsize = resolution / splitFaces;
    int prevGrid = prevRes / splitFaces;

    int faceNo = segIndex * 6 / faceSegs.Length;
    int i = faceNo / 3;
    int j = faceNo % 3;

    Vector2Int coord = segCoords[faceNo, segIndex % (faceSegs.Length / 6)];

    CalcSegMesh(coord, prevRes, prevGrid, resolution, gridsize, i, j);
}
private void CalcSegMesh(Vector2Int coord, int resolution, int gridsize, int i,
int j)
{
    CalcSegMesh(coord, resolution, gridsize, resolution, gridsize, i, j);
}
private void CalcSegMesh(Vector2Int coord, int resIndex, int gridIndex, int
resolution, int gridsize, int i, int j)
{
    Vector3[] start = new Vector3[] { -Vector3.one, Vector3.one };
    Vector3[] ends = new Vector3[] { Vector3.right, Vector3.up, Vector3.forward };

    Vector3 end = 2 * (ends[j] + (i == 0 ? ends[(j + 1) % 3] : Vector3.zero)) -
Vector3.one;

    (Vector3[] verts, int[] tris, Vector2[] uvs) = DivideSeg(start[i], end, !(j ==
0 || (j == 1 && i == 1)), resolution, gridsize, coord);

    faceSegs[i * 3 + j, (coord.x / gridIndex * resIndex + coord.y) / gridIndex] =
NewMesh(verts, tris, uvs);
}

private (Vector3[], int[], Vector2[]) DivideSeg(Vector3 start, Vector3 end, bool
flip, int resolution, int gridsize, Vector2Int coord)
{
    //Determine the axis of the line segment
    int axisConst = start.x == end.x ? 0 : start.y == end.y ? 1 : 2;

    Vector3[] verts = new Vector3[(gridsize + 1) * (gridsize + 1)];
    Vector2[] uvs = new Vector2[verts.Length];

    //Iterate through all of the points in the grid to calculate the vertices and
    UV coordinates
    for (int x = coord.x; x < gridsize + coord.x + 1; x++)
    {
        float xPos = start.x + (end.x - start.x) * x / resolution;
        float zPos = start.z;
        if (axisConst == 0)
        {
            xPos = start.x;
            zPos = start.z + (end.z - start.z) * x / resolution;
        }
        for (int y = coord.y; y < gridsize + coord.y + 1; y++)
        {
            float yPos = start.y + (end.y - start.y) * y / resolution;
            if (axisConst == 1)
            {

```

```

        yPos = start.y;
        zPos = start.z + (end.z - start.z) * y / resolution;
    }
    //Concentrate verticies at the centre of the face so that they do not
    cluster at the corners when projected onto a sphere
    float xPosI = xPos * Mathf.Sqrt(1 - (yPos * yPos + zPos * zPos)) / 2 +
    (yPos * yPos * zPos * zPos) / 3);
    float yPosI = yPos * Mathf.Sqrt(1 - (zPos * zPos + xPos * xPos)) / 2 +
    (zPos * zPos * xPos * xPos) / 3);
    float zPosI = zPos * Mathf.Sqrt(1 - (xPos * xPos + yPos * yPos)) / 2 +
    (xPos * xPos * yPos * yPos) / 3);
    verts[(x - coord.x) * (gridsize + 1) + (y - coord.y)] = new
    Vector3(xPosI, yPosI, zPosI).normalized;
    uvs[(x - coord.x) * (gridsize + 1) + (y - coord.y)] = new
    Vector2(axisConst == 0 ? zPos : xPos, (axisConst == 1 ? -zPos : yPos));
}
int[] tris = new int[gridsize * gridsize * 6];
//Iterate through each square in the grid and add its triangles to the
triangle array
for (int x = 0; x < gridsize; x++)
{
    for (int y = 0; y < gridsize; y++)
    {
        int n = x * (gridsize + 1) + y;
        int[] order = new int[6] { n, n + 1, n + gridsize + 2, n, n + gridsize
+ 2, n + gridsize + 1 };

        //Flip the triangle order if necessary
        for (int i = 0; i < 6; i++)
        {
            int index = flip ? 5 - i : i;
            tris[(x * gridsize + y) * 6 + i] = order[index];
        }
    }
}
return (verts, tris, uvs);
}

public void ApplyQuadSphere()
{
    //Already generated quad sphere and saved as prefabrication to load below
    /*
    string foldername = planet.splitFaces + "_" + planet.resolution + "_" +
planet.gameObject.layer;
    string editor_foldername = "Assets/Planet
Generation/Scripts/Creator/Resources/" + foldername;

    if (!AssetDatabase.IsValidFolder(editor_foldername))
    {
        GameObject _segs = ReplaceChild(planet.transform.GetChild(0).gameObject,
new GameObject());
        Debug.Log("Creating new folder");

        AssetDatabase.CreateFolder("Assets/Planet
Generation/Scripts/Creator/Resources", planet.splitFaces + "_" + planet.resolution);

        CalculateQuadSphere();
        int i = 0;
        foreach (Mesh _seg in faceSegs)
    }
}
```

```

    {
        GameObject _child = new GameObject(i.ToString());

        _child.AddComponent<MeshRenderer>();
        _child.AddComponent<MeshCollider>();
        MeshFilter meshFilter = _child.AddComponent<MeshFilter>();
        meshFilter.sharedMesh = _seg;

        _child.transform.position = planet.transform.position;
        _child.transform.parent = _segs.transform;

        Segment _childSeg = _child.AddComponent<Segment>();
        _childSeg.SegIndex = i;

        AssetDatabase.CreateAsset(_seg, editor_foldername + "/" + i +
        ".mesh");
        AssetDatabase.SaveAssets();

        i++;
    }
    PrefabUtility.SaveAsPrefabAsset(_segs, editor_foldername +
    "/Segment.prefab");
}
*/

```

GameObject segs = ReplaceChild(planet.transform.GetChild(0).gameObject,
Object.Instantiate(planet.planetMeshSegmentPrefabs[planet.gameObject.layer - 7]));

```

for (int i = 0; i < numSegs; i++)
{
    GameObject child = segs.transform.GetChild(i).gameObject;
    child.layer = segs.layer;

    Mesh prefabMesh = child.GetComponent<MeshFilter>().sharedMesh;
    Mesh childMesh = prefabMesh;
    //childMesh.vertices = prefabMesh.vertices;
    //childMesh.triangles = prefabMesh.triangles;
    //childMesh.uv = prefabMesh.uv;

    Segment childSeg = child.GetComponent<Segment>();

    childSeg.planet = planet;
    childSeg.SetMesh(childMesh);
    childSeg.SetRenderMaterial(planet.terrainMat);
}
}

public GameObject ReplaceChild(GameObject child, GameObject new_child)
{
    new_child.name = child.name;
    new_child.transform.parent = child.transform.parent;
    new_child.layer = child.layer;
    new_child.transform.position = child.transform.position;
    new_child.transform.SetSiblingIndex(child.transform.GetSiblingIndex());
    if (Application.isEditor)
        Object.DestroyImmediate(child);
    else
        Object.Destroy(child);
    return new_child;
}

static Mesh NewMesh(Vector3[] verts, int[] tris, Vector2[] uvs)

```

```

    {
        Mesh mesh = new Mesh();
        mesh.vertices = verts;
        mesh.triangles = tris;
        mesh.uv = uvs;
        return mesh;
    }

    public void UpdateNoiseGPU()
    {
        elevationData.Reset(planet.planetValues.radius);

        Vector3[] wOffsets = new Vector3[planet.warpNoise.octaves];
        Vector3[] cOffsets = new Vector3[planet.continentNoise.octaves];
        Vector3[] mOffsets = new Vector3[planet.mountainNoise.octaves +
planet.mountainNoise.initialExtraOctaves];
        Vector3[] rOffsets = new Vector3[planet.roughNoise.octaves];

        System.Random wPrng = new System.Random(planet.warpNoise.seed);
        System.Random cPrng = new System.Random(planet.continentNoise.seed);
        System.Random mPrng = new System.Random(planet.mountainNoise.seed);
        System.Random rPrng = new System.Random(planet.roughNoise.seed);

        for (int i = 0; i < wOffsets.Length; i++)
        {
            float x = wPrng.Next(-1000, 1000);
            float y = wPrng.Next(-1000, 1000);
            float z = wPrng.Next(-1000, 1000);
            wOffsets[i] = new Vector3(x, y, z);
        }
        for (int i = 0; i < cOffsets.Length; i++)
        {
            float x = cPrng.Next(-1000, 10000);
            float y = cPrng.Next(-1000, 1000);
            float z = cPrng.Next(-1000, 1000);
            cOffsets[i] = new Vector3(x, y, z);
        }
        for (int i = 0; i < mOffsets.Length; i++)
        {
            float x = mPrng.Next(-1000, 1000);
            float y = mPrng.Next(-1000, 1000);
            float z = mPrng.Next(-1000, 1000);
            mOffsets[i] = new Vector3(x, y, z);
        }
        for (int i = 0; i < rOffsets.Length; i++)
        {
            float x = rPrng.Next(-1000, 1000);
            float y = rPrng.Next(-1000, 1000);
            float z = rPrng.Next(-1000, 1000);
            rOffsets[i] = new Vector3(x, y, z);
        }
    }

    ComputeShader computeShader = planet.noiseShader;

    int handle = computeShader.FindKernel("GeneratePlanetNoise");

    computeShader.SetInt("numCraters", planet.craterValues.numCraters);
    computeShader.SetFloat("ctrSmoothness", planet.craterValues.smoothness);
    computeShader.SetFloat("ctrRimWidth", planet.craterValues.rimWidth);
    computeShader.SetFloat("ctrRimSteepness", planet.craterValues.rimSteepness);

    computeShader.SetInt("wOctaves", planet.warpNoise.octaves);
}

```

```

computeShader.SetFloat("wScale", planet.warpNoise.scale);
computeShader.SetFloat("wPersistance", planet.warpNoise.persistance);
computeShader.SetFloat("wLacunarity", planet.warpNoise.lacunarity);

computeShader.SetInt("cOctaves", planet.continentNoise.octaves);
computeShader.SetFloat("cScale", planet.continentNoise.scale);
computeShader.SetFloat("cPersistance", planet.continentNoise.persistance);
computeShader.SetFloat("cLacunarity", planet.continentNoise.lacunarity);
computeShader.SetFloat("cDropoff", planet.continentNoise.dropoff);

computeShader.SetInt("mOctaves", planet.mountainNoise.octaves);
computeShader.SetFloat("mScale", planet.mountainNoise.scale);
computeShader.SetFloat("mPersistance", planet.mountainNoise.persistance);
computeShader.SetFloat("mLacunarity", planet.mountainNoise.lacunarity);
computeShader.SetFloat("mDropoff", planet.mountainNoise.dropoff);
computeShader.SetInt("mInitialExtraOctaves",
planet.mountainNoise.initialExtraOctaves);

computeShader.SetInt("rOctaves", planet.roughNoise.octaves);
computeShader.SetFloat("rPersistance", planet.roughNoise.persistance);
computeShader.SetFloat("rLacunarity", planet.roughNoise.lacunarity);
computeShader.SetFloat("rStartingOctave", planet.roughNoise.startingOctave);

computeShader.SetFloat("groundLevel", planet.planetValues.groundLevel);
computeShader.SetFloat("seabedLevel", planet.planetValues.seabedLevel);
computeShader.SetFloat("radius", planet.planetValues.radius);
computeShader.SetBool("roughBed", planet.planetValues.roughBed);
computeShader.SetBool("crackedGround", planet.planetValues.crackedGround);

ComputeBuffer craterBuffer = new ComputeBuffer(planet.craters.Length,
sizeof(float) * 5);
craterBuffer.SetData(planet.craters);
computeShader.SetBuffer(handle, "craters", craterBuffer);

ComputeBuffer warpBuffer = new ComputeBuffer(wOffsets.Length, sizeof(float) *
3);
ComputeBuffer continentBuffer = new ComputeBuffer(cOffsets.Length,
sizeof(float) * 3);
ComputeBuffer mountainBuffer = new ComputeBuffer(mOffsets.Length,
sizeof(float) * 3);
ComputeBuffer roughBuffer = new ComputeBuffer(rOffsets.Length, sizeof(float) *
3);
warpBuffer.SetData(wOffsets);
continentBuffer.SetData(cOffsets);
mountainBuffer.SetData(mOffsets);
roughBuffer.SetData(rOffsets);
computeShader.SetBuffer(handle, "wOffsets", warpBuffer);
computeShader.SetBuffer(handle, "cOffsets", continentBuffer);
computeShader.SetBuffer(handle, "mOffsets", mountainBuffer);
computeShader.SetBuffer(handle, "rOffsets", roughBuffer);

for (int i = 0; i < numSegs; i++)
{
    Mesh seg =
planet.transform.GetChild(0).GetChild(i).GetComponent<Segment>().mesh;
    if (seg == null)
        continue;

    Vector3[] verts = seg.vertices;
    computeShader.SetInt("vertexCount", seg.vertexCount);
}

```

```
        ComputeBuffer vertBuffer = new ComputeBuffer(seg.vertexCount,
sizeof(float) * 3);
        vertBuffer.SetData(verts);
        computeShader.SetBuffer(handle, "rwVerts", vertBuffer);

        ComputeBuffer elevationBuffer = new ComputeBuffer(seg.vertexCount,
sizeof(float));
        computeShader.SetBuffer(handle, "rwWorldHeight", elevationBuffer);

        computeShader.Dispatch(handle, Mathf.CeilToInt(seg.vertexCount / 256f), 1,
1);

        vertBuffer.GetData(verts);

        float[] elevations = new float[seg.vertexCount];
        elevationBuffer.GetData(elevations);

        int j = 0;
        foreach (float elevation in elevations)
        {
            elevationData.Add(elevation);
            j++;
        }
        elevationData.AddMap(verts);

        elevationBuffer.Dispose();
        vertBuffer.Dispose();
        seg.vertices = verts;

planet.transform.GetChild(0).GetChild(i).GetComponent<Segment>().SetMesh(seg);

planet.transform.GetChild(0).GetChild(i).GetComponent<Segment>().UpdateMeshFilter();
    }
    elevationData.CalcStats();

    craterBuffer.Dispose();
    warpBuffer.Dispose();
    continentBuffer.Dispose();
    mountainBuffer.Dispose();
    roughBuffer.Dispose();
}
}
```

Minimap.cs

```

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

public class MiniMap
{
    private Planet planet;
    private ElevationData elevationData;
    private Vector3[,,] faceHeightMaps;
    private Texture2D globeMap;
    public GameObject Map { get; private set; }

    public MiniMap(Planet _planet, GameObject _Map)
    {
        planet = _planet;
        ResetElevationData();

        Map = _Map;
        Mesh plane = Map.GetComponent<MeshFilter>().sharedMesh;
        Mesh mapMesh = new Mesh();
        mapMesh.vertices = plane.vertices;
        mapMesh.uv = plane.uv;
        mapMesh.triangles = plane.triangles;
        mapMesh.normals = plane.normals;
        Map.GetComponent<MeshFilter>().sharedMesh = mapMesh;
        Map.GetComponent<MeshRenderer>().sharedMaterial = new
Material(Map.GetComponent<MeshRenderer>().sharedMaterial);
    }
    void ResetElevationData()
    {
        elevationData = new ElevationData();
        elevationData = planet.planetMesh.elevationData;
    }

    public void CreatePlanetMap()
    {
        //return; //Make work for GPU first
        CreateHeightMapCubeMesh();
        CreateGlobeMap();
    }
    void CreateHeightMapCubeMesh()
    {
        int segsPerFaceRow = planet.splitFaces;
        int segsPerFace = segsPerFaceRow * segsPerFaceRow;

        int vertsPerSegRow = planet.resolution / planet.splitFaces + 1;
        int vertsPerSeg = vertsPerSegRow * vertsPerSegRow;

        int vertsPerFaceRow = vertsPerSegRow * segsPerFaceRow;
        int vertsPerFace = vertsPerFaceRow * vertsPerFaceRow;

        faceHeightMaps = new Vector3[6, vertsPerFaceRow, vertsPerFaceRow];

        for (int faceNo = 0; faceNo < 6; faceNo++)
        {
            bool leftStart = faceNo == 0 || faceNo == 2 || faceNo == 5;
            bool bottomStart = faceNo == 0 || faceNo == 1;

            //Texture2D sets pixels bottom row (left to right) to top row. Therefore
            we need to convert the vertex arrangement of the face to this order
            //For face 0 this order is: left column (bottom to top) to right column
            //For face 1: right column (bottom to top)...
        }
    }
}

```

```

int firstSeg = faceNo * segsPerFace;
int bottomLeftSeg = 0;
int bottomLeftVert = 0;

if (leftStart && bottomStart)
{
    bottomLeftSeg = firstSeg;
    bottomLeftVert = 0;
}
else if (leftStart && !bottomStart)
{
    bottomLeftSeg = firstSeg + segsPerFaceRow - 1;
    bottomLeftVert = vertsPerSegRow - 1;
}
else if (!leftStart && bottomStart)
{
    bottomLeftSeg = firstSeg + segsPerFace - segsPerFaceRow;
    bottomLeftVert = vertsPerSeg - vertsPerSegRow;
}
else if (!leftStart && !bottomStart)
{
    bottomLeftSeg = firstSeg + segsPerFace - 1;
    bottomLeftVert = vertsPerSeg - 1;
}

for (int x = 0; x < vertsPerFaceRow; x++)
{
    for (int y = 0; y < vertsPerFaceRow; y++)
    {
        //Determine the segment index for the current vertex
        int seg = bottomLeftSeg;
        seg += x / vertsPerSegRow * segsPerFaceRow * (leftStart ? 1 : -1);
        seg += y / vertsPerSegRow * (bottomStart ? 1 : -1);
        Vector3[] verts = elevationData.GetMapTexture(seg);

        //Determine the local x and y indices of the current vertex within
        //its segment
        int localX = x - x / vertsPerSegRow * vertsPerSegRow;
        int localY = y - y / vertsPerSegRow * vertsPerSegRow;

        int vert = bottomLeftVert;
        vert += localX * vertsPerSegRow * (leftStart ? 1 : -1);
        vert += localY * (bottomStart ? 1 : -1);

        //Color colour = Color.Lerp(Color.black, Color.white,
        Mathf.InverseLerp(elevationData.Min, elevationData.Max, verts[vert].magnitude));

        faceHeightMaps[faceNo, x, y] = verts[vert];
    }
}
}

void CreateGlobeMap()
{
    List<Vector3> towerNormals = new List<Vector3>();
    foreach (TowerGen tower in planet.GetComponentInChildren<TowerGen>())
        towerNormals.Add((tower.transform.position -
planet.transform.position).normalized);
    Vector3 playerNormal = (Map.transform.position -
planet.transform.position).normalized;
}

```

```

int segsPerFaceRow = planet.splitFaces;

int vertsPerSegRow = planet.resolution / planet.splitFaces + 1;

int vertsPerFaceRow = vertsPerSegRow * segsPerFaceRow;

int normalLength =
(int)Mathf.Sqrt(Map.GetComponent<MeshFilter>().sharedMesh.vertexCount);
int w = 800;
int h = 400;

globeMap = new Texture2D(w, h);

Vector3[] vertNormal = new
Vector3[Map.GetComponent<MeshFilter>().sharedMesh.vertexCount];

//http://paulbourke.net/panorama/cubemaps/ (Converting to a spherical
projection from 6 cubic environment maps, roughly a quarter down the page)
for (int x = 0; x < w; x++)
{
    float xNP = (float)x / w * 2 - 1;
    for (int y = 0; y < h; y++)
    {
        float yNP = (float)y / h * 2 - 1;

        //Calculate the position of the vertex at the given UV coordinates
        Vector3 pos = VertAtPolarCoord(xNP, yNP, vertsPerFaceRow);

        //Calculate the distance of the vertex from each tower and the player,
        and determine the minimum distance
        float distance = float.MaxValue;
        foreach (Vector3 normal in towerNormals)
            distance = Mathf.Min(distance, Mathf.Max(Mathf.InverseLerp(0,
0.04f, (pos.normalized - normal).sqrMagnitude), 0.1f));
        distance = Mathf.Min(distance, Mathf.InverseLerp(0, 0.03f,
(pos.normalized - playerNormal).sqrMagnitude));

        //The alpha channel stores the highlight splodges for each tower and
player on the map
        globeMap.SetPixel(x, y, Color.Lerp(new Color(0,0,0, distance), new
Color(1,1,1, distance), Mathf.InverseLerp(elevationData.Min, elevationData.Max,
pos.magnitude)));
    }
}
//Store the normalized position of each vertex at the given UV coordinates
//(allows biomes to be coloured properly on map)
for (int x = 0; x < normalLength; x++)
{
    float xNP = (float)x / normalLength * 2 - 1;
    for (int y = 0; y < normalLength; y++)
    {
        float yNP = (float)y / normalLength * 2 - 1;

        Vector3 pos = VertAtPolarCoord(xNP, yNP, vertsPerFaceRow);

        vertNormal[y * normalLength + x] = pos;
    }
}
globeMap.Apply();

```

```

        Map.GetComponent<MeshFilter>().sharedMesh.normals = vertNormal;

        Map.GetComponent<MeshRenderer>().sharedMaterial.SetTexture("_map", globeMap);

        UpdatePointers();
    }
    public void UpdatePointers()
    {
        if (planet.planetValues.roughBed)

Map.GetComponent<MeshRenderer>().sharedMaterial.EnableKeyword("_ROUGHBED");
        else

Map.GetComponent<MeshRenderer>().sharedMaterial.DisableKeyword("_ROUGHBED");
        Map.GetComponent<MeshRenderer>().sharedMaterial.SetColor("_seaColour",
planet.oceanMat.GetColor("_seaColour"));
        Map.GetComponent<MeshRenderer>().sharedMaterial.SetTexture("_planetTexture",
planet.terrainMat.GetTexture("_planetTexture"));
    }

    int ClosestSeg()
    {
        (int index, float minDist) = (-1, float.MaxValue);
        for (int i = 0; i < planet.transform.GetChild(0).childCount; i++)
        {
            float thisCamDist =
planet.transform.GetChild(0).GetChild(i).GetComponent<Segment>().camDist;
            if (thisCamDist < minDist)
                (index, minDist) = (i, thisCamDist);
        }
        return index;
    }

    Vector3 VertAtPolarCoord(float x, float y, int vertsPerFaceRow)
    {
        float s = x * Mathf.PI;
        float t = y * Mathf.PI / 2;

        //I created a Geogebra file to help me visualise converting a direction to a
        point on a 'skybox' https://www.geogebra.org/m/kwkbpd5k
        //The variable names correspond to the variable names in the file
        float a = 1 / (Mathf.Cos(t) * Mathf.Cos(s));
        float b = 1 / Mathf.Sin(t);
        float c = 1 / (Mathf.Cos(t) * Mathf.Sin(s));
        float m = Mathf.Min(Mathf.Abs(a), Mathf.Abs(b), Mathf.Abs(c));
        Vector3 P = new Vector3(1 / a, 1 / b, 1 / c);
        Vector3 M = m * P;
        Vector3 C = M * Vector3.Dot(M, P) / Mathf.Abs(Vector3.Dot(M, P));
        C = (C + Vector3.one) / 2;

        int faceNo = m == Mathf.Abs(a) ? (a > 0 ? 0 : 5) : m == Mathf.Abs(b) ? (b > 0
? 4 : 2) : (c > 0 ? 3 : 1);
        Vector2 samplePoint = Vector2.zero;
        if (faceNo == 0)
            samplePoint = new Vector2(C.z, C.y);
        else if (faceNo == 1)
            samplePoint = new Vector2(C.x, C.y);
        else if (faceNo == 2)
            samplePoint = new Vector2(C.z, C.x);
        else if (faceNo == 3)
            samplePoint = new Vector2(1 - C.x, C.y);
        else if (faceNo == 4)
    }
}

```

```
        samplePoint = new Vector2(C.z, 1 - C.x);
    else if (faceNo == 5)
        samplePoint = new Vector2(1 - C.z, C.y);

    float sampleX = samplePoint.x * (vertsPerFaceRow - 1);
    float sampleY = samplePoint.y * (vertsPerFaceRow - 1);

    Vector3 dd = faceHeightMaps[faceNo, Mathf.FloorToInt(sampleX),
Mathf.FloorToInt(sampleY)];
    Vector3 du = faceHeightMaps[faceNo, Mathf.FloorToInt(sampleX),
Mathf.CeilToInt(sampleY)];
    Vector3 ud = faceHeightMaps[faceNo, Mathf.CeilToInt(sampleX),
Mathf.FloorToInt(sampleY)];
    Vector3 uu = faceHeightMaps[faceNo, Mathf.CeilToInt(sampleX),
Mathf.CeilToInt(sampleY)];

    return Vector3.Lerp(Vector3.Lerp(dd, du, sampleY - Mathf.Floor(sampleY)),
Vector3.Lerp(ud, uu, sampleY - Mathf.Floor(sampleY)), sampleX - Mathf.Floor(sampleX));
}
}
```

Noise.cginc

```

#include "PerlinNoise.cginc"

float3 GenerateWarpMap float3 vert float persistance float lacunarity float scale
int octaves StructuredBuffer<float3> offsets

float height = 0.0f
float amp = 1.0f
float freq = 1.0f
for int i = 0 i < octaves i++

    float3 samplePoint = vert * freq + offsets i

    float perlinValue = PerlinNoise3D samplePoint * scale

    height += perlinValue * amp

    //Higher octaves have higher frequency and smaller amplitude (finer noise)
    amp *= persistance
    freq *= lacunarity

return height * vert


float GenerateContinents float3 vert float persistance float lacunarity float
scale int octaves StructuredBuffer<float3> offsets float dropoff float
groundLevel float seabedLevel float3 warpMap

float height = 0.0f
float amp = 1.0f
float freq = 1.0f
for int i = 0 i < octaves i++

    //Offset by warp map for more interesting (warped) noise
    float3 samplePoint = vert * freq + offsets i + warpMap
    float perlinValue = 1 + PerlinNoise3D samplePoint * scale
    height += perlinValue * amp

    amp *= persistance
    freq *= lacunarity

if height > groundLevel
    height = lerp groundLevel height dropoff

return height


float GenerateMountains float3 vert float persistance float lacunarity float scale
int octaves StructuredBuffer<float3> offsets float dropoff float
initialExtraOctaves float groundLevel float seabedLevel

float height = 0.0f
float amp = 1.0f
float freq = 1.0f

for int i = 0 i < octaves i++

    float3 samplePoint = vert * freq + offsets i
    float perlinValue = 1 + PerlinNoise3D samplePoint * scale

    //Run first octave multiple times to generate lots of mountains
    if i == 0

```

```

        for int j = octaves j < initialExtraOctaves j++
            samplePoint = vert * freq + offsets j
            perlinValue = max perlinValue 1 + PerlinNoise3D samplePoint *
scale

height += perlinValue * amp

amp *= persistance
freq *= lacunarity

if height > groundLevel
    height = height * height * dropoff

return height

float GenerateRoughTerrain float3 vert float persistance float lacunarity int
octaves StructuredBuffer<float3> offsets float startingOctave

float height = 0.0f
float amp = pow abs persistance startingOctave
float freq = pow abs lacunarity startingOctave

for int i = 0 i < octaves i++
    float3 samplePoint = vert * freq + offsets i
    float perlinValue = 1 + PerlinNoise3D samplePoint
    height += perlinValue * amp

    amp *= persistance
    freq *= lacunarity

height /= 10
return 1 + height

struct Craters

    float3 position
    float radius
    float floorHeight

///START OF NOT MY CODE

float GenerateCraters float3 vert float rimWidth float rimSteepness float
smoothness int numCraters StructuredBuffer<Craters> craters float radius

float craterHeight = 0
for int i = 0 i < numCraters i++

    float x = length vert - craters i .position / craters i .radius

    float cavity = x * x - 1
    float rimX = min 0 x - 1 - rimWidth
    float rim = rimSteepness * rimX * rimX

    float craterShape = Smooth cavity craters i .floorHeight -smoothness

```

```
craterShape = Smooth craterShape rim smoothness
craterHeight += craterShape * craters i .radius

return max -radius / 2 craterHeight

///END OF NOT MY CODE
```

NoiseShader.compute

```

#pragma kernel GeneratePlanetNoise
#include "Noise.cginc"

const int numCraters
const float ctrSmoothness
const float ctrRimWidth
const float ctrRimSteepness

const int wOctaves
const float wScale
const float wPersistence
const float wLacunarity

const int cOctaves
const float cScale
const float cPersistence
const float cLacunarity
const float cDropoff

const int mOctaves
const float mScale
const float mPersistence
const float mLacunarity
const float mDropoff
const int mInitialExtraOctaves

const int rOctaves
const float rPersistence
const float rLacunarity
const float rStartingOctave

const float groundLevel
const float seabedLevel
const float radius
const bool roughBed
const bool crackedGround

const int vertexCount

StructuredBuffer<Craters> craters

StructuredBuffer<float3> cOffsets
StructuredBuffer<float3> mOffsets
StructuredBuffer<float3> wOffsets
StructuredBuffer<float3> rOffsets

RWStructuredBuffer<float> rwWorldHeight
RWStructuredBuffer<float3> rwVerts

numthreads 256 1 1
void GeneratePlanetNoise uint3 id : SV_DispatchThreadID

if int id.x > vertexCount
    return

if length rwVerts id.x != 1.0f
    rwVerts id.x = normalize rwVerts id.x

```

```
rwVerts id.x *= radius / 500

const float3 warpMap = GenerateWarpMap rwVerts id.x wPersistence wLacunarity
wScale wOctaves wOffsets

const float cHeight = GenerateContinents rwVerts id.x cPersistence cLacunarity
cScale cOctaves cOffsets cDropoff groundLevel seabedLevel warpMap

const float mHeight = GenerateMountains rwVerts id.x mPersistence mLacunarity
mScale mOctaves mOffsets mDropoff mInitialExtraOctaves groundLevel seabedLevel

const float ctrHeight = GenerateCraters rwVerts id.x ctrRimWidth
ctrRimSteepness ctrSmoothness numCraters craters radius

float minLevel
if roughBed
    minLevel = GenerateRoughTerrain rwVerts id.x rPersistence rLacunarity
rOctaves rOffsets rStartingOctave
else
    minLevel = seabedLevel

float worldHeight = max max cHeight mHeight minLevel

if crackedGround && abs PerlinNoise3D rwVerts id.x * 3 < 0.03
    worldHeight = lerp seabedLevel * minLevel worldHeight
pow abs PerlinNoise3D rwVerts id.x * 3 / 0.03 0.5

worldHeight += ctrHeight
worldHeight *= radius

rwWorldHeight id.x = worldHeight
rwVerts id.x = normalize rwVerts id.x * worldHeight
```

NoiseStructs.cs

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

public struct MeshData
{
    public NodePack[] nodes;
    public int[] tris;
}
public struct NodePack
{
    public Vector3 vert;
    public Vector2 uv;
}

public struct Craters
{
    public Vector3 point;
    public float radius;
    public float floorHeight;
}

public struct PlanetValues
{
    public int terrainSeed;
    public int colourSeed;
    public int environmentSeed;
    public Dictionary<float, Gradient> biomeGradients;
    public float groundLevel;
    public float seabedLevel;
    public bool roughBed;
    public bool crackedGround;
    public float radius;
    public float temperature;
    public float windSpeed;
}
public struct CraterValues
{
    public int seed;
    public int numCraters;
    public float smoothness;
    public float rimWidth;
    public float rimSteepness;
}
public struct ContinentNoiseValues
{
    public int seed;
    public int octaves;
    public float scale;
    public float persistance;
    public float lacunarity;
    public float dropoff;
}
public struct MountainNoiseValues
{
    public int seed;
    public int octaves;
    public float scale;
```

```
public float persistance;
public float lacunarity;
public float dropoff;
public int initialExtraOctaves;
}
public struct RoughNoiseValues
{
    public int seed;
    public int octaves;
    public float scale;
    public float persistance;
    public float lacunarity;
    public float startingOctave;
}
public struct WarpNoiseValues
{
    public int seed;
    public int octaves;
    public float scale;
    public float persistance;
    public float lacunarity;
}
```

OpenDoor.cs

```

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

public class OpenDoor : MonoBehaviour
{
    [SerializeField] private AudioClip doorOpening;
    [SerializeField] private AudioClip doorClosing;
    [SerializeField] private AudioClip doorLocked;

    [SerializeField] private float maxDist = 2;
    [SerializeField] private float maxAngle = 60;
    [SerializeField] private float openSpeed = 1;
    private float dynamicOpenVel = 0;
    [SerializeField] private float handleSpeed = 2;
    [SerializeField] private float handleAngle = -70;
    [SerializeField] private Transform Door;
    [SerializeField] private Transform handle;
    private enum DoorState { open, rotating, closed }
    [SerializeField] private DoorState doorState = DoorState.closed;
    private float rotLerp = 0;
    private float handleCycle = 2;
    private bool opening;
    public bool locked = false;
    public bool stuck = false;
    private IEnumerator openIfAndroidClose;

    private void Start()
    {
        if (locked)
            return;
        openIfAndroidClose = OpenIfAndroidClose(1);
        StartCoroutine(openIfAndroidClose);
    }
    private IEnumerator OpenIfAndroidClose(float time)
    {
        while (true)
        {
            yield return new WaitForSeconds(time);

            if (doorState != DoorState.closed)
                continue;

            foreach (Android android in FindObjectsOfType<Android>(false))
            {
                if ((android.transform.position - transform.position).sqrMagnitude > 25)
                    continue;

                //Start opening process
                opening = true;
                rotLerp = 0;
                doorState = DoorState.rotating;
                dynamicOpenVel = 0;
                GetComponent< AudioSource >().PlayOneShot(doorOpening);
                Door.GetChild(0).GetComponent< BoxCollider >().enabled = false;
                handleCycle = 0;
                break;
            }
        }
    }
}

```

```

        }

    }

// Update is called once per frame
void Update()
{
    if (doorState == DoorState.rotating && handleCycle > 1)
    {
        Door.localRotation = Quaternion.Slerp(Quaternion.Euler(0, 0, 0),
Quaternion.Euler(0, 90, 0), rotLerp);

        //If door is not stuck, adjust the rotation lerp value based on the
direction of movement
        if (!stuck)
        {
            rotLerp += (opening ? 1 : -1) * openSpeed * dynamicOpenVel *
Time.smoothDeltaTime;
            dynamicOpenVel = Mathf.Min(1, dynamicOpenVel + Time.smoothDeltaTime *
10);
        }
        else
            dynamicOpenVel = 0;

        if (rotLerp < 0)
        {
            doorState = DoorState.closed;
            Door.localRotation = Quaternion.Euler(0, 0, 0);
        }
        else if (rotLerp > 1)
        {
            doorState = DoorState.open;
            Door.localRotation = Quaternion.Euler(0, 90, 0);
            Door.GetComponent<BoxCollider>().enabled = true;
//disabled due to android
        }
    }

    else if (CameraState.CamIsInteractingW(handle.position,
Camera.main.transform.forward, maxDist, maxAngle))
    {
        //If the door is not locked, start the door opening or closing process
based on its current state
        if (!locked)
        {
            opening = doorState == DoorState.closed;
            rotLerp = opening ? 0 : 1;
            doorState = DoorState.rotating;
            dynamicOpenVel = 0;

            if (opening)
                GetComponent< AudioSource >().PlayOneShot(doorOpening);
            else
                GetComponent< AudioSource >().PlayOneShot(doorClosing);
        }
        else
            GetComponent< AudioSource >().PlayOneShot(doorLocked);
        handleCycle = 0;
    }

    handleCycle += handleSpeed * Time.smoothDeltaTime;
}

```

```
//Different animation for using the door handle dependent on whether the door
is locked
    if (handleCycle < 1)
        handle.localRotation = Quaternion.Slerp(Quaternion.Euler(0,
handle.localEulerAngles.y, 0), Quaternion.Euler(0, handle.localEulerAngles.y, locked ?
-5 : handleAngle), handleCycle);
    else if (handleCycle < 2)
        handle.localRotation = Quaternion.Slerp(Quaternion.Euler(0,
handle.localEulerAngles.y, locked ? -5 : handleAngle), Quaternion.Euler(0,
handle.localEulerAngles.y, 0), handleCycle - 1);
    else
        handle.localRotation = Quaternion.Euler(0, handle.localEulerAngles.y, 0);
}

//This is triggered by DoorCollide
public void Stuck(Collider collider)
{
    if (collider.gameObject.tag == "Player" && handleCycle > 2 && rotLerp < 1 &&
rotLerp > 0)
    {
        stuck = true;
    }
}

//This is triggered by DoorCollide
public void UnStuck(Collider collider)
{
    if (collider.gameObject.tag == "Player")
    {
        stuck = false;
    }
}
```

OreGen.cs

```

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

public class OreGen : MonoBehaviour
{
    private float sqrMaxNatureDist;
    private int oreCount = 0;
    [SerializeField] private Transform[] bounds;

    private AudioSource pickSource;
    [SerializeField] private AudioClip pickSound;

    public void Init(int _layer, int seed)
    {
        Random.InitState(seed);

        //50% chance no ore is generated on the rock
        bool justRock = Random.value < 0.5f;

        pickSource = GetComponent<AudioSource>();

        gameObject.layer = _layer;
        for (int i = 0; i < transform.childCount - 1; i++)
        {
            transform.GetChild(i).gameObject.layer = _layer;
            //Each ore looks different

            transform.GetChild(i).GetComponent<MeshRenderer>().sharedMaterial.SetFloat("_seed",
            Random.Range(-9999, 9999));
            bool addRock = justRock && Random.value < 0.5f;
            transform.GetChild(i).gameObject.SetActive(addRock);
            if (addRock)
                oreCount++;

        }
        //The base stone is the last child, we want it to be active but also on the
        planet's layer
        transform.GetChild(transform.childCount - 1).gameObject.layer = _layer;

        Random.InitState(System.Environment.TickCount);
    }

    public void SetMaterials(Planet planet, Vector3 relativePosition)
    {
        sqrMaxNatureDist = planet.maxNatureDist * planet.maxNatureDist;

        for (int i = 0; i < transform.childCount; i++)
        {
            Material oreMat = new
Material(transform.GetChild(i).GetComponent<MeshRenderer>().sharedMaterial);
            transform.GetChild(i).GetComponent<MeshRenderer>().sharedMaterial = oreMat;

            oreMat.SetFloat("_minElevation", planet.planetMesh.elevationData.Min);
            oreMat.SetFloat("_maxElevation", planet.planetMesh.elevationData.Max);
            oreMat.SetTexture("_planetTexture",
planet.terrainMat.GetTexture("_planetTexture"));
            oreMat.SetVector("_position", relativePosition);
        }
    }
}

```

```

        }

    private IEnumerator RenderAtLOD(float time)
    {
        while (true)
        {
            yield return new WaitForSeconds(time);

            float LOD = Mathf.InverseLerp(0, sqrMaxNatureDist, (transform.position -
Camera.main.transform.position).sqrMagnitude);

            //If the player is looking at the ore and there is sufficient ore for its
proximity to warrant its highlighting on the UI
            if (LOD < 0.1f * oreCount && Physics.Raycast(transform.position,
Camera.main.transform.position - transform.position, out RaycastHit hitInfo) &&
LayerMask.NameToLayer("Player") == hitInfo.collider.gameObject.layer)
            {
                RequestBounds();
            }

        }
    }

    void RequestBounds()
    {
        RoboVision.TargetBounds targetBounds = new RoboVision.TargetBounds(bounds);
        if (RoboVision.highlightBounds.ID == transform.GetInstanceID() &&
targetBounds.golfScore == float.MaxValue)
            //Forget about highlighting this ore if it is too close or too far
            RoboVision.highlightBounds = new RoboVision.TargetBounds(float.MaxValue);
        else if (RoboVision.highlightBounds.ID == transform.GetInstanceID() ||
targetBounds.golfScore < RoboVision.highlightBounds.golfScore)
            //Update the bounds for this highlighted ore
            RoboVision.highlightBounds = targetBounds;
    }

    void Start()
    {
        StartCoroutine(RenderAtLOD(0.5f));
    }

    private void Update()
    {
        if (oreCount == 0 || !CameraState.CamIsInteractingW(transform.position, 15))
            return;

        for (int i = 0; i < 3; i++)
        {
            if (transform.GetChild(i).gameObject.activeSelf &&
CameraState.CamIsInteractingW(transform.GetChild(i).position, 7))
            {
                //Collect the ore
                InventoryUI.fuelRemaining +=
InventoryUI.player.Stat("robotFuelPerOre");
                InventoryUI.robotMetalCount += 4 + Random.Range(0, 3 - i);
                transform.GetChild(i).gameObject.SetActive(false);
                oreCount--;
                pickSource.PlayOneShot(pickSound);
                break;
            }
        }
    }
}

```

}

PauseMenu.cs

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

public class PauseMenu : MonoBehaviour
{
    [SerializeField] private Text fpsText;
    [SerializeField] private TMPro.TextMeshProUGUI pauseText;
    [SerializeField] private RoboVision roboVision;
    public static int selectedPauseMenuOption;

    bool pauseTextShow;
    string defaultPauseMessage = "-New Game-\n\n-Self Destruct-\n\n-Continue-\n\n-Controls-\n\n-Quit-";
    string pauseMessage = "";
    float pauseLerp;

    int frameCount = 0;
    float dt = 0.0f;
    float fps = 0.0f;
    float updateRate = 4.0f; // 4 updates per sec.

    private void Start()
    {

    }

    void Update()
    {
        UpdateFrameRate();

        if (selectedPauseMenuOption == 4)
        {
            Application.Quit();
            Debug.Log("Quit registered");
        }

        if (pauseTextShow != CameraState.isPaused)
        {
            if (CameraState.isPaused)
                pauseLerp = 0;
            pauseTextShow = CameraState.isPaused;
        }
        pauseLerp += 4 * Time.deltaTime;
        pauseText.enabled = CameraState.isPaused &&
roboVision.cameraIsPausedLerpDirection == -1;

        selectedPauseMenuOption = -1;

        float b = 0.05f * Mathf.Abs(Mathf.Sin(Time.realtimeSinceStartup * 5));
        pauseText.fontSharedMaterial.SetColor("_bgColour", new
Color(RoboVision.visionColour.r * b, RoboVision.visionColour.g * b,
RoboVision.visionColour.b * b, 1));

        if (!CameraState.isPaused)
            return;
    }
}
```

```

        string[] splits = defaultPauseMessage.Split('\n');
        List<string> options = new List<string>();
        foreach (string split in splits)
        {
            if (split != "")
                options.Add(split);
        }

        //Get and highlight currently selected pause menu option
        int i = Mathf.Clamp(options.Count - Mathf.CeilToInt(Input.mousePosition.y /
Screen.height * options.Count), 0, options.Count - 1);
        options[i] = (Input.GetKey(KeyCode.Mouse0) ? "+" : "[" +
options[i].Substring(1, options[i].Length - 2) + (Input.GetKey(KeyCode.Mouse0) ? "+" :
"]"));

        //Set this option when the mouse is clicked
        if (Input.GetKeyUp(KeyCode.Mouse0))
            selectedPauseMenuOption = i;

        string newPauseMessage = string.Join("\n\n", options);

        if (newPauseMessage != pauseMessage)
        {
            pauseMessage = newPauseMessage;
        }

        //Write the option text letter by letter when the menu appears
        pauseText.text = pauseMessage.Substring(0,
Mathf.Min(Mathf.FloorToInt(Mathf.Max(0, pauseLerp - 2) * 20), pauseMessage.Length));
    }
    void UpdateFrameRate()
    {
        //Press Ctrl+F to toggle frame rate view
        fpsText.enabled ^= (Input.GetKey(KeyCode.LeftControl) ||
Input.GetKeyDown(KeyCode.RightControl)) && Input.GetKeyDown(KeyCode.F)
            || (Input.GetKeyDown(KeyCode.LeftControl) ||
Input.GetKeyDown(KeyCode.RightControl)) && Input.GetKey(KeyCode.F);

        frameCount++;
        dt += Time.deltaTime;
        if (dt > 1.0 / updateRate)
        {
            fps = frameCount / dt;
            frameCount = 0;
            dt -= 1.0f / updateRate;
        }
        fpsText.text = "FPS: " + fps;
        fpsText.color = fpsText.color == Color.white ? Color.black : Color.white;
    }
}

```

PhysicsUpdate.cs

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

public class PhysicsUpdate : MonoBehaviour
{
    public static List<Weight> weights;
    public static ZeroWeight[] zeroWeights;
    public static ShipWeight shipWeight;
    public static RobotWeight robotWeight;
    public List<Weight> readonlyWeights;

    private void Start()
    {
        Reinit();
    }
    public static void Reinit()
    {
        weights = new List<Weight>();
        zeroWeights = FindObjectsOfType<ZeroWeight>();
        foreach (ZeroWeight zeroWeight in zeroWeights)
            AddWeight(zeroWeight);

        shipWeight = FindObjectOfType<ShipWeight>();
        robotWeight = FindObjectOfType<RobotWeight>();
    }
    public static void AddWeight(Weight weight)
    {
        if (weight != null && weights != null)
        {
            weights.Add(weight);
            weight.Init();
        }
    }
    public static void RemoveWeight(Weight weight)
    {
        if (weights.Contains(weight))
            weights.Remove(weight);
    }
    private void FixedUpdate()
    {
        if (CameraState.isPaused)
            return;

        readonlyWeights = weights;

        Vector3[] displacements = new Vector3[weights.Count];
        Vector3[] zeroDisplacements = new Vector3[zeroWeights.Length];
        int k = 0;
        int i = 0;

        //Get all displacements due to gravity
        foreach (Weight weight in weights)
            weight.GetGravity(out displacements[k++]);
    }
}
```

```

        //Apply these displacements to weights with mass, and apply displacements of
        the most proximate mass to weights without mass
        k = 0;
        foreach (Weight weight in weights)
        {
            if (weight.mass == 0)
            {
                if (weight.sigWeight != null)
                {
                    int j = weights.IndexOf(weight.sigWeight);
                    weight.Teleport(displacements[j]);
                    zeroDisplacements[i] = displacements[k] - displacements[j];
                }
                else
                    zeroDisplacements[i] = displacements[k];
                i++;
                k++;
            }
            else
                weight.Teleport(displacements[k++]);
        }

        //Collision Accounted for with Relative Displacement (CARD), collision
        detection algorithm works properly for a zeroWeight iff the displacement parameter of
        MoveRelative is relative to every collider significant
        //Apply CARD for Ship first (so it doesn't clip through Planets)
        i = System.Array.IndexOf(zeroWeights, shipWeight);
        Vector3 shipSafeDisplacement = shipWeight.MoveRelative(zeroDisplacements[i]);

        //Apply CARD on Robot relative to Ship if necessary (shipSafeDisplacement
        accounts for robot displacement relative to planet, since the robot is in the ship and
        shipSafeDisplacement is relative to the planet)
        i = System.Array.IndexOf(zeroWeights, robotWeight);
        if (CameraState.inShip)
        {
            robotWeight.Teleport(shipSafeDisplacement);
            if (CameraState.InLockState(CameraState.LockState.unlocked))
                robotWeight.MoveRelative(zeroDisplacements[i] - shipSafeDisplacement);
        }
        else
            robotWeight.MoveRelative(zeroDisplacements[i]);

        //Apply CARD to the other zero weights last
        i = 0;
        foreach (ZeroWeight weight in zeroWeights)
        {
            if (weight != robotWeight && weight != shipWeight)
                weight.MoveRelative(zeroDisplacements[i]);
            i++;
        }

        robotWeight.UpdateForceMeter();
        //Camera close to origin minimises floating point error in camera position
        (stops jittering)
        Vector3 origin = robotWeight.position;

        StarGenSystem.galaxyCentre -= origin;
        foreach (Weight weight in weights)
            weight.Teleport(-origin, true);
    }
}

```

Planet.cs

```

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

public class Planet : MonoBehaviour
{
    public RobotWeight robotWeight; //NOTE THIS VARIABLE IS INITIALISED BY THE FIRST
    INTERFACE THAT ACCESSES IT

    public MeshGen planetMesh { get; private set; }
    public PlanetEffect atmosphere;
    public GameObject[] planetMeshSegmentPrefabs;
    private ColourGenerator colourGenerator;

    private MiniMap miniMap;
    public GameObject globeMapPrefab;

    public Material terrainMat { get; private set; }
    public Material oceanMat { get; private set; }
    public Craters[] craters { get; private set; }
    public readonly int resolution = 600;
    public readonly int splitFaces = 12;

    [Header("Manually Set Objects")]
    public ComputeShader noiseShader;
    [SerializeField] private Material terrainPrefab;
    [SerializeField] private Material oceanPrefab;
    public Gradient heatGrad1;
    public Gradient heatGrad2;

    [Header("Nature")]
    public GameObject Tree;
    public GameObject Ore;
    public GameObject Tower;

    [Tooltip("X: Start of random range Y: End inclusive")]
    public Vector2 treeCountPer500Seg;
    [Tooltip("X: Start of random range Y: End inclusive")]
    public Vector2 oreCountPer500Seg;
    [Tooltip("X: Start of random range Y: End inclusive")]
    public Vector2 grassCountPer500Seg;
    [Tooltip("X: Start of random range Y: End inclusive")]
    public Vector2 stoneCountPer500Seg;
    public float maxNatureDist;

    private float renderPlanetSegsBelowSqrDist = 100_000_000; //10_000^2
    private float dontRenderPlanetSegsAboveSqrDist = 6_400_000_000; //80_000^2

    [System.Serializable]
    public class SetPlanetValues
    {
        [Tooltip("12 lower-case letters")]
        public string masterSeed;

        public float seabedLevel;
        public bool roughBed;
    }
}

```

```

        public bool crackedGround;
        public int mtInitialExtraOctaves;
        public float readonlyplanetRadius;
        public float readonlyTemperature;
        public Texture readonlyPlanetTexture;
    }

    [System.Serializable]
    public class SetCraterValues
    {
        public int numCraters;
        public float smoothness;
        public float rimWidth;
        public float rimSteepness;
    }

    public SetPlanetValues setPlanetValues;
    public SetCraterValues setCraterValues;
    public PlanetValues planetValues;
    public CraterValues craterValues;
    public ContinentNoiseValues continentNoise;
    public MountainNoiseValues mountainNoise;
    public WarpNoiseValues warpNoise;
    public RoughNoiseValues roughNoise;

    public PlanetSettings planetSettings { get; private set; }

    public void Init()
    {
        for (int i = 0; i < transform.childCount; i++)
            transform.GetChild(i).gameObject.layer = gameObject.layer;

        terrainMat = new Material(terrainPrefab);
        oceanMat = new Material(oceanPrefab);

        //transform.GetChild(1).GetComponent<MeshRenderer>().sharedMaterial =
        oceanMat;

        planetSettings = new PlanetSettings(this);
        colourGenerator = new ColourGenerator(this);

        //Set the ocean & atmosphere size to the radius of the planet
        transform.GetChild(2).localScale = Vector3.one * planetValues.radius;

        planetMesh = new MeshGen(this);
        //float start = (float)EditorApplication.timeSinceStartup;
        planetMesh.ApplyQuadSphere();
        //Debug.Log("Quadsphere generated in: " +
        ((float)EditorApplication.timeSinceStartup - start) + "s");

        planetSettings.SyncSettings();
    }
    public void UpdateTerrain()
    {
        //only show the ocean if roughBed is false;
        //transform.GetChild(1).gameObject.SetActive(!planetValues.roughBed);

        craters = GetCrators(craterValues);
        //float start = (float)EditorApplication.timeSinceStartup;
        planetMesh.UpdateNoiseGPU();
        //Debug.Log("Noise generated in: " +
        ((float)EditorApplication.timeSinceStartup - start) + "s");
    }
}

```

```

        colourGenerator.UpdateElevation(planetMesh.elevationData);
    }

    public Craters[] GetCraters(CraterValues craterValues)
    {
        Random.InitState(craterValues.seed);

        Craters[] craters = new Craters[craterValues.numCraters];
        for (int i = 0; i < craterValues.numCraters; i++)
        {
            craters[i] = new Craters();
            craters[i].floorHeight = Random.Range(-1f, 0);
            craters[i].radius = 0.1f + Random.Range(0f, 1) * Random.Range(0f, 1) *
0.1f;
            craters[i].point = Random.onUnitSphere;
        }
        return craters;
    }

    private void UpdateMaterials()
    {
        planetSettings.SyncValues();
        colourGenerator.UpdateColours();
        colourGenerator.UpdatePlanetInfo();

        planetSettings.SyncValues();

        atmosphere.ResetMaterial();

        if (miniMap != null)
            miniMap.UpdatePointers();
    }

    public GameObject GenerateMiniMap(GameObject _Map)
    {
        miniMap = new MiniMap(this, _Map);
        miniMap.CreatePlanetMap();

        return miniMap.Map;
    }

    public void Create(Vector3 position, Vector3 initialVelocity)
    {
        transform.position = position;
        GetComponent<Weight>().initialVelocity = initialVelocity;

        for (int i = 0; i < transform.GetChild(0).childCount; i++)
            transform.GetChild(0).GetChild(i).GetComponent<Segment>().RemoveFoliage();

        Init();

        UpdateTerrain();
        UpdateMaterials();
    }

    public void Create(Vector3 position, Vector3 initialVelocity, System.Random prng)
    {
        transform.position = position;
        GetComponent<Weight>().initialVelocity = initialVelocity;

        for (int i = 0; i < transform.GetChild(0).childCount; i++)
            transform.GetChild(0).GetChild(i).GetComponent<Segment>().RemoveFoliage();
    }
}

```

```

        Init();
        planetSettings.RandomiseTerrainSeed(prng.Next(-9999, 9999));
        planetSettings.RandomiseColourSeed(prng.Next(-9999, 9999));
        planetSettings.RandomiseEnvironmentSeed(prng.Next(-9999, 9999));
        planetSettings.SyncSettings();

        UpdateTerrain();
        UpdateMaterials();
    }

private void Update()
{
    if (colourGenerator != null)
        colourGenerator.UpdatePlanetInfo();

    Vector3 viewPoint = Camera.main.WorldToViewportPoint(transform.position);

    //Show the planet segments if (player is not in hive) AND (player is close
    enough OR player is not quite close enough but player is looking at planet) AND (if
    robot is within a planet then this is that planet)
    transform.GetChild(0).gameObject.SetActive(
        !CameraState.inHive
        &&
        (transform.position.sqrMagnitude < renderPlanetSegsBelowSqrDist ||
        transform.position.sqrMagnitude < dontRenderPlanetSegsAboveSqrDist &&
        viewPoint.x > -0.1f && viewPoint.x < 1.1f && viewPoint.y > -0.1f &&
        viewPoint.y < 1.1f && viewPoint.z > 0)
        &&
        (robotWeight.sigWeight == null || robotWeight.sigWeight.planet == this ||
        robotWeight.sigWeight.position.sqrMagnitude >
        robotWeight.sigWeight.planet.atmosphere.atmosRadius *
        robotWeight.sigWeight.planet.atmosphere.atmosRadius)
    );

    transform.GetChild(2).gameObject.SetActive(!CameraState.inHive);
    transform.GetChild(4).gameObject.SetActive(!CameraState.inHive);
    transform.GetChild(5).gameObject.SetActive(!CameraState.inHive);
}
}

```

PlanetEffect.cs

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

#if UNITY_EDITOR
[ExecuteAlways]
#endif
public class PlanetEffect : MonoBehaviour
{
    public float cameraSqrDist { get; private set; }

    public bool active;

    [Space(20)]
    public float planetRadius; //updated in ColourGenerator.UpdatePlanetInfo()
    public float maxFogRange; //updated in ColourGenerator.UpdatePlanetInfo()
    public Transform sun;

    [Space(20)]
    public float atmosRadius;
    public Color atmosColour; //set to ocean colour in ColourGenerator.UpdateColours()
    [Tooltip("Mix between red and sun colour")]
    public Color sunsetColour;
    public int environmentSeed; //set to environment seed in PlanetSettings.SyncValues()
    [Range(0, 6)]
    public float density;

    [Space(20)]
    [Tooltip("Between 30 and (atmosRadius & planetRadius) * (1 / density)")]
    public float fogRange;
    [Range(0, 10)]
    public float noiseFreq;
    [Range(0, 0.5f)]
    public float noiseAmp;
    [Range(0, 0.5f)]
    public float noiseBlend;

    [Space(20)]
    [Range(1, 10)]
    public float dispersionPower;
    [Range(0, 1)]
    public float dispersionScale;

    public Texture2D waveNormalA;
    public Texture2D waveNormalB;
    public float waveSpeed;
    public float waveStrength;
    public float waveNormalScale;
    public float smoothness;
    public float visibleDepth;

    private int numRings = 1;
    public Color ringColour; //set to ocean colour in ColourGenerator.UpdateColours()
    private Color ringColour1;
    private Color ringColour2;
    private Vector3 ringNormal1;
    private Vector3 ringNormal2;
```

```
private float rings;
private Vector2 iris1;
private Vector2 iris2;

public Material material { get; private set; }
[Space(20)]
private Shader effectShader;
[SerializeField] private Shader atmosphereShader;
[SerializeField] private Shader cloudShader;
[SerializeField] private Gradient atmosGradient;
[SerializeField] private Texture2D atmosTexture;

[SerializeField] private Texture2D blueNoise;
[SerializeField] private Texture2D cloudMap;
[SerializeField] private float ditherStrength;
[SerializeField] private float ditherScale;

private int textureRes = 100;

public void UpdateInfo()
{
    cameraSqrDist = (Camera.main.transform.position -
transform.position).sqrMagnitude;
}
public Material GetMaterial()
{
    if (atmosTexture == null || material == null)
    {
        SetShaderVars();
        if (material == null || effectShader.name != material.shader.name)
            material = new Material(effectShader);
    }

    SetMaterial();

    return material;
}

public void OnValidate()
{
    ResetMaterial();
}

public void ResetMaterial()
{
    SetShaderVars();
    if (material == null || effectShader.name != material.shader.name)
        material = new Material(effectShader);
    SetMaterial();
}

private void SetMaterial()
{
    material.SetFloat("numDiscs", numRings);
    material.SetFloat("rings", rings);
    material.SetVector("iris1", iris1);
    material.SetVector("iris2", iris2);
    material.SetVector("disc1Normal", ringNormal1);
    material.SetVector("disc2Normal", ringNormal2);
    material.SetVector("disc1Colour", ringColour1);
    material.SetVector("disc2Colour", ringColour2);
```

```

        material.SetVector("sunCentre", sun.position);
        material.SetVector("planetCentre", transform.parent.position);
        material.SetFloat("atmosphereRadius", atmosRadius);
        material.SetFloat("planetRadius", planetRadius);

        material.SetFloat("oceanRadius", planetRadius *
            (transform.parent.GetComponent<Planet>().planetValues.roughBed &&
            transform.parent.GetChild(0).gameObject.activeSelf ?
            transform.parent.GetComponent<Planet>().planetValues.seabedLevel : 0.99f));

        material.SetFloat("fogRange", fogRange);
        material.SetTexture("fogColourRings", atmosTexture);

        material.SetFloat("noiseFreq", noiseFreq);
        material.SetFloat("noiseAmp", noiseAmp);

        material.SetTexture("CloudMap", cloudMap);

        material.SetTexture("BlueNoise", blueNoise);
        material.SetFloat("ditherScale", ditherScale);
        material.SetFloat("ditherStrength", ditherStrength);

        material.SetFloat("density", density);
        material.SetFloat("dispersionPower", dispersionPower);
        material.SetFloat("dispersionScale", dispersionScale);

        material.SetFloat("windSpeed",
            transform.parent.GetComponent<Planet>().planetValues.windSpeed);
            material.SetFloat("roughBed",
            transform.parent.GetComponent<Planet>().planetValues.roughBed ? 1 : 0);

        material.SetFloat("cloudAltitude", 700);
        material.SetFloat("cloudLyrDepth", 25);

        material.SetTexture("waveNormalA", waveNormalA);
        material.SetTexture("waveNormalB", waveNormalB);

        Color.RGBToHSV(atmosColour, out float H, out float S, out _);
        material.SetVector("oceanColour", Color.HSVToRGB(H, S, 0.5f));
        material.SetFloat("visibleDepth", visibleDepth);
        material.SetFloat("waveSpeed", waveSpeed);
        material.SetFloat("waveStrength", waveStrength);
        material.SetFloat("waveNormalScale", waveNormalScale);
        material.SetFloat("smoothness", smoothness);
    }

private void SetShaderVars()
{
    Random.initState(environmentSeed);

    //Rings
    numRings = Random.Range(0, 3);

    ringColour1 = Color.Lerp(ringColour, Random.ColorHSV(), 0.1f);
    ringColour2 = Color.Lerp(ringColour, Random.ColorHSV(), 0.1f);

    ringNormal1 = Random.onUnitSphere;
    ringNormal2 = Random.onUnitSphere;

    rings = 80;

    float a = Random.value * 0.4f + 0.4f;
}

```

```

        float b = Random.Range(a + 0.1f, a + 0.3f);
        iris1 = new Vector2(a, b);

        a = Random.value * 0.4f + 0.4f;
        b = Random.Range(a + 0.1f, a + 0.3f);
        iris2 = new Vector2(a, b);

        float h, s, v;
        //Atmosphere
        if (FindObjectOfType<SunGenSystem>() != null)

Color.RGBToHSV(FindObjectOfType<SunGenSystem>().transform.GetChild(1).GetComponent<Mes
hRenderer>().sharedMaterial.GetColor("_sunColour"), out h, out s, out v);
        else
            (h, s, v) = (0, 0, 0);
        sunsetColour = Color.HSVToRGB(h * 0.5f, s, v);

        density = Mathf.Lerp(0.4f, 6, Mathf.Pow(Mathf.Sin(Random.value * Mathf.PI /
2), 3));

        effectShader = density > 3 && Random.value < 0.5f ? cloudShader :
atmosphereShader; //~25% chance of clouds

        atmosRadius = Mathf.Lerp(1.1f * planetRadius, 2 * planetRadius,
Mathx.MiddleCommon(Random.value));
        fogRange = Random.Range(30, Mathf.Min(maxFogRange, atmosRadius -
planetRadius));

        noiseFreq = Random.value * Random.value * 10;
        noiseAmp = Random.value * Random.value * 0.5f;
        noiseBlend = Random.value * Random.value * 0.5f;

        int numKeys = 8;

        atmosGradient = new Gradient();
        GradientColorKey[] colourKey = new GradientColorKey[numKeys];
        GradientAlphaKey[] alphaKeys = new GradientAlphaKey[numKeys];

        for (int i = 0; i < numKeys; i++)
        {
            float i01 = (float)i / (numKeys - 1);

            if (i == numKeys - 1)
                colourKey[i].color = colourKey[0].color; //smooth wrap around
            else
                colourKey[i].color = Color.Lerp(atmosColour, Random.ColorHSV(),
noiseBlend);

            colourKey[i].time = Mathx.MiddleCommon(i01) + Random.Range(-0.01f, 0.01f);
            alphaKeys[i].alpha = 1;
        }

        atmosGradient.SetKeys(colourKey, alphaKeys);

        Color.RGBToHSV(sunsetColour, out float lerpH, out _, out _);

        Color[] colours = new Color[textureRes * textureRes];

        //Atmosphere texture with hue on y-axis (redder hue for sunset) and latitude
        on x-axis (for colour bands)
        for (int hue = 0; hue < textureRes; hue++)
        {

```

```
        for (int i = 0; i < textureRes; i++)
    {
        Color.RGBToHSV(atmosGradient.Evaluate((float)i / (textureRes - 1)),
out float H, out float S, out float V);

        //Color.RGBToHSV(Color.Lerp(Color.HSVToRGB(H, 1, 1),
Color.HSVToRGB(lerpH, 1, 1), (float)hue / (textureRes - 1)), out H, out _, out _);
        //H = Mathf.Lerp(H, lerpH, 1 - Mathf.Cos(Mathf.PI * hue / (textureRes
- 1)));

        H = Mathf.Lerp(H, lerpH, (float)hue / (textureRes - 1));// ;
        colours[hue * textureRes + i] = Color.HSVToRGB(H, S, V);
    }
}

atmosTexture = new Texture2D(textureRes, textureRes);
atmosTexture.SetPixels(colours);
atmosTexture.Apply();

Random.InitState(System.Environment.TickCount);
}
```

PlanetGeneral.cs

```

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

public struct Mathx
{
    //Continous function where a random value01 is more likely to be closer to 0.5 than 0 or 1: https://www.desmos.com/calculator/3rani8m1v0
    public static float MiddleCommon(float value01)
    {
        if (value01 > 1 || value01 < 0)
        {
            Debug.LogError("Error: float parameter \"value01\" in function \"MiddleCommon\" must be between 0 and 1 inclusive");
            return -1;
        }

        return value01 < 0.5f ? 0.5f * Mathf.Sin(value01 * Mathf.PI) : 1 - 0.5f * Mathf.Sin(value01 * Mathf.PI);
    }

    //Continous function where a random value01 is more likely to be closer to 0 or 1 than 0.5: https://www.desmos.com/calculator/3rani8m1v0
    public static float EndsCommon(float value01)
    {
        if (value01 > 1 || value01 < 0)
        {
            Debug.LogError("Error: float parameter \"value01\" in function \"MiddleCommon\" must be between 0 and 1 inclusive");
            return -1;
        }

        return 0.5f * Mathf.Sin((value01 - 0.5f) * Mathf.PI) + 0.5f;
    }

    private static List<char> alphaIndexLookup = new List<char>(26) { 'a', 'b', 'c', 'd',
    'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
    'w', 'x', 'y', 'z' };
    public static string IntToAlpha26(int i)
    {
        string alpha = "";
        alpha += alphaIndexLookup[i % 26];
        i /= 26;
        if (i != 0)
            alpha += IntToAlpha26(i);
        return alpha;
    }
    public static int Alpha26ToInt(string alpha)
    {
        List<int> alphaInts = new List<int>(alpha.Length);
        foreach (char _a in alpha)
        {
            int index = alphaIndexLookup.IndexOf(_a);
            if (index < 0)
                Debug.LogError("Error: " + _a + " is not a lower case letter, using 'z' instead");
            alphaInts.Add(index);
        }
    }
}

```

```
int Alpha26ToInt(int q, int n)
{
    return n < alpha.Length ? Alpha26ToInt(q * 26 + alphaInts[alpha.Length -
(n + 1)], n + 1) : q;
}

return Alpha26ToInt(0, 0);
}
```

PlanetLighting.cs

```

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

#if UNITY_EDITOR
[ExecuteAlways]
#endif
public class PlanetLighting : MonoBehaviour
{
    [SerializeField] private Transform sun;
    [SerializeField] private RobotWeight robotWeight;
    [SerializeField] private float lightIntensityDropoff;
    [SerializeField] private float maxIntensity;
    [SerializeField] private float minIntensity;
    [SerializeField] private float maxIntensityAtDistance;
    [SerializeField] private float minIntensityAtDistance;
    private float minAtSqrDist;
    private float maxAtSqrDist;
    private Light sunlight;
    private bool isPlanet;
    private PlanetEffect atmosphere;

    private void Start()
    {
        minAtSqrDist = minIntensityAtDistance * minIntensityAtDistance;
        maxAtSqrDist = maxIntensityAtDistance * maxIntensityAtDistance;

        sunlight = GetComponent<Light>();
        isPlanet = transform.parent.GetChild(2).TryGetComponent(out atmosphere);
    }

    //Only want to light this planet (and the player and ship if they are on it)
    private void Update()
    {
        sunlight.cullingMask = 0;

        bool lightingPlayer;

        if (isPlanet)
        {
            sunlight.cullingMask |= 1 << gameObject.layer;
            lightingPlayer = robotWeight.sigWeight != null &&
transform.IsChildOf(robotWeight.sigWeight.transform);
        }
        else if (robotWeight.sigWeight != null)
        {
            lightingPlayer = false;
            atmosphere = robotWeight.sigWeight.planet.atmosphere;
        }
        else
        {
            lightingPlayer = true;
            atmosphere = null;
        }

        //If lighting player then mask player and ship outside (layers 6 & 12)
        if (lightingPlayer)
    }
}

```

```
sunlight.cullingMask |= 1 << 6 | 1 << 12;

lightIntensityDropoff = 1 - Mathf.InverseLerp(0.4f, 6, atmosphere == null ?
0.4f : atmosphere.density);

transform.rotation = Quaternion.LookRotation(transform.position -
sun.position);
    float t = Mathf.InverseLerp(maxAtSqrDist, minAtSqrDist, (transform.position -
sun.position).sqrMagnitude);
    sunlight.intensity = lightIntensityDropoff * Mathf.Lerp(maxIntensity,
minIntensity, t);
    sunlight.shadowStrength = lightIntensityDropoff;
}

private void OnValidate()
{
    Start();
}
}
```

[PlanetSettings.cs](#)

```

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

public class PlanetSettings
{
    private Planet planet;
    private PlanetValues planetValues;
    private CraterValues craterValues;
    private ContinentNoiseValues continentNoise;
    private MountainNoiseValues mountainNoise;
    private WarpNoiseValues warpNoise;
    private RoughNoiseValues roughNoise;

    public PlanetSettings(Planet _planet)
    {
        planet = _planet;
        continentNoise.octaves = 3;
        mountainNoise.octaves = 3;
        warpNoise.octaves = 3;
        roughNoise.octaves = 4;
        roughNoise.startingOctave = 5;

        planetValues.terrainSeed =
            Mathx.Alpha26ToInt(planet.setPlanetValues.masterSeed.Substring(0, 4));
        planetValues.colourSeed =
            Mathx.Alpha26ToInt(planet.setPlanetValues.masterSeed.Substring(4, 4));
        planetValues.environmentSeed =
            Mathx.Alpha26ToInt(planet.setPlanetValues.masterSeed.Substring(8, 4));
        planetValues.seabedLevel = planet.setPlanetValues.seabedLevel;

        mountainNoise.initialExtraOctaves = planet.setPlanetValues.mtInitialExtraOctaves;

        craterValues.numCraters = planet.setCraterValues.numCraters;
        craterValues.smoothness = planet.setCraterValues.smoothness;
        craterValues.rimWidth = planet.setCraterValues.rimWidth;
        craterValues.rimSteepness = planet.setCraterValues.rimSteepness;
    }
    public void SyncSettings()
    {
        planetValues.crackedGround = planet.setPlanetValues.crackedGround;

        GetTerrain();
        GetGradients();

        SyncValues();
    }

    public void RandomiseTerrainSeed(int seed = int.MaxValue)
    {
        if (seed == int.MaxValue)
            seed = Random.Range(-9999, 9999);
        Random.InitState(seed);
        planetValues.terrainSeed = Random.Range(0, 456975); // (26^4 - 1) is zzzz in
alpha26 (and thus the max value the seed can take)
        Random.InitState(System.Environment.TickCount);
    }
    public void RandomiseColourSeed(int seed = int.MaxValue)

```

```

{
    if (seed == int.MaxValue)
        seed = Random.Range(-9999, 9999);
    Random.InitState(seed);
    planetValues.colourSeed = Random.Range(0, 456975); //(26^4 - 1) is zzzz in
alpha26 (and thus the max value the seed can take)
    Random.InitState(System.Environment.TickCount);
}
public void RandomiseEnvironmentSeed(int seed = int.MaxValue)
{
    if (seed == int.MaxValue)
        seed = Random.Range(-9999, 9999);
    Random.InitState(seed);
    planetValues.environmentSeed = Random.Range(0, 456975); //(26^4 - 1) is zzzz
in alpha26 (and thus the max value the seed can take)
    Random.InitState(System.Environment.TickCount);
}

private void GetTerrain()
{
    Random.InitState(planetValues.terrainSeed);

    planetValues.radius = Random.Range(250, 1000);
    planet.transform.GetChild(3).localScale = planetValues.radius * Vector3.one;

    continentNoise.seed = Random.Range(-9999, 9999);
    mountainNoise.seed = Random.Range(-9999, 9999);
    warpNoise.seed = Random.Range(-9999, 9999);
    craterValues.seed = Random.Range(-9999, 9999);
    roughNoise.seed = Random.Range(-9999, 9999);

    planetValues.groundLevel = Random.Range(1, 1.02f);
    planetValues.windSpeed = Random.Range(0.5f, 2);

    continentNoise.scale = Random.Range(0.1f, 2.5f);
    continentNoise.persistance = Random.Range(0.1f, 0.3f);
    continentNoise.lacunarity = Random.Range(5f, continentNoise.persistance > 0.2f
? 7.5f : 10);
    continentNoise.dropoff = Random.Range(0.1f, 0.2f);

    mountainNoise.scale = Random.Range(0.1f, 2);
    mountainNoise.persistance = Random.Range(0.1f, 0.15f);
    mountainNoise.lacunarity = Random.Range(5f, 10);
    mountainNoise.dropoff = Random.Range(0.25f, 0.35f);

    warpNoise.scale = Random.Range(0f, 2);
    warpNoise.persistance = Random.Range(0f, 1);
    warpNoise.lacunarity = Random.Range(1f, 3);

    roughNoise.persistance = Random.Range(0.5f, 0.6f);
    roughNoise.lacunarity = Random.Range(1.5f, 1.7f);

    Random.InitState(System.Environment.TickCount);
}

private void GetGradients()
{
    Random.InitState(planetValues.colourSeed);

    planetValues.temperature = Mathx.MiddleCommon(Mathf.InverseLerp(10000, 50000,
(Object.FindObjectOfType<SunGenSystem>()).transform.position -
planet.transform.position).magnitude));
}

```

```

    planetValues.roughBed = planetValues.temperature < 0.2f ||  

    planetValues.temperature > 0.8f || Random.value < 0.5f;  
  

        //Zero if the planet has a rough bed (idea being there is no large water  

        supply on the planet).  

        planet.treeCountPer500Seg = planetValues.roughBed ? new Vector2(0, 0) : new  

        Vector2(0.1f, 0.9f);  

        planet.grassCountPer500Seg = planetValues.roughBed ? new Vector2Int(0, 1) :  

        new Vector2Int(2, 6);  
  

        int numBiomes = 3;  

        planetValues.biomeGradients = new Dictionary<float, Gradient>(3);  

        float heatLerp = Random.value;  

        float biomeHeat = planetValues.temperature;  
  

        for (int i = 0; i < numBiomes; i++)  

        {  

            planetValues.biomeGradients.Add(i == 0 ? 0 : Random.value,  

            NewBiomeGradient(Mathf.Clamp01(biomeHeat + Random.Range(-0.1f, 0.1f)),  

            Mathf.Clamp01(heatLerp + Random.Range(-0.1f, 0.1f))));  

        }  
  

        Random.InitState(System.Environment.TickCount);  

    }  
  

    private Gradient NewBiomeGradient(float biomeTemperature, float heatLerp)  

    {  

        Color sunColour =  

Object.FindObjectOfType<SunGenSystem>().transform.GetChild(1).GetComponent<MeshRenderere  

r>().sharedMaterial.GetColor("_sunColour");  
  

        Gradient heatGrad = new Gradient();  

        GradientColorKey[] heatColourKeys = new GradientColorKey[4];  

        GradientAlphaKey[] heatAlphaKeys = new GradientAlphaKey[4];  

        int i = 0;  

        //Blend between the two fixed heat gradients by heatLerp and store in heatGrad  

        foreach (float _t in new float[4] { 0, 0.4f, 0.6f, 1 })  

        {  

            float t = Mathf.Clamp01(_t + Random.Range(-0.1f, 0.1f));  

            heatColourKeys[i].color = Color.Lerp(planet.heatGrad1.Evaluate(t),  

            planet.heatGrad2.Evaluate(t), heatLerp);  

            heatColourKeys[i].time = t;  

            heatAlphaKeys[i].alpha = 1;  

            i++;  

        }  

        heatGrad.SetKeys(heatColourKeys, heatAlphaKeys);  
  

        float lowerMean01 = planet.planetMesh.elevationData.lowerMean01;  

        float upperMean01 = planet.planetMesh.elevationData.upperMean01;  
  

        Gradient gradient = new Gradient();  
  

        int keys = 6;  

        //Initial time is smaller if terrain is more common at lower altitudes, this  

        concentrates more colours at lower altitudes if necessary  

        float time = Mathf.Lerp(0, lowerMean01, Random.value * Random.value);  

        //Want at least one new colour beyond the upperMean  

        float minTimeInc = (upperMean01 - time) / (keys - 1);  

        //Want no new colours in the top 20% of altitudes, we will manually add these  

        float maxTimeInc = (0.8f - time) / (keys - 1);  
  

        float temperature = biomeTemperature;
    }
}

```

```

//Temperature decreases with altitude
float temperatureInc = Mathf.Min(0.2f, (1 - temperature)) / (keys - 1);

GradientColorKey[] colourKeys = new GradientColorKey[keys + 2];
GradientAlphaKey[] alphaKeys = new GradientAlphaKey[keys + 2];
for (int j = 0; j < keys; j++)
{
    colourKeys[j].time = time;

    //A colour at a time on the gradient is a 50% mix of the appropriate
    temperature colour for the time and a random colour (makes planets more visually
    unique while still being coherent)
    colourKeys[j].color = Color.Lerp(heatGrad.Evaluate(temperature),
RandomColourVarient(sunColour, 0.25f), 0.5f);
    alphaKeys[j].alpha = 1;

    time += Random.Range(minTimeInc, maxTimeInc);
    temperature += Random.Range(0, temperatureInc);
    temperature = Mathf.Clamp(temperature, 0, 1);
}
//Manually add colours in the top 20% of altitudes (to resemble the cold white
tips at the top of mountains)
colourKeys[keys].time = 0.8f;
colourKeys[keys].color = colourKeys[keys - 1].color;
alphaKeys[keys].alpha = 1;
colourKeys[keys + 1].time = 0.85f;
colourKeys[keys + 1].color =
Color.Lerp(heatGrad.Evaluate(Mathf.Clamp01(biomeTemperature + 0.5f)),
RandomColourVarient(sunColour, 0.25f), 0.5f);
alphaKeys[keys + 1].alpha = 1;

gradient.SetKeys(colourKeys, alphaKeys);
return gradient;
}

private Color RandomColourVarient(Color original, float lerp)
{
    Color.RGBToHSV(original, out float h, out _, out_);
    float newH = Random.value * (1 - lerp) + h * lerp;
    return Color.HSVToRGB(newH, Random.value, Random.value);
}

public void SyncValues()
{
    planet.setPlanetValues.masterSeed =
Mathx.IntToAlpha26(planetValues.terrainSeed) +
Mathx.IntToAlpha26(planetValues.colourSeed) +
Mathx.IntToAlpha26(planetValues.environmentSeed);
    while (planet.setPlanetValues.masterSeed.Length < 12)
        planet.setPlanetValues.masterSeed = "a" +
planet.setPlanetValues.masterSeed;

    planet.atmosphere.environmentSeed = planetValues.environmentSeed;

    planet.transform.GetComponent<Weight>().SetMass(10 + 20 *
Mathf.InverseLerp(0.4f, 6, planet.atmosphere.density), planetValues.radius);
    planet.setPlanetValues.roughBed = planetValues.roughBed;

    planet.planetValues = planetValues;
    planet.craterValues = craterValues;
    planet.continentNoise = continentNoise;
}

```

```
planet.mountainNoise = mountainNoise;
planet.warpNoise = warpNoise;
planet.roughNoise = roughNoise;

planet.setPlanetValues.readonlyplanetRadius = planetValues.radius;
planet.setPlanetValues.readonlyTemperature = planetValues.temperature;
}

}
```

RaiseSeat.cs

```

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

public class RaiseSeat : MonoBehaviour
{
    private AudioSource liftSource;
    [SerializeField] private ShipWeight shipWeight;
    [SerializeField] private FlipSwitch trigger;
    [SerializeField] private float topHeight = 1.5f;
    [SerializeField] private float bottomHeight = 0.3f;
    [SerializeField] private Vector3 ogLocalPos;
    [SerializeField] private Vector3 prevSpeed;
    [SerializeField] private Vector3 deltaSpeed;

    private float lerp;
    [SerializeField] private float lerpSpeed = 1;

    [SerializeField] private bool interacting = false;

    private void Start()
    {
        liftSource = GetComponent<AudioSource>();
        ogLocalPos = transform.localPosition;
        CameraState.flyingShip = false;
        lerp = 0;
    }

    private void Update()
    {
        if (trigger.switchState == FlipSwitch.State.top && lerp < 1)
            lerp += lerpSpeed * Time.deltaTime;
        else if (trigger.switchState == FlipSwitch.State.bottom && lerp > 0)
            lerp -= lerpSpeed * Time.deltaTime;

        transform.GetChild(0).localScale = new Vector3(transform.GetChild(0).localScale.x
            , transform.GetChild(0).localScale.y, Mathf.Lerp(bottomHeight, topHeight, lerp));

        if (!liftSource.isPlaying && lerp < 1 && lerp > 0)
            PlayLift();

        if (!interacting && CameraState.InLockState(CameraState.LockState.unlocked) &&
            (transform.position - Camera.main.transform.position).sqrMagnitude < 1 &&
            trigger.switchState == FlipSwitch.State.top && lerp >= 1)
        {
            interacting = true;
            CameraState.LockCamera(transform);
            CameraState.flyingShip = true;
        }
        else if (interacting && CameraState.InLockState(CameraState.LockState.locked) &&
            Input.GetKeyDown(KeyCode.E))
        {
            CameraState.UnlockCamera();
            CameraState.flyingShip = false;
            //but still interacting
        }
        else if (interacting && (transform.position -
            Camera.main.transform.position).sqrMagnitude > 1)
    }
}

```

```

    {
        interacting = false;
        if (trigger.switchState == FlipSwitch.State.top)
            trigger.override = true;
    }

    transform.localPosition = ogLocalPos;
    //If flying ship, move the target position of the camera based on the delta
    speed so the player leans backward while accelerating forward for instance
    if (CameraState.flyingShip)
    {
        Vector3 newSpeed = shipWeight.velocity;
        Vector3 newDelta = newSpeed - prevSpeed;
        if (newDelta.sqrMagnitude > 16)
            newDelta = 4 * newDelta.normalized;

        deltaSpeed = Vector3.Lerp(deltaSpeed, 0.025f * newDelta, 0.05f);

        transform.position -= deltaSpeed;
        prevSpeed = newSpeed;
    }
}

void PlayLift()
{
    StartCoroutine(CoPlayLift());
}
IEnumerator CoPlayLift()
{
    float startVolume = liftSource.volume;

    liftSource.volume = 0;
    liftSource.Play();

    while (liftSource.volume < startVolume)
    {
        liftSource.volume += startVolume * Time.deltaTime * 5;
        yield return new WaitForEndOfFrame();
    }

    yield return new WaitUntil(new System.Func<bool>(() => lerp <= 0 || lerp >=
1));

    while (liftSource.volume > 0)
    {
        liftSource.volume -= startVolume * Time.deltaTime * 5;
        yield return new WaitForEndOfFrame();
    }

    liftSource.Stop();
    liftSource.volume = startVolume;
}
}

```

RampOpen.cs

```

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

public class RampOpen : MonoBehaviour
{
    private ShipWeight shipWeight;
    [SerializeField] private ShipDoorOpen doorToFollow;
    private readonly float closedLength = 0.1f;
    private float openLength = 3f;
    private bool calculated;

    private void Start()
    {
        shipWeight = FindObjectOfType<ShipWeight>();
    }

    private void LateUpdate()
    {
        if (doorToFollow.lerp > 0 && doorToFollow.lerp < 1)
            CalculateRampEnd();
        else
            calculated = false;

        if (shipWeight.onHelipad)
            openLength = 2;

        transform.GetChild(0).localScale = new
        Vector3(transform.GetChild(0).localScale.x, Mathf.Lerp(closedLength, openLength,
        doorToFollow.lerp), transform.GetChild(0).localScale.z);
    }

    void CalculateRampEnd()
    {
        if (calculated)
            return;

        calculated = true;

        transform.localRotation = Quaternion.Euler(90, 0, 90);
        bool flag = false;
        float angle = 0;

        //Get shortest length and flattest angle required by the ramp to hit the terrain
        for (openLength = 10; openLength < 20; openLength++)
        {
            for (angle = -30; angle > -50; angle--)
            {
                Vector3 fwd = Quaternion.Euler(angle, 0, 0) * -transform.right;

                if (Physics.Raycast(transform.position, fwd, out RaycastHit hitInfo,
                openLength, ~(1 << 11 | 1 << 12)))
                {
                    openLength = hitInfo.distance * 0.7f;
                    flag = true;
                    break;
                }
            }
        }
    }
}

```

```

        if (flag)
            break;
    }

    transform.localRotation = Quaternion.Euler(90 + angle, 0, 90);
}
}

```

RoboEyes.cs

```

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

public class RoboEyes
{
    [SerializeField] private Transform[] Eyes;
    [SerializeField] private Light[] lights;
    public Color colour { get; private set; }

    public RoboEyes(RoboVision _roboVision)
    {
        Eyes = _roboVision.Eyes;
        lights = _roboVision.GetComponentsInChildren<Light>();
        Eyes[0].GetChild(0).GetComponent<MeshRenderer>().sharedMaterial = new
Material(Eyes[0].GetChild(0).GetComponent<MeshRenderer>().sharedMaterial);
        Eyes[1].GetChild(0).GetComponent<MeshRenderer>().sharedMaterial =
Eyes[0].GetChild(0).GetComponent<MeshRenderer>().sharedMaterial;
    }
    public RoboEyes(AndroidVision _roboVision)
    {
        Eyes = _roboVision.Eyes;
        lights = _roboVision.GetComponentsInChildren<Light>();
        Eyes[0].GetChild(0).GetComponent<MeshRenderer>().sharedMaterial = new
Material(Eyes[0].GetChild(0).GetComponent<MeshRenderer>().sharedMaterial);
        Eyes[1].GetChild(0).GetComponent<MeshRenderer>().sharedMaterial =
Eyes[0].GetChild(0).GetComponent<MeshRenderer>().sharedMaterial;
    }

    public void SetEyeColour(Color eyeColour)
    {
        colour = eyeColour;

        Eyes[0].GetChild(0).GetComponent<MeshRenderer>().sharedMaterial.SetColor("_Color",
eyeColour);

        Eyes[1].GetChild(0).GetComponent<MeshRenderer>().sharedMaterial.SetColor("_Color",
eyeColour);
        foreach (Light light in lights)
            light.color = eyeColour;
    }
}

```

RobotWeight.cs

```

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

public class RobotWeight : ZeroWeight
{
    [SerializeField] private Transform Chest;

    [SerializeField] private float walkSpeed = 0.1f;
    [SerializeField] private float thrustSpeed = 0.1f;
    [SerializeField] private float defaultPowerSeconds = 0.2f;
    [Range(0, 5)]
    [SerializeField] private float jitterFix = 0;

    private float powerTimer = 0;

    public float forceMeter { get; set; }
    private float prevSqrSpeed;

    [SerializeField] private ShipWeight Ship;

    [SerializeField] private AudioSource walkSource;

    public void UpdateForceMeter()
    {
        float sqrSpeed = position.sqrMagnitude;
        forceMeter = Mathf.Abs(sqrSpeed - prevSqrSpeed);
        prevSqrSpeed = sqrSpeed;
    }

    public override Vector3 MoveRelative(Vector3 displacement)
    {
        walkSource.volume *= 0.9f;

        bool onPlanet = sigWeight != null;
        Vector3 normalUp = CameraState.inShip ? Ship.transform.up : onPlanet ? (position - sigWeight.position).normalized : transform.up;

        Quaternion target =
        Quaternion.LookRotation(Vector3.ProjectOnPlane(transform.forward, normalUp), normalUp);
        rb.MoveRotation(Quaternion.RotateTowards(rb.rotation, target, 5));

        if (!CameraState.InLockState(CameraState.LockState.unlocked))
            return Vector3.zero;

        Vector3 moveBy = Vector3.zero;

        if (kinematicBody.isGrounded && onPlanet || CameraState.inShip)
        {
            moveBy = Vector3.ProjectOnPlane(Camera.main.transform.forward, Chest.up) *
            Input.GetAxis("Vertical") + Vector3.ProjectOnPlane(Camera.main.transform.right, Chest.up) *
            Input.GetAxis("Horizontal");

            if (moveBy.sqrMagnitude > 1)
                moveBy = moveBy.normalized;
        }

        walkSource.volume = 0.02f * moveBy.sqrMagnitude;
    }
}

```

```

        moveBy *= walkSpeed;
    }

    if (CameraState.inShip)
        displacement = -0.1f * normalUp;

    if (Input.GetButton("Jump"))
    {
        //Only allow the player to apply a jump force for defaultPowerSeconds
        if (powerTimer < defaultPowerSeconds)
        {
            //Maintain some lateral velocity during jump
            Vector3 lateral =
Vector3.ProjectOnPlane(Camera.main.transform.forward, Chest.up) *
Input.GetAxis("Vertical") + Vector3.ProjectOnPlane(Camera.main.transform.right,
Chest.up) * Input.GetAxis("Horizontal");
            velocity += thrustSpeed * ((defaultPowerSeconds - powerTimer) /
(Time.fixedDeltaTime * defaultPowerSeconds) * (normalUp + lateral.normalized * walkSpeed));
            powerTimer += Time.fixedDeltaTime;
        }
    }
    else if (kinematicBody.isGrounded)
    {
        powerTimer = Mathf.Max(0, powerTimer - Time.fixedDeltaTime);
        Vector3 intoGround = Vector3.Project(displacement + moveBy, normalUp);
        //Remove requested displacement into ground and apply smaller constant
displacement down instead to fix the player jittering up and down
        if (intoGround.sqrMagnitude < jitterFix * jitterFix)
            moveBy += -normalUp * jitterFix - intoGround;
    }
}

displacement += moveBy;

return base.MoveRelative(displacement);
}

//Rotates the player the same way they would rotate were they a child of the
parent that performed the rotation at their position
public void RotateAsChild(Vector3 parentPosition, Quaternion rotation)
{
    if (CameraState.inShip)
    {
        Vector3 normalUp = Ship.transform.up;

        Quaternion target =
Quaternion.LookRotation(Vector3.ProjectOnPlane(transform.forward, normalUp),
normalUp);
        rb.MoveRotation(Quaternion.RotateTowards(rb.rotation, target, 5));

        Vector3 direction = position - parentPosition;
        Teleport(rotation * direction - direction);
    }
}
}

```

RoboVision.cs

```

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

public class RoboVision : MonoBehaviour
{

    //Structure which assigns a score to a target requested to be highlighted
    public struct TargetBounds
    {
        public int ID;
        public Transform[] bounds;
        public float golfScore;
        public float timeTargeted;

        public TargetBounds(Transform[] bounds)
        {
            ID = bounds[0].GetInstanceID();
            this.bounds = bounds;

            Vector3 toBounds = bounds[0].position - Camera.main.transform.position;
            float distance = toBounds.magnitude;
            bool valid = Vector3.Dot(toBounds / distance, Camera.main.transform.forward)
                > 0.5 && distance > 10;

            golfScore = valid ? distance : float.MaxValue;
            timeTargeted = Time.realtimeSinceStartup;
        }

        public TargetBounds(Transform planetBounds)
        {
            bounds = new Transform[planetBounds.childCount + 1];
            bounds[0] = planetBounds;
            ID = bounds[0].GetInstanceID();

            for (int i = 0; i < planetBounds.childCount; i++)
                bounds[i + 1] = planetBounds.GetChild(i);

            golfScore = float.MaxValue;
            timeTargeted = Time.realtimeSinceStartup;
        }

        public TargetBounds(float golfScoreToBeat)
        {
            ID = -1;
            bounds = null;
            golfScore = golfScoreToBeat;
            timeTargeted = Application.isPlaying ? Time.realtimeSinceStartup :
                float.MaxValue;
        }
    }

    public static TargetBounds highlightBounds;
    private IEnumerator updateScreenEffects;

    [SerializeField] private Material screenEffectsMat;

    [SerializeField] private Color eyeColour = Color.cyan;
    [SerializeField] private Color dangerColour = Color.red;
}

```

```
public static Color visionColour;

[SerializeField] private Slider fuel;
[SerializeField] private Slider oxygen;
[Range(0, 1)]
public float blur01 = 1;
private float danger01 = -1;

public CameraState.LockState isLocked { get; private set; }
private bool locking;
private Vector3 ogLocalPos;

private Vector3 startPos;
private Vector3 endPos;
private Quaternion startRot;
private Quaternion endRot;
private Transform endTransform;
private float lerp = 1;
public float lockSpeed = 2;

[SerializeField] private RobotWeight robotWeight;
[SerializeField] private Transform Body;
[SerializeField] private Transform Arms;
[SerializeField] private Transform Chest;
[SerializeField] private Transform Head;

public Transform[] Eyes;

[SerializeField] private float lookSpeed = 2;
[SerializeField] private float inertia = 100;
[SerializeField] private float lookHoldTime = 0.5f;

[SerializeField] private Vector2 minMaxLookClamps;
[SerializeField] private float maxHeadRot;
[SerializeField] private float maxEyeRot;
private float bodyRot = 0;

private float xRotation = 0;
private float yRotation = 0;

private float nearClipPlaneLerp;
private float cameraIsPausedLerp = 1.1f;
public int cameraIsPausedLerpDirection { get; private set; }

private float deadBlurLerp = 2;

private float yRotStillCount = 0;

private bool prevOutOfRange = false;
private bool prevGrounded = false;
private bool justLanded = false;

private void Start()
{
    CameraState.isPaused = false;
    Cursor.visible = false;
    Cursor.lockState = CursorLockMode.Locked;

    highlightBounds = new TargetBounds(float.MaxValue);

    isLocked = CameraState.LockState.unlocked;
```

```

        ogLocalPos = transform.localPosition;
        endPos = transform.parent.position + ogLocalPos;

        updateScreenEffects = UpdateScreenEffects(0.5f);
        StartCoroutine(updateScreenEffects);
    }

    private void OnValidate()
    {
        if (screenEffectsMat != null && !Application.isPlaying)
            screenEffectsMat.SetFloat("blur01", 0);
    }

    private void Update()
    {
        //Kill player if they go into the ocean
        CameraState.isDead |= robotWeight.sigWeight != null &&
        (Camera.main.transform.position - robotWeight.sigWeight.position).sqrMagnitude <
        (robotWeight.sigWeight.planet.planetValues.radius - 10) *
        (robotWeight.sigWeight.planet.planetValues.radius - 10);
        //Kill player if they go into empty space
        CameraState.isDead |= robotWeight.sigWeight == null && !CameraState.inShip;

        nearClipPlaneLerp = Mathf.Clamp01(nearClipPlaneLerp + Time.deltaTime *
        (CameraState.flyingShip ? 1 : -1));
        Camera.main.nearClipPlane = Mathf.Lerp(0.4f, 1.5f, nearClipPlaneLerp);
        fuel.GetComponentInParent<Canvas>().planeDistance = Camera.main.nearClipPlane
        + 0.01f;

        EnableEyeRays(!CameraState.inShip || InventoryUI.shipEngineOn01 <= 0,
        !CameraState.inShip);

        deadBlurLerp += Time.smoothDeltaTime;
        //Pixelate screen to black if dead
        if (deadBlurLerp < 2)
        {
            PixelateBlack(Mathf.Pow(Mathf.Max(deadBlurLerp, 1) - 1, 4));
            screenEffectsMat.SetFloat("temperature01", CameraState.inShip ||
        CameraState.inHive ? 0.5f : InventoryUI.robotTemperature);
        }
        else if (!CameraState.isPaused)
        {
            //Pixelate screen based on forceMeter

            float blur01 = robotWeight.forceMeter;

            if (CameraState.flyingShip && robotWeight.sigWeight == null)
                blur01 /= 1000;
            else if (CameraState.flyingShip)
                blur01 /= 10;
            else
                blur01 /= 0.5f;

            blur01 = Mathf.Floor(blur01 * 20) / 20;

            screenEffectsMat.SetFloat("blur01", Mathf.Clamp(1 - blur01, 0.05f, 1));
            screenEffectsMat.SetFloat("temperature01", CameraState.inShip ||
        CameraState.inHive ? 0.5f : InventoryUI.robotTemperature);
        }
    }

    private IEnumerator UpdateScreenEffects(float time)

```

```

{
    while (true)
    {
        yield return new WaitForSeconds(time);

        if (fuel == null || oxygen == null)
            continue;

        bool nl = highlightBounds.bounds == null;
        screenEffectsMat.SetVector("target", nl ? Vector3.zero :
highlightBounds.bounds[0].position);
        screenEffectsMat.SetVector("b1", nl ? Vector3.zero :
highlightBounds.bounds[1].position);
        screenEffectsMat.SetVector("b2", nl ? Vector3.zero :
highlightBounds.bounds[2].position);
        screenEffectsMat.SetVector("b3", nl ? Vector3.zero :
highlightBounds.bounds[3].position);
        screenEffectsMat.SetVector("b4", nl ? Vector3.zero :
highlightBounds.bounds[4].position);
        screenEffectsMat.SetVector("b5", nl ? Vector3.zero :
highlightBounds.bounds[5].position);
        screenEffectsMat.SetVector("b6", nl ? Vector3.zero :
highlightBounds.bounds[6].position);
        screenEffectsMat.SetVector("b7", nl ? Vector3.zero :
highlightBounds.bounds[7].position);
        screenEffectsMat.SetVector("b8", nl ? Vector3.zero :
highlightBounds.bounds[8].position);

        if (Time.realtimeSinceStartup > highlightBounds.timeTargeted + 5)
            highlightBounds = new TargetBounds(float.MaxValue);

        float new_danger01 = 1 - fuel.value / fuel.MaxValue;
        if (danger01 == new_danger01)
            continue;

        danger01 = new_danger01;

        visionColour = Color.Lerp(eyeColour, dangerColour, danger01);

        screenEffectsMat.SetColor("linesCol", visionColour);
        screenEffectsMat.SetFloat("danger01", danger01);

        foreach (Light light in transform.GetComponentsInChildren<Light>())
            light.color = visionColour;
    }

}

//Fix camera position and rotation
public void Lock(Transform cameraPos)
{
    locking = true;
    lerp = 0;

    startPos = transform.position;
    startRot = transform.rotation;

    endTransform = cameraPos;
    endPos = Vector3.zero;

    EnableEyeRays(false, false);
}

```

```

//Allow the player to rotate the camera again
public void Unlock()
{
    locking = false;
    lerp = 0;

    startPos = transform.position;
    endPos = transform.parent.position + ogLocalPos;
    startRot = transform.rotation;
    endRot = Body.rotation * Quaternion.Euler(-xRotation, yRotation, 0);

    EnableEyeRays(true, true);
}

private void EnableEyeRays(bool eyesOnOff, bool bloomerOnOff)
{
    transform.GetChild(0).GetComponent<Light>().enabled = eyesOnOff;
    transform.GetChild(1).GetComponent<Light>().enabled = eyesOnOff;
    transform.GetChild(2).GetComponent<Light>().enabled = bloomerOnOff;
}

public void PixelateBlack(float x)
{
    screenEffectsMat.SetFloat("blur01", x);
    screenEffectsMat.SetFloat("dark01", x);
}
public void KillTransition()
{
    PixelateBlack(0);
    deadBlurLerp = 0;
}

void LateUpdate()
{
    if (Input.GetKeyDown(KeyCode.Escape) || PauseMenu.selectedPauseMenuOption == 0
    || PauseMenu.selectedPauseMenuOption == 1 || PauseMenu.selectedPauseMenuOption == 2 ||
    PauseMenu.selectedPauseMenuOption == 3)
    {
        Cursor.visible ^= true;
        Cursor.lockState = Cursor.lockState == CursorLockMode.Locked ?
    CursorLockMode.None : CursorLockMode.Locked;

        if (CameraState.isPaused)
        {
            cameraIsPausedLerp = Mathf.Max(0, cameraIsPausedLerp);
            cameraIsPausedLerpDirection = 1;
        }
        else
        {
            CameraState.isPaused = true;
            cameraIsPausedLerp = Mathf.Min(1, cameraIsPausedLerp);
            cameraIsPausedLerpDirection = -1;
        }
    }

    //If the camera is paused or pausing, return after updating
    //cameraIsPausedLerp, apply pixilation transition to pause menu if the death transition
    //is not in effect
    if (cameraIsPausedLerp >= 0 && cameraIsPausedLerp <= 1)
    {
        cameraIsPausedLerp += 2 * Time.deltaTime * cameraIsPausedLerpDirection;
    }
}

```

```

        if (deadBlurLerp > 2)
            PixalateBlack(Mathf.Pow(cameraIsPausedLerp, 4));
        return;
    }
    else if (CameraState.isPaused)
    {
        CameraState.isPaused = cameraIsPausedLerpDirection == -1;
        if (deadBlurLerp > 2)
            PixalateBlack(Mathf.Clamp01(cameraIsPausedLerp));
        return;
    }
    if (deadBlurLerp > 2)
    {
        screenEffectsMat.SetFloat("dark01", 1);
    }

    //If the camera is not locked, set the target position to the original camera
    position
    if (!locking)
        endPos = transform.parent.position + ogLocalPos;

    isLocked = locking ? CameraState.LockState.locked :
    CameraState.LockState.unlocked;

    //If the camera has finished lerping, set the camera's position and rotation
    to the final values
    if (lerp >= 1)
    {
        transform.position = endPos == Vector3.zero ? endTransform.position :
    endPos;
        transform.rotation = endPos == Vector3.zero ? endTransform.rotation :
    endRot;
    }
    //If the camera is still lerping, set the camera's position and rotation to
    the interpolated values based on the current lerp value
    if (lerp < 1)
    {
        isLocked = CameraState.LockState.changing;
        transform.position = Vector3.Slerp(startPos, endPos == Vector3.zero ?
    endTransform.position : endPos, lerp);
        transform.rotation = Quaternion.Slerp(startRot, endPos == Vector3.zero ?
    endTransform.rotation : endRot, lerp);
        lerp += lockSpeed * Time.smoothDeltaTime;
    }
    //If the camera is not locked and the lerp is finished, handle camera rotation
    based on user input
    else if (!locking)
    {
        justLanded |= !prevGrounded && robotWeight.kinematicBody.isGrounded;

        if (justLanded && xRotation < minMaxLookClamps.x)
            xRotation = Mathf.MoveTowards(xRotation, minMaxLookClamps.x,
    Time.deltaTime * 1000);
        else if (justLanded && xRotation > minMaxLookClamps.y)
            xRotation = Mathf.MoveTowards(xRotation, minMaxLookClamps.y,
    Time.deltaTime * 1000);
        else
        {
            xRotation += Input.GetAxis("Mouse Y") * lookSpeed;
            justLanded = false;
        }
    }
}

```

SadGirl.cs

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

public class SadGirl : MonoBehaviour
{
    private enum State { readyToSpeak, waitingToVanish, vanished }

    [SerializeField] private State girlState;
    private bool interacting = false;
    [SerializeField] private Material shipLightMap;
    [SerializeField] private ShipWeight shipWeight;

    private TextRender textRender;
    Vector3 ogLocalAngles;
    private float spokenTime;
    private float spokenGapMinutes = 5;

    private float prevGenTime;

    public static float faceLight01;

    private int conversationCount;

    [System.Serializable]
    private struct ConversationWrap
    {
        public int conversationCount;

        public ConversationWrap(int conversationCount)
        {
            this.conversationCount = conversationCount;
        }
    }

    public void Wipe()
    {
        conversationCount = 0;
        JsonSaver.SaveData("Eva_Conversation", new ConversationWrap(0));
    }

    // Start is called before the first frame update
    void Start()
    {
        conversationCount = JsonSaver.LoadData<ConversationWrap>("Eva_Conversation", out
bool success).conversationCount;
        if (!success)
        {
            conversationCount = 0;
            JsonSaver.SaveData("Eva_Conversation", new
ConversationWrap(conversationCount));
        }

        ogLocalAngles = transform.GetChild(1).localEulerAngles;
        textRender = transform.GetChild(2).GetComponent<TextRender>();
    }
}
```

```

// Update is called once per frame
void Update()
{
    if (girlState == State.waitingToVanish && (!CameraState.inShip || CameraState.flyingShip))
        girlState = State.vanished;
    else if (girlState == State.vanished && Time.realtimeSinceStartup > spokenTime + spokenGapMinutes * 60 && prevGenTime < StarGenSystem.genTime)
        girlState = State.readyToSpeak;

    //Set the opaque or transparent skin based on whether Eva has vanished
    transform.GetChild(1).gameObject.SetActive(girlState != State.vanished);
    transform.GetChild(3).gameObject.SetActive(girlState == State.vanished);

    if (CameraState.InLockState(CameraState.LockState.unlocked))
        interacting = false;

    if (!interacting && CameraState.CamIsInteractingW(transform.position, transform.GetChild(0).forward, 3, 60))
    {
        interacting = true;
        CameraState.LockCamera(transform.GetChild(0));
        LoadConversation();
    }
    else if (interacting && CameraState.InLockState(CameraState.LockState.locked) && Input.GetKeyDown(KeyCode.E) && !textRender.NextSentence())
    {
        interacting = false;
        CameraState.UnlockCamera();

        JsonSaver.SaveData("Eva_Conversation", new ConversationWrap(conversationCount));
    }
    Material skin =
    transform.GetChild(1).GetComponent<MeshRenderer>().sharedMaterial;
    skin.SetFloat("_light", Mathf.Lerp(skin.GetFloat("_light"),
    shipLightMap.GetFloat("_brightness") * Mathf.Lerp(0.75f, 1, faceLight01),
    Time.deltaTime));

    //Teeter forward and back for realism (nobody sits completely still)
    transform.GetChild(1).localEulerAngles = ogLocalAngles + 1 *
    Mathf.Sin(Time.realtimeSinceStartup) * Vector3.right;
    transform.GetChild(3).localEulerAngles =
    transform.GetChild(1).localEulerAngles;
}

void LoadConversation()
{
    if (girlState != State.readyToSpeak)
    {
        textRender.LoadConversation("eva/blank");
        return;
    }
    textRender.LoadConversation("eva/" + (conversationCount + 1).ToString());

    spokenTime = Time.realtimeSinceStartup;
    conversationCount++;

    girlState = State.waitingToVanish;

    prevGenTime = StarGenSystem.genTime;
}
}

```

Save.cs

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

public class JsonSaver
{
    public static void SaveData(string filename, object data)
    {
        File.WriteAllText(PathOf(filename), JsonUtility.ToJson(data));
        Debug.Log("Saved: " + PathOf(filename));
    }

    public static Type LoadData<T>(string filename, out bool success)
    {
        success = File.Exists(PathOf(filename));
        return success ? JsonUtility.FromJson<T>(File.ReadAllText(PathOf(filename))) :
default;
    }

    private static string PathOf(string filename)
    {
        return Application.persistentDataPath + "/" + filename + ".json";
    }
}
```

ScreenEffects.shader

```

Shader "Custom/ScreenEffects"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
    }
    SubShader
    {
        // No culling or depth
        Cull Off ZWrite Off ZTest Always

        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"
            #include "Math.cginc"
            #include "PerlinNoise.cginc"
            //

            struct appdata {
                float4 vertex : POSITION;
                float4 uv : TEXCOORD0;
            };

            struct v2f {
                float4 pos : SV_POSITION;

                float2 uv : TEXCOORD0;
                float3 viewVector : TEXCOORD1;
            };

            v2f vert (appdata v) {
                v2f output;
                output.pos = UnityObjectToClipPos(v.vertex);

                output.uv = v.uv;
                // Camera space matches OpenGL convention where cam
forward is -z. In unity forward is positive z.
                // (https://docs.unity3d.com/ScriptReference/Camera-cameraToWorldMatrix.html)
                float3 viewVector = mul(unity_CameraInvProjection,
float4(v.uv.xy * 2 - 1, 0, -1));
                output.viewVector = mul(unity_CameraToWorld,
float4(viewVector,0));
            }

            f32 frag (v2f i) : SV_Target
            {
                float4 col = tex2D(_MainTex, i.uv);
                return col;
            }
        }
    }
}

```

```

        return output;
    }

    float2 squareUV(float2 uv) {
        float width = _ScreenParams.x;
        float height = _ScreenParams.y;
        //float minDim = min(width, height);
        float scale = 1000;
        float x = uv.x * width;
        float y = uv.y * height;
        return float2 (x/scale, y/scale);
    }

    sampler2D _MainTex;

    float danger01;
    float blur01;
    float dark01;
    float temperature01; //hot = 1, i wish this was the convention
throughout my code but alas...
    float4 linesCol;

    float2 group;
    float3 forward;

    float3 target;
    float3 b1;
    float3 b2;
    float3 b3;
    float3 b4;
    float3 b5;
    float3 b6;
    float3 b7;
    float3 b8;

    //Convert world pos to uv
    float2 w(float3 pos)
    {
        pos = normalize(pos - _WorldSpaceCameraPos) *
(_ProjectionParams.y + _ProjectionParams.z - _ProjectionParams.y) +
_WorldSpaceCameraPos;
        fixed2 uv = 0;
        fixed3 toCam = mul(unity_WorldToCamera, pos);
        fixed camPosZ = toCam.z;
        fixed height = 2 * camPosZ / unity_CameraProjection._m11;
        fixed width = _ScreenParams.x / _ScreenParams.y * height;
        uv.x = (toCam.x + 0.5 * width) / width;
        uv.y = (toCam.y + 0.5 * height) / height;
    }
}

```

```

        return (float2)uv;
    }

    //Find the minimum and maximum X and Y UV values for each of the
bounds of the target to highlight
    float4 DetectionXYMinMax()
    {
        float minX = min(min(min(min(min(w(b1).x, w(b2).x),
w(b3).x), w(b4).x), w(b5).x), w(b6).x), w(b7).x), w(b8).x);
        float maxX = max(max(max(max(max(max(w(b1).x, w(b2).x),
w(b3).x), w(b4).x), w(b5).x), w(b6).x), w(b7).x), w(b8).x);
        float minY = min(min(min(min(min(w(b1).y, w(b2).y),
w(b3).y), w(b4).y), w(b5).y), w(b6).y), w(b7).y), w(b8).y);
        float maxY = max(max(max(max(max(max(w(b1).y, w(b2).y),
w(b3).y), w(b4).y), w(b5).y), w(b6).y), w(b7).y), w(b8).y);
        return float4(minX, maxX, minY, maxY);
    }

    float4 DrawDetection(float4 originalCol, float4 squareCol, float2
uv)
{
    float4 uvBounds = DetectionXYMinMax();

    //Pixelate bounds
    uvBounds.xz = floor(uvBounds.xz * group) / group;
    uvBounds.yw = floor(uvBounds.yw * group) / group;

    if (dot(normalize(target - _WorldSpaceCameraPos), forward) <
0)
        return originalCol;

    if ((uv.x == uvBounds.x || uv.x == uvBounds.y) && uv.y >=
uvBounds.z && uv.y <= uvBounds.w ||
        (uv.y == uvBounds.z || uv.y == uvBounds.w) && uv.x >=
uvBounds.x && uv.x <= uvBounds.y)
        return lerp(originalCol, squareCol, 0.1);

    return originalCol;
}

float nrnd(float2 uv)
{
    return frac(sin(dot(uv, float2(12.9898, 78.233))) *
43758.5453);
}

float4 frag (v2f i) : SV_Target
{

```

```

        forward = mul((float3x3)unity_CameraToWorld, float3(0,0,1));
        blur01 = pow(blur01, 0.5);

        float2 temperatureUV;
        //1 is hot, 0 is cold, offset UV based to evoke temperature
        (heat waves & shaking for hot & cold)
        if (temperature01 > 0.5)
            temperatureUV = i.uv + 0.01 * pow(clamp(remap01(0.5, 1,
        temperature01), 0, 1), 3) * float2(PerlinNoise3D(float3(3 * i.uv, _Time.w)),
        0.2 * PerlinNoise3D(float3(3 * i.uv, -1 - _Time.w)));
        else
            temperatureUV = i.uv + 0.0005 * pow(clamp(remap01(0.5, 0,
        temperature01), 0, 1), 3) * float2(floor(_Time.w * unity_DeltaTime.w) % 2 * 2
        - 1, 0.2 * (floor(_Time.w * unity_DeltaTime.w) % 2 * 2 - 1));

        temperatureUV = float2(clamp(temperatureUV.x, 0, 1),
        clamp(temperatureUV.y, 0, 1));
        //Sample original color from texture and apply pixelating and
        darkening effect
        float4 originalCol = tex2D(_MainTex, floor(temperatureUV *
        _ScreenParams * blur01) / (_ScreenParams * blur01)) * pow(dark01, 0.15);

        int pixelWidth = 6;
        group = _ScreenParams / pixelWidth * blur01;
        float2 modulatedUV = floor(i.uv * group) / group;

        float length01 = remap01(lerp(0.55, 0.75, 1 - danger01), 1,
        length(modulatedUV - float2(0.5, 0.5)) * 1.41);

        pixelWidth = floor(lerp(pixelWidth * 0.5, 6, length01) * 2) *
        0.5;

        float2 localGroup = _ScreenParams / pixelWidth * blur01;
        float2 localModulatedUV = floor(temperatureUV * localGroup) /
        localGroup;

        float2 seed = 10 * localModulatedUV;

        float pixelNoise1 = nrnd(seed * forward.xy);
        float pixelNoise2 = nrnd(seed * forward.xz);
        float pixelNoise3 = nrnd(seed * forward.yz);
        float4 pixelNoise = float4(pixelNoise1, pixelNoise2,
        pixelNoise3, 1) * linesCol;

        float border01 = clamp(length(localModulatedUV - float2(0.5,
        0.5)) * 1.41 - 0.1 * pixelNoise1 - 0.25, 0, 1);
    
```

```

        float borderLerp = 1 - clamp(danger01 + 0.2 * sin(_Time.w) -
0.2, 0, 1);

        originalCol = DrawDetection(originalCol, pixelNoise,
modulatedUV);

        //Close to centre of screen return originalCol (with blur
(pixelate) and dark and temperature effects applied)
        if (border01 < lerp(0.25, 0.45, borderLerp))
            return originalCol;

        float4 modulatedCol = tex2D(_MainTex, localModulatedUV);

        //Further out return modulated (even more pixalated)
originalCol
        if (border01 < lerp(0.35, 0.45, borderLerp) + 0.05)
            return lerp(originalCol, pixelNoise, 0.01 * border01);

        //Apply moving grid lines to screen border

        float4 newCol = lerp(modulatedCol, pixelNoise, 0.01 *
border01);

        uint2 pixel = (uint2)floor(i.uv * group * blur01);
        uint n = (uint)lerp(15, 20, danger01);
        uint timeMod = (uint)(_Time.x * lerp(100, 200, danger01) +
pixelNoise1) % n;
        uint xTimeMod = timeMod;
        uint yTimeMod = timeMod;
        if (i.uv.x < 0.5)
            xTimeMod = n - timeMod - 1;
        if (i.uv.y < 0.5)
            yTimeMod = n - timeMod - 1;

        if (pixel.x % n == xTimeMod || pixel.y % n == yTimeMod)
            newCol = lerp(newCol, linesCol, 0.01);

        return newCol * pow(lerp(1 - length01, 1, 1 - danger01), 1);
    }

    ENDCG
}
}
}

```

ScreenEffectsHandler.cs

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

[CreateAssetMenu(menuName = "PostProcessing/ScreenEffectsHandler")]
public class ScreenEffectsHandler : EffectsHandler
{
    public Material screenEffects;
    public bool enabled;
    public bool debug;

    public override List<Material> GetMaterials()
    {
        if (Application.isPlaying && enabled || debug)
            return new List<Material>(1) { screenEffects };

        return new List<Material>();
    }
}
```

Segment.cs

```

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

public class Segment : MonoBehaviour
{
    public Planet planet;
    public Mesh mesh { get; private set; }
    public float camDist { get; private set; }
    public int SegIndex;
    private bool natureGenerated;
    private bool requestTower;
    private bool generatedTower = false;

    private int natureCount;
    private int treeCountPerSeg;
    private int oreCountPerSeg;
    private int grassCountPerSeg;
    private int stoneCountPerSeg;

    public static SunGenSystem sun;
    public static Stack<GameObject> instantiatedGrass;
    public static Stack<GameObject> instantiatedStone;
    private Dictionary<int, GameObject> segmentGrass = new Dictionary<int, GameObject>();
    private Dictionary<int, GameObject> segmentStone = new Dictionary<int, GameObject>();
    private bool addedFoilage = false;

    private MeshFilter meshFilter;
    private MeshRenderer meshRenderer;
    private MeshCollider meshCollider;

    public int currentRes;

    private System.Random nPrng;

    public void SetMesh(Mesh _mesh)
    {
        mesh = _mesh;
        mesh.RecalculateNormals();
        mesh.RecalculateBounds();
        //mesh.Optimize(); //Omitting saves lots of time when creating mesh, but mesh
        is supposedly then less optimised for rendering
    }

    public void UpdateMeshFilter()
    {
        if (meshFilter == null)
            GetComponents();
        meshFilter.sharedMesh = mesh;
    }

    private void UpdateMeshCollider()
    {
        if (meshCollider == null)
            GetComponents();
        meshCollider.sharedMesh = mesh;
    }

    public void SetRenderMaterial(Material _material)
    {
        if (meshRenderer == null)
    }
}

```

```

        GetComponents();
        meshRenderer.sharedMaterial = _material;
    }
    public void SetRendererVisible(bool state)
    {
        if (meshRenderer == null)
            GetComponents();
        meshRenderer.enabled = state;
    }
    private void SetColliderActive(bool state)
    {
        if (meshCollider == null)
            GetComponents();
        meshCollider.enabled = state;
    }
    private void GetComponents()
    {
        meshFilter = GetComponent<MeshFilter>();
        meshRenderer = GetComponent<MeshRenderer>();
        meshCollider = GetComponent<MeshCollider>();
    }

    private bool ShowNature()
    {
        return IsCameraClose(); // && !CameraState.flyingShip
    }
    private bool IsCameraClose()
    {
        camDist = mesh.bounds.SqrDistance(Camera.main.transform.position -
transform.position);
        return camDist < planet.maxNatureDist * planet.maxNatureDist;
    }
    private void AddNature()
    {
        float sqrRadius = planet.planetValues.radius * planet.planetValues.radius *
0.98f;

        //Want a mesh vertex to add nature on that is not at a steep gradient and is
above the ocean
        int index;
        float surfaceAngle;
        int error = 0;
        do
        {
            index = nPrng.Next(0, mesh.vertexCount);
            surfaceAngle = Vector3.Dot(mesh.normals[index],
mesh.vertices[index].normalized);
            error++;
        } while (error < 100 && (surfaceAngle < 0.35f ||
mesh.vertices[index].sqrMagnitude < sqrRadius));

        if (natureCount < treeCountPerSeg)
        {
            GameObject _treeObj = Instantiate(planet.Tree, transform.position +
mesh.vertices[index], transform.rotation *
Quaternion.LookRotation(mesh.normals[index]), transform.GetChild(0));

            _treeObj.layer = gameObject.layer;

            TreeGen _tree = _treeObj.GetComponent<TreeGen>();
            _tree.planetNormal = mesh.normals[index];
        }
    }
}

```

```

        _tree.SetMaterials(planet, mesh.vertices[index]);
        _tree.Init(gameObject.layer, nPrng.Next(-9999, 9999));
    }
    else if (natureCount < treeCountPerSeg + oreCountPerSeg)
    {

        GameObject _oreObj = Instantiate(planet.Ore, transform.position +
mesh.vertices[index], transform.rotation *
Quaternion.LookRotation(mesh.normals[index]), transform.GetChild(0));

        _oreObj.layer = gameObject.layer;

        OreGen _ore = _oreObj.GetComponent<OreGen>();

        _ore.SetMaterials(planet, mesh.vertices[index]);
        _ore.Init(gameObject.layer, nPrng.Next(-9999, 9999));
    }

    natureCount++;
}

private void AddFoilage()
{
    if (addedFoilage)
        return;

    //Need to do this each time instead of using the nPrng since foilage can be
    re-added
    Random.InitState(planet.planetValues.environmentSeed + SegIndex);

    for (int i = 0; i < grassCountPerSeg; i++)
    {
        if (instantiatedGrass.Count == 0)
            break;
        GameObject grass = instantiatedGrass.Pop();
        int index;
        if (segmentGrass.Count < grassCountPerSeg)
        {
            do index = Random.Range(0, mesh.vertices.Length);
            while (segmentGrass.ContainsKey(index));
            segmentGrass.Add(index, grass);
        }
        else
        {
            int[] indices = new int[segmentGrass.Keys.Count];
            segmentGrass.Keys.CopyTo(indices, 0);
            index = indices[i];
            segmentGrass[index] = grass;
        }
    }

    grass.SetActive(Vector3.Dot(mesh.vertices[index], mesh.normals[index]) >
0.7f);

    grass.transform.parent = transform.GetChild(0);
    grass.transform.position = transform.position + mesh.vertices[index];
    grass.transform.rotation = transform.rotation *
Quaternion.LookRotation(mesh.normals[index]);
    grass.transform.localScale = new Vector3(2, 2, Random.Range(1f, 2));

    Material grassMat =
grass.transform.GetChild(0).GetChild(0).GetComponent<MeshRenderer>().sharedMaterial;
    grassMat.SetFloat("_minElevation", planet.planetMesh.elevationData.Min);
}

```

```

        grassMat.SetFloat("_maxElevation", planet.planetMesh.elevationData.Max);
        grassMat.SetTexture("_planetTexture",
planet.terrainMat.GetTexture("_planetTexture"));
        grassMat.SetVector("_position", grass.transform.position -
planet.transform.position);
    }

    for (int i = 0; i < stoneCountPerSeg; i++)
    {
        if (instantiatedStone.Count == 0)
            break;
        GameObject stone = instantiatedStone.Pop();
        int index;
        if (segmentStone.Count < stoneCountPerSeg)
        {
            do index = Random.Range(0, mesh.vertices.Length);
            while (segmentStone.ContainsKey(index));
            segmentStone.Add(index, stone);
        }
        else
        {
            int[] indices = new int[segmentStone.Keys.Count];
            segmentStone.Keys.CopyTo(indices, 0);
            index = indices[i];
            segmentStone[index] = stone;
        }

        stone.SetActive(Vector3.Dot(mesh.vertices[index], mesh.normals[index]) >
0.7f);
        stone.transform.parent = transform.GetChild(0);
        stone.transform.position = transform.position + mesh.vertices[index];
        stone.transform.rotation = transform.rotation *
Quaternion.LookRotation(mesh.normals[index]);
        stone.transform.localScale = Vector3.one * Random.Range(0.2f, 0.4f);

        for (int j = 0; j < stone.transform.childCount; j++)
            stone.transform.GetChild(j).gameObject.SetActive(Random.value < 0.5f);

        Material stoneMat =
stone.transform.GetChild(0).GetComponent<MeshRenderer>().sharedMaterial;
        stoneMat.SetFloat("_minElevation", planet.planetMesh.elevationData.Min);
        stoneMat.SetFloat("_maxElevation", planet.planetMesh.elevationData.Max);
        stoneMat.SetTexture("_planetTexture",
planet.terrainMat.GetTexture("_planetTexture"));
        stoneMat.SetVector("_position", stone.transform.position -
planet.transform.position);
    }

    addedFoilage = true;
}

public void RemoveFoilage()
{
    if (!addedFoilage)
        return;

    foreach (int key in segmentGrass.Keys)
    {
        segmentGrass[key].SetActive(false);
        segmentGrass[key].transform.parent =
sun.transform.GetChild(0).GetChild(0);
        instantiatedGrass.Push(segmentGrass[key]);
    }
}

```

```

        }

        foreach (int key in segmentStone.Keys)
        {
            segmentStone[key].SetActive(false);
            segmentStone[key].transform.parent =
sun.transform.GetChild(0).GetChild(1);
            instantiatedStone.Push(segmentStone[key]);
        }
        segmentGrass.Clear();
        segmentStone.Clear();

        addedFoilage = false;
    }

private bool AddTower()
{
    if (generatedTower)
        return false;

    SetColliderActive(true);

    Vector3 localUp = mesh.vertices[mesh.vertexCount / 2].normalized;
    Vector3 localRight = Vector3.Cross(localUp, Vector3.forward).normalized;
    Vector3 localForward = Vector3.Cross(localUp, localRight).normalized;

    if (!Physics.Raycast(localUp * planet.planetMesh.elevationData.Max +
transform.position, -localUp, out RaycastHit originalHit))
        return true;

    float maxHeight = planet.planetMesh.elevationData.Max - originalHit.distance;

    float square = 12;
    float steps = 3;

    //Check if terrain surrounding selected tower position is flat enough for the
    tower, return false if not
    for (float x = -square; x <= square; x += 2 * square / (steps - 1))
    {
        for (float z = -square; z <= square; z += 2 * square / (steps - 1))
        {
            Vector3 offset = localRight * x + localForward * z;

            Vector3 rayCastPos = localUp * planet.planetMesh.elevationData.Max +
transform.position + offset;

            if (Physics.Raycast(rayCastPos, -localUp, out RaycastHit cornerHit))
            {
                float newHeight = planet.planetMesh.elevationData.Max -
cornerHit.distance;

                if (newHeight > maxHeight)
                    maxHeight = newHeight;

                if (Vector3.Dot(localUp, cornerHit.normal) < 0.7f)
                    return false;
            }
        }
    }
    //Add tower

    Vector3 towerPos = localUp * maxHeight + transform.position;
}

```

```

        GameObject _towerObj = Instantiate(planet.Tower, towerPos,
Quaternion.identity, transform);

        generatedTower = true;

        _towerObj.layer = gameObject.layer;

        TowerGen _tower = _towerObj.GetComponent<TowerGen>();
        _tower.normalUp = localUp;
        _tower.Init(nPrng.Next(-9999, 9999));

        SetColliderActive(false);

        return false;
    }

    public void Start()
{
    nPrng = new System.Random(planet.planetValues.environmentSeed + SegIndex);
    natureGenerated = false;
    natureCount = 0;

    float scaleFactor = Mathf.Pow(planet.planetValues.radius, 2) / 250000;

    float probability = scaleFactor / 144f;
    requestTower = nPrng.NextDouble() < probability;

    treeCountPerSeg = Mathf.RoundToInt(Mathf.Lerp(planet.treeCountPer500Seg.x,
planet.treeCountPer500Seg.y, (float)nPrng.NextDouble() * scaleFactor));
    oreCountPerSeg = Mathf.RoundToInt(Mathf.Lerp(planet.oreCountPer500Seg.x,
planet.oreCountPer500Seg.y, (float)nPrng.NextDouble() * scaleFactor));
    grassCountPerSeg = Mathf.RoundToInt(Mathf.Lerp(planet.grassCountPer500Seg.x,
planet.grassCountPer500Seg.y, (float)nPrng.NextDouble() * scaleFactor));
    stoneCountPerSeg = Mathf.RoundToInt(Mathf.Lerp(planet.stoneCountPer500Seg.x,
planet.stoneCountPer500Seg.y, (float)nPrng.NextDouble() * scaleFactor));

    GameObject natureHolder = new GameObject("Nature Holder (0)");
    natureHolder.layer = gameObject.layer;
    natureHolder.transform.parent = transform;

    SetColliderActive(false);

    UpdateMeshCollider();
}
public void Update()
{
    if (!requestTower)
        SetColliderActive(IsCameraClose());
    if (requestTower)
        requestTower = AddTower();

    //Rather than creating all the new nature objects within 1 frame, it is much
    smoother to create each on a seperate frame
    else if (natureCount > 0 && natureCount < oreCountPerSeg + treeCountPerSeg &&
Time.frameCount % 4 == 0)
        AddNature();
    else if (ShowNature())
    {
        if (natureCount == 0)
        {
            UpdateMeshCollider();

```

```

        AddNature();
    }
    if (!natureGenerated)
    {
        natureGenerated = true;
        transform.GetChild(0).gameObject.SetActive(true);
    }

    AddFoilage();
}
else
{
    if (natureGenerated)
    {
        natureGenerated = false;
        transform.GetChild(0).gameObject.SetActive(false);
    }

    RemoveFoilage();
}

//Generate 500 instances of grass and stone on Start to improve performance
public static void InstantiateFoilage(SunGenSystem system)
{
    sun = system;

    int grassCount = 500;
    int stoneCount = 500;

    instantiatedGrass = new Stack<GameObject>(grassCount);
    for (int i = 0; i < grassCount; i++)
    {
        GameObject grass;
        if (i < sun.transform.GetChild(0).GetChild(0).childCount)
            grass = sun.transform.GetChild(0).GetChild(0).GetChild(i).gameObject;
        else
            grass = Instantiate(sun.Grass, sun.transform.GetChild(0).GetChild(0));

        grass.transform.GetChild(0).GetChild(0).GetComponent<MeshRenderer>().sharedMaterial =
        new
        Material(grass.transform.GetChild(0).GetChild(0).GetComponent<MeshRenderer>().sharedMa
        terial);
            grass.transform.GetChild(0).GetChild(0).gameObject.layer = 6;
            for (int j = 1; j < grass.transform.childCount; j++)
            {

        grass.transform.GetChild(j).GetChild(0).GetComponent<MeshRenderer>().sharedMaterial =
        grass.transform.GetChild(0).GetChild(0).GetComponent<MeshRenderer>().sharedMaterial;
            grass.transform.GetChild(j).GetChild(0).gameObject.layer = 6;
        }
        grass.SetActive(false);

        instantiatedGrass.Push(grass);
    }

    instantiatedStone = new Stack<GameObject>(stoneCount);
    for (int i = 0; i < stoneCount; i++)
    {
        GameObject stone;
        if (i < sun.transform.GetChild(0).GetChild(1).childCount)

```

```
        stone = sun.transform.GetChild(0).GetChild(1).GetChild(i).gameObject;
    else
        stone = Instantiate(sun.Stone, sun.transform.GetChild(0).GetChild(1));

    stone.transform.GetChild(0).GetComponent<MeshRenderer>().sharedMaterial =
new Material(stone.transform.GetChild(0).GetComponent<MeshRenderer>().sharedMaterial);
    stone.transform.GetChild(0).gameObject.layer = 6;
    for (int j = 1; j < stone.transform.childCount; j++)
    {

stone.transform.GetChild(j).GetComponent<MeshRenderer>().sharedMaterial =
stone.transform.GetChild(0).GetComponent<MeshRenderer>().sharedMaterial;
        stone.transform.GetChild(j).gameObject.layer = 6;
    }
    stone.SetActive(false);

    instantiatedStone.Push(stone);
}
}
}
```

Shelf.cs

```

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

public class Shelf : MonoBehaviour
{
    public bool lowest = false;

    public int height = 0;

    // Start is called before the first frame update
    public void Init()
    {
        Transform shelf = transform.GetChild(0);

        int lowIndex = Random.Range(0, shelf.GetChild(height + 1).childCount);

        int topIndex = Random.Range(0, shelf.GetChild(3).childCount);
        int desktopIndex = Random.Range(0, shelf.GetChild(4).childCount);

        if (lowest)
            shelf.GetChild(height + 1).GetChild(lowIndex).gameObject.SetActive(true);
        else
            shelf.GetChild(3).GetChild(topIndex).gameObject.SetActive(true);

        bool biggunOnShelf = desktopIndex == 1 || desktopIndex == 7 || desktopIndex == 8;

        //Bigguns only spawned on top shelf
        if (biggunOnShelf && lowest == true)
        {
            desktopIndex = 0;
            biggunOnShelf = false;
        }

        shelf.GetChild(4).GetChild(desktopIndex).gameObject.SetActive(true);

        //Bigguns include computer and sitting robot, they prevent desktop objects from
being spawned
        if (biggunOnShelf)
            return;

        //For each division on the shelf
        for (int x = -2; x < 2; x++)
        {
            int desktopObjectIndex = Random.Range(0, shelf.GetChild(5).childCount);

            //50% chance of spawning a desktop object
            if (Random.value < 0.5f)
            {

                shelf.GetChild(5).GetChild(desktopObjectIndex).gameObject.SetActive(true);
                Vector3 oldPos =
                shelf.GetChild(5).GetChild(desktopObjectIndex).localPosition;
                shelf.GetChild(5).GetChild(desktopObjectIndex).localPosition = new
                Vector3(x + 0.5f, oldPos.y, oldPos.z);
                shelf.GetChild(5).GetChild(desktopObjectIndex).localRotation =
                Quaternion.Euler(0, Random.value * 90 + 135, 0);
            }
        }
    }
}

```

```
        }  
    }  
}
```

ShipDoorOpen.cs

```

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

public class ShipDoorOpen : MonoBehaviour
{
    private AudioSource liftSource;
    private BoxCollider boxCollider;
    [SerializeField] private FlipSwitch trigger;
    [SerializeField] private float closedLength = 3.514f;
    [SerializeField] private float openLength = 1;
    public float lerp { get; private set; } = 0.5f;
    [SerializeField] private bool initiallyClosed = true;
    [SerializeField] private float lerpSpeed = 1;

    public void Start()
    {
        boxCollider = GetComponent<BoxCollider>();
        liftSource = GetComponent<AudioSource>();
        lerp = initiallyClosed ? -0.1f : 1.1f;
    }

    private void Update()
    {
        //If the switch is in the "top" state and the door is not fully open, increase
        the lerp value
        if (trigger.switchState == FlipSwitch.State.top && lerp < 1)
            lerp += lerpSpeed * Time.deltaTime;
        //If the switch is in the "bottom" state and the door is not fully closed,
        decrease the lerp value
        else if (trigger.switchState == FlipSwitch.State.bottom && lerp > 0)
            lerp -= lerpSpeed * Time.deltaTime;

        //If there's a doorLerpOffset set in the switch component, set the lerp value to
        it and reset the offset, this is used to delay the opening of the door (for example the
        outer door opens after the smoke effect)
        if (trigger.doorLerpOffset != 0)
        {
            lerp = trigger.doorLerpOffset;
            trigger.doorLerpOffset = 0;
        }

        if (!liftSource.isPlaying && lerp < 1 && lerp > 0)
            PlayLift();

        //If the door is fully open, disable the collider component so that the player
        can pass through
        if (lerp > 1 && trigger.switchState != FlipSwitch.State.falling)
        {
            if (boxCollider.enabled == true)
                boxCollider.enabled = false;
        }
        else if (boxCollider.enabled == false)
            boxCollider.enabled = true;

        transform.GetChild(0).localScale = new Vector3(0, 1, 1) + Vector3.right *
        Mathf.Lerp(closedLength, openLength, lerp);
    }
}

```

```
        transform.GetChild(1).localScale = new Vector3(0, 1, 1) - Vector3.right *  
Mathf.Lerp(closedLength, openLength, lerp);  
    }  
  
    void PlayLift()  
    {  
        StartCoroutine(CoPlayLift());  
    }  
IEnumerator CoPlayLift()  
{  
    float startVolume = liftSource.volume;  
  
    liftSource.volume = 0;  
    liftSource.Play();  
  
    while (liftSource.volume < startVolume)  
    {  
        liftSource.volume += startVolume * Time.deltaTime * 5;  
        yield return new WaitForEndOfFrame();  
    }  
  
    yield return new WaitUntil(new System.Func<bool>(() => lerp <= 0 || lerp >=  
1));  
  
    while (liftSource.volume > 0)  
    {  
        liftSource.volume -= startVolume * Time.deltaTime * 5;  
        yield return new WaitForEndOfFrame();  
    }  
  
    liftSource.Stop();  
    liftSource.volume = startVolume;  
}  
}
```

ShipMapAngle.cs

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

public class ShipMapAngle : MonoBehaviour
{
    [SerializeField] private ShipWeight shipWeight;
    private Material starMap;

    // Start is called before the first frame update
    void Start()
    {
        starMap = GetComponent<MeshRenderer>().sharedMaterial;
    }

    //Ship map should always rotate to the direction the ship is facing
    void Update()
    {
        Vector3 forward = Vector3.ProjectOnPlane(shipWeight.transform.forward,
Vector3.up);

        starMap.SetFloat("_rotationRad", Mathf.Atan2(forward.z, forward.x));
    }
}
```

ShipRouter.cs

```

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

public class ShipRouter : MonoBehaviour
{
    public GameObject Map;
    private Material globeMap;
    public Transform[] Legs;
    private float[] old_extensions;
    private float[] extensions;
    private float lerp = 0;

    public float shipFuelBurnRate;
    public float shipFuelRemaining;

    [SerializeField] private ShipDoorOpen outerDoor;
    public Vector3 centre = new Vector3(0, 0.3f, 0.8f);
    public Vector3 size = new Vector3(12, 5.6f, 14.5f);

    public void Start()
    {
        globeMap = Map.GetComponent<MeshRenderer>().sharedMaterial;
        old_extensions = new float[Legs.Length];
        extensions = new float[Legs.Length];
        for (int i = 0; i < extensions.Length; i++)
        {
            old_extensions[i] = 1;
            extensions[i] = 1;
        }

        Update();
    }
    public void Update()
    {
        shipFuelBurnRate = (InventoryUI.shipFuelRemaining - shipFuelRemaining) / Time.deltaTime;
        shipFuelRemaining = InventoryUI.shipFuelRemaining;

        globeMap.SetFloat("_engineOn", InventoryUI.shipEngineOn01);
        RecheckCameraInShip();
        UpdateLegs();
    }
    public bool RecheckCameraInShip()
    {
        Vector3 p = transform.InverseTransformDirection(Camera.main.transform.position - transform.position);
        Vector3 ceil = centre + size / 2;
        Vector3 floor = centre - size / 2;
        bool check = floor.x < p.x && p.x < ceil.x;
        if (check)
            check &= floor.y < p.y && p.y < ceil.y; //kinda the same as check &=
        if (check)
            check &= floor.z < p.z && p.z < ceil.z;

        CameraState.withinShip = check;
    }
}

```

```
//In ship if within ship and (outer door is closed or the player has just died
meaning the outer door may still be closing)
    CameraState.inShip = CameraState.withinShip && (outerDoor.lerp <= 0 ||
Time.realtimeSinceStartup < StarGenSystem.genTime + 5);

    return CameraState.inShip;
}
public void CastLegs()
{
    lerp = 0;
}

public void ResetLegs()
{
    if (extensions[0] == 1)
        return;

    //Set taget of legs to their withdrawn length
    for (int i = 0; i < Legs.Length; i++)
    {
        old_extensions[i] = 1;
        extensions[i] = 1;
    }
}
public void UpdateLegs()
{
    int i = 0;
    if (lerp == 0)
    {
        foreach (Transform leg in Legs)
        {
            old_extensions[i] = extensions[i];
            if (Physics.Raycast(leg.position, -leg.transform.up, out RaycastHit
hitInfo, 10, ~(1 << 11 | 1 << 12)))
                extensions[i] = hitInfo.distance * 0.9f;
            i++;
        }
    }

    lerp += Time.deltaTime;
    i = 0;
    foreach (Transform leg in Legs)
    {
        leg.localScale = new Vector3(leg.localScale.x,
Mathf.Lerp(old_extensions[i], extensions[i], lerp), leg.localScale.z);
        i++;
    }
}
```

ShipWeight.cs

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

public class ShipWeight : ZeroWeight
{
    private Vector3 normalUp = Vector3.up;
    private ShipRouter shipRouter;
    [SerializeField] private FlipSwitch wingOpen;
    [SerializeField] private Transform[] wingIndicators;
    private Vector3[] wingIndicatorsStartLocalPos;
    private Vector3[] wingIndicatorsEndLocalPos;

    public static bool xRotatesYaw;
    private float wingSwapLerp;
    [SerializeField] private float wingSwapLerpTime = 0.2f;
    float wingSwapCooldown;

    private float legCastTimer;
    private bool normalContacted;

    [SerializeField] private float lookSpeed = 2;

    private Quaternion alignRot;

    private float xRotation = 0;
    private float yRotation = 0;
    private float zRotation = 0;

    [SerializeField] private float thrustSpeed = 0.1f;
    public float maxVelocity;
    public float maxHyperVelocity;
    private float hyperLerp;
    public static bool hyperOn;

    public bool onHelipad;

    [SerializeField] private RobotWeight robotWeight;

    public override void Start()
    {
        alignRot = transform.rotation;
        shipRouter = transform.GetChild(0).GetComponent<ShipRouter>();
        isShip = true;
        legCastTimer = 0;

        wingSwapLerp = 1;
        xRotatesYaw = true;
        wingIndicatorsEndLocalPos = new Vector3[2] { wingIndicators[0].localPosition,
wingIndicators[1].localPosition };
        wingIndicatorsStartLocalPos = new Vector3[2] { wingIndicators[0].localPosition - Vector3.right, wingIndicators[1].localPosition - Vector3.right };
        wingOpen.override = true;
        wingSwapCooldown = 0;

        base.Start();
    }
    private void Update()
```

```

{
    wingSwapCooldown += Time.deltaTime;

    if (InventoryUI.shipEngineOn01 <= 0)
        return;

    if (CameraState.flyingShip && Input.GetKeyDown(KeyCode.R) && wingSwapCooldown
> 1)
    {
        wingOpen.override = true;
        wingSwapLerp = 0;
        wingSwapCooldown = 0;
        xRotatesYaw ^= true;
        (wingIndicatorsStartLocalPos, wingIndicatorsEndLocalPos) =
(wingIndicatorsEndLocalPos, wingIndicatorsStartLocalPos);
    }
    else if (!CameraState.flyingShip && wingOpen.switchState ==
FlipSwitch.State.bottom != xRotatesYaw)
    {
        wingSwapLerp = 0;
        wingSwapCooldown = 0;
        xRotatesYaw ^= true;
        (wingIndicatorsStartLocalPos, wingIndicatorsEndLocalPos) =
(wingIndicatorsEndLocalPos, wingIndicatorsStartLocalPos);
    }
    if (Input.GetKeyDown(KeyCode.X))
        hyperOn ^= true;
    //Turn hyper mode off for when player returns to ship to launch
    if (!CameraState.flyingShip && sigWeight != null)
        hyperOn = false;
}
public override Vector3 MoveRelative(Vector3 displacement)
{
    //If player is exploring hive or ship is grounded and player is exploring
    planet and is not on the helipad, don't move as this will result in clipping when the
    terrain collier is unloaded
    if (sigWeight != null && !CameraState.inShip && (CameraState.inHive || 
normalContacted && !onHelipad))
    {
        return Vector3.zero;
    }
    //Will be set to true by Helipad.cs if on helipad between MoveRelative calls
    onHelipad = false;

    transform.GetChild(1).gameObject.SetActive(sigWeight == null);

    //Cast legs when continuously normal contacted / grounded for a second
    float newTime = Mathf.Clamp01(legCastTimer + Time.fixedDeltaTime *
(normalContacted ? 1 : -3));
    if (newTime == 1 && legCastTimer != 1)
        shipRouter.CastLegs();
    else if (newTime == 0 && legCastTimer != 0)
        shipRouter.ResetLegs();

    legCastTimer = newTime;

    if (!CameraState.flyingShip)
    {
        //Velocity streaks set to default rotation as otherwise they are visible
        inside the ship
        transform.GetChild(1).localRotation = Quaternion.Euler(90, 0, 0);
    }
}

```

```

        RotateToGround();

        return base.MoveRelative(displacement);
    }
    //Rotate velocity streaks to direction of motion
    transform.GetChild(1).rotation = Quaternion.LookRotation(velocity) *
    Quaternion.Euler(90, 0, 0);

    hyperLerp = Mathf.Clamp01(hyperLerp + Time.fixedDeltaTime * (hyperOn ? 1 : -1));

    wingSwapLerp = Mathf.Clamp01(wingSwapLerp + Time.fixedDeltaTime /
    wingSwapLerpTime);

    wingIndicators[0].localPosition = Vector3.Lerp(wingIndicatorsStartLocalPos[0],
    wingIndicatorsEndLocalPos[0], wingSwapLerp);
    wingIndicators[1].localPosition = Vector3.Lerp(wingIndicatorsStartLocalPos[1],
    wingIndicatorsEndLocalPos[1], wingSwapLerp);

    if (!normalContacted)
    {
        //Start to ignore the planet's normal up
        normalUp = Vector3.RotateTowards(normalUp, transform.up, 6 *
Mathf.Deg2Rad, 1);
    }
    if (normalContacted)
        RotateToGround();
    else if (!hyperOn)
        Rotate(GetRotationCommand());
    else if (hyperOn)
    {
        xRotation = 0;
        yRotation = 0;
        zRotation = 0;
    }

    velocity += GetMoveCommand();

    float maxCurrentVelocity = Mathf.Lerp(maxVelocity, maxHyperVelocity,
hyperLerp);
    if (velocity.sqrMagnitude > maxCurrentVelocity * maxCurrentVelocity)
        velocity = velocity.normalized * maxCurrentVelocity;

    return base.MoveRelative(displacement);
}
public void ApplyNormal(Vector3 normalDir)
{
    //Normals applied are filtered so that increasingly only those closer to the
    normalUp are selected over 1 second
    //The normalUp is then smoothly slerped towards the normal applied by 0% at t
    = 0, 1 and 10% at t = 0.5
    if (Vector3.Dot(normalUp, normalDir) > legCastTimer)
        normalUp = Vector3.Slerp(normalUp, normalDir, 0.1f *
Mathf.Sin(legCastTimer * Mathf.PI));

    //Bounce the ship back if do not land near ship legs
    if ((sigWeight == null || legCastTimer < 1) && Vector3.Dot(transform.up,
normalDir) < 0.5f)
        velocity += normalDir * 10;

    normalContacted = true;
}

```

```

private Vector3 GetMoveCommand()
{
    Vector3 direction = transform.forward * Input.GetAxis("Vertical") +
    transform.right * Input.GetAxis("Horizontal") + transform.up *
    Input.GetAxis("Liftoff");

    bool matching = Input.GetKey(KeyCode.Space) && InventoryUI.shipTargetX != null;
    if (matching)
        direction += (InventoryUI.shipTargetX.velocity - velocity).normalized;

    return direction.normalized * thrustSpeed / Time.fixedDeltaTime *
    Mathf.Lerp(matching ? 2 : 1, 25, hyperLerp) * InventoryUI.shipEngineOn01;
}

private Quaternion GetRotationCommand()
{
    if (InventoryUI.shipEngineOn01 > 0)
    {
        xRotation -= Mathf.Clamp(Input.GetAxis("Mouse Y"), -1, 1) * lookSpeed;
        if (xRotatesYaw)
            yRotation += Mathf.Clamp(Input.GetAxis("Mouse X"), -1, 1) * lookSpeed;
        else
            zRotation -= Mathf.Clamp(Input.GetAxis("Mouse X"), -1, 1) * lookSpeed;
    }

    xRotation = Mathf.Lerp(xRotation, 0, Time.fixedDeltaTime * 0.5f);
    yRotation = Mathf.Lerp(yRotation, 0, Time.fixedDeltaTime * 0.5f);
    zRotation = Mathf.Lerp(zRotation, 0, Time.fixedDeltaTime * 0.5f);

    Quaternion rotation = rb.rotation * Quaternion.Euler(xRotation, yRotation,
    zRotation);

    Vector3 up = sigWeight == null ? transform.up : (position -
    sigWeight.position).normalized;
    //If on sun or in planet atmosphere
    if (sigWeight != null && (sigWeight.planet == null || (position -
    sigWeight.position).sqrMagnitude <
    Mathf.Pow(sigWeight.transform.GetChild(2).GetComponent<PlanetEffect>().atmosRadius,
    2)))
    {
        //Align rotation with local planet normal
        alignRot = Quaternion.RotateTowards(alignRot,
        Quaternion.LookRotation(Vector3.ProjectOnPlane(transform.forward, up), up), 0.5f);
        rotation = Quaternion.RotateTowards(rotation, alignRot,
        Mathf.Pow(Quaternion.Angle(rotation, alignRot), 0.6f));
    }
    else
        alignRot =
    Quaternion.LookRotation(Vector3.ProjectOnPlane(transform.forward, up), up);

    return Quaternion.Slerp(rb.rotation, rotation, Time.fixedDeltaTime);
}
private void RotateToGround()
{
    normalContacted = false;

    Quaternion target =
    Quaternion.LookRotation(Vector3.ProjectOnPlane(transform.forward, normalUp),
    normalUp);
    alignRot = Quaternion.RotateTowards(rb.rotation, target, 6);
}

```

```

        Rotate(alignRot);
    }

    private void Rotate(Quaternion rotation)
    {
        //So that the robot does not clip out of the ship when the ship rotates
        if (!CameraState.InLockState(CameraState.LockState.unlocked))
            robotWeight.RotateAsChild(position, rotation *
        Quaternion.Inverse(rb.rotation));
        rb.MoveRotation(rotation);
    }
}

```

ShipWingOpen.cs

```

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

public class ShipWingOpen : MonoBehaviour
{
    [SerializeField] private FlipSwitch trigger;
    [SerializeField] private float closedAngle = 300;
    [SerializeField] private float openAngle = 75;
    private Quaternion closedRot;
    private Quaternion openRot;

    private float lerp;
    [SerializeField] private float lerpSpeed = 1;

    private void Start()
    {
        lerp = 0;
        closedRot = Quaternion.Euler(closedAngle, transform.localEulerAngles.y,
        transform.localEulerAngles.z);
        openRot = Quaternion.Euler(openAngle, transform.localEulerAngles.y,
        transform.localEulerAngles.z);
    }

    private void Update()
    {
        if (trigger.switchState == FlipSwitch.State.top && lerp < 1)
            lerp += lerpSpeed * Time.deltaTime;
        else if (trigger.switchState == FlipSwitch.State.bottom && lerp > 0)
            lerp -= lerpSpeed * Time.deltaTime;

        transform.localRotation = Quaternion.Slerp(closedRot, openRot, lerp);
    }
}

```

SittingRobot.cs

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

public class SittingRobot : MonoBehaviour
{
    /* Just another day?
     * This robot's just sitting...
     * It is listening to something though:
     * https://youtu.be/Z3m7HXeiHpg
     */
}
```

StarGenSystem.cs

```

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

public class StarGenSystem : MonoBehaviour
{
    public enum GenPurpose { starting, respawning, wiping }

    public static Vector3 galaxyCentre;
    public int kingSeed;

    [SerializeField] private Material starMat;
    [SerializeField] private Material starMatSmall;
    private Texture2D starMap;
    [SerializeField] private ComputeShader starUpdateShader;
    private int handle;
    private Vector3[] starPosRelativeToGalaxy;
    private float[] starSphereRadii;
    private Vector3[] starRenderPos;
    private float[] starRenderLocalScaleX;

    [SerializeField] private GameObject starPrefab;
    public SunGenSystem sun;
    [SerializeField] private float galaxyRadius;
    public float starcoordLength = 250_000;
    [SerializeField] private int closeScale = 30_000;
    [SerializeField] private int approxStars;
    [SerializeField] private float updateStarsEachFrame;
    [SerializeField] private float probabilityScale;
    [SerializeField] float minPixelStarSize = 1;
    [SerializeField] float renderDistance = 100_000;
    private float minAt1Dist = -1;

    private Dictionary<Transform, StarCircle> stars;
    public Dictionary<string, int> starCodes { get; private set; }
    private string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    public int HME { get; private set; }
    public int EDN { get; private set; }
    public float GUL { get; private set; }

    private List<int> visitedSystems;
    public Transform closestStar;
    public bool generatedAtStar;

    public static float genTime;

    public struct StarCircle
    {
        public Vector3 starPosRelativeToGalaxy;
        public float starSphereRadius;

        public Color starColour;
        public Color sunColour;

        public StarCircle(Vector3 starPosRelativeToGalaxy, float starSphereRadius, float hue, float absoluteSaturation)
        {
    
```

```

        this.starPosRelativeToGalaxy = starPosRelativeToGalaxy;
        this.starSphereRadius = starSphereRadius;
        starColour = Color.HSVToRGB(hue, absoluteSaturation * 0.4f - 0.2f, 1);
        sunColour = Color.HSVToRGB(hue, absoluteSaturation, 1);
    }
}

[System.Serializable]
private struct GalaxySaveData
{
    public int kingSeed;
    public Vector3 galaxyCentre;
    public int[] visitedSystems;
}

private bool LoadTheGalaxy()
{
    GalaxySaveData galaxySaveData =
JsonSaver.LoadData<GalaxySaveData>("Galaxy_Save_Data", out bool success);

    kingSeed = galaxySaveData.kingSeed;
    galaxyCentre = galaxySaveData.galaxyCentre;
    if (success)
        visitedSystems = galaxySaveData.visitedSystems.ToList();

    if (generatedAtStar)
    {
        generatedAtStar = false;
        UnloadSun();
    }

    return success;
}
private void SaveTheGalaxy()
{
    GalaxySaveData galaxySaveData;
    galaxySaveData.kingSeed = kingSeed;
    galaxySaveData.galaxyCentre = galaxyCentre;
    galaxySaveData.visitedSystems = visitedSystems.ToArray();
    JsonSaver.SaveData("Galaxy_Save_Data", galaxySaveData);
    MeetingHandler.Save();
}

private void DrawVisitedGalaxy()
{
    Color32[] starMapColours = new Color32[starMap.width * starMap.height];

    foreach (int i in visitedSystems)
    {
        Vector3 position01 = starPosRelativeToGalaxy[i] / galaxyRadius * 0.5f +
0.5f * Vector3.one;
        starMapColours[Mathf.FloorToInt(position01.x * starMap.width) *
starMap.height + Mathf.FloorToInt(position01.z * starMap.height)] =
stars[transform.GetChild(0).GetChild(i)].starColour;
    }

    starMap.SetPixels32(starMapColours);

    starMap.Apply();
    starMat.SetTexture("_starTexture", starMap);
}

```

```

    }

    private void ApplyToSmallMap()
    {
        Vector3 playerPos = -galaxyCentre / galaxyRadius * 0.5f + 0.5f * Vector3.one;
        int scale = Mathf.RoundToInt(5 * starcoordLength / galaxyRadius * 0.5f *
starMap.width);

        Vector2Int playerPix = new Vector2Int(Mathf.RoundToInt(playerPos.z * starMap.width), Mathf.RoundToInt(playerPos.x * starMap.height));

        Color[] starMapSmallColours = starMap.GetPixels(Mathf.Clamp(playerPix.x - scale, 0, starMap.width - 2 * scale), Mathf.Clamp(playerPix.y - scale, 0, starMap.height - 2 * scale), 2 * scale, 2 * scale);

        Texture2D starMapSmall = new Texture2D(2 * scale, 2 * scale);
        starMapSmall.SetPixels(starMapSmallColours);
        starMapSmall.Apply();

        starMatSmall.SetTexture("_starTexture", starMapSmall);
    }

    public void InitGalaxy(GenPurpose purpose)
    {
        CameraState.isDead = false;

        if (transform.childCount > 0)
        {
            if (Application.isPlaying)
                Destroy(transform.GetChild(0).gameObject);
            else
                DestroyImmediate(transform.GetChild(0).gameObject);
        }

        starMap = new Texture2D(960, 960);
        visitedSystems = new List<int>();

        galaxyRadius = 2 * starcoordLength * Mathf.Sqrt(approxStars);

        Random.initState(System.Environment.TickCount);

        GUL = Random.value;

        //If there is no save or we are making a new save
        if (!LoadTheGalaxy() || purpose == GenPurpose.wiping)
        {
            Vector3 randomRadius = Random.onUnitSphere * Random.Range(0.5f * galaxyRadius, 0.6f * galaxyRadius);
            galaxyCentre = new Vector3(randomRadius.x, 0.01f * randomRadius.y, randomRadius.z);

            kingSeed = Random.Range(0, 999999);

            visitedSystems = new List<int>();
            MeetingHandler.Wipe();
            InventoryUI.Wipe();
            FindObjectOfType<SadGirl>().Wipe();
            SaveTheGalaxy();
        }
        //If we are not starting the game, Init the galaxy with the player in a random
        position in it (between 50% & 60% out)
        if (purpose != GenPurpose.starting && Application.isPlaying)
    }
}

```

```

    {
        Vector3 randomRadius = Random.onUnitSphere * Random.Range(0.5f *
galaxyRadius, 0.6f * galaxyRadius);
        galaxyCentre = new Vector3(randomRadius.x, 0.01f * randomRadius.y,
randomRadius.z);

        ShipWeight shipWeight = FindObjectOfType<ShipWeight>();
        shipWeight.Teleport(-shipWeight.position);
        shipWeight.transform.rotation =
FindObjectOfType<RobotWeight>().transform.rotation;
        PhysicsUpdate.Reinit();
        CameraState.Reset();
        FindObjectOfType<InventoryUI>().Start();
    }
}

public void GenerateStars(GenPurpose purpose)
{
    genTime = Time.realtimeSinceStartup;
    FindObjectOfType<RoboVision>().KillTransition();
    InitGalaxy(purpose);

    Transform almightyHolder = new GameObject("AlmightyHolder (0)").transform;
    almightyHolder.parent = transform;
    almightyHolder.SetAsFirstSibling();

    Random.InitState(kingSeed);

    int capacity = Mathf.CeilToInt(approxStars * 1.25f);

    stars = new Dictionary<Transform, StarCircle>(capacity);
    starCodes = new Dictionary<string, int>(capacity);

    //Need individual lists as well as the StarCircles for our compute shader
    List<Vector3> starPosRelativeToGalaxy = new List<Vector3>(capacity);
    List<float> starSphereRadii = new List<float>(capacity);
    List<Vector3> starRenderPos = new List<Vector3>(capacity);
    List<float> starRenderLocalScaleX = new List<float>(capacity);

    float spiralAngle = Random.Range(Mathf.PI, 2 * Mathf.PI);
    float spiralDirection = Random.Range(0, 2) * 2 - 1;

    Color32[] starMapColours = new Color32[starMap.width * starMap.height];

    float minCamSqrDist = float.MaxValue;
    float minGalSqrDist = float.MaxValue;
    float maxGalSqrDist = float.MinValue;

    float totalScale = 0;
    for (float x = -galaxyRadius; x < galaxyRadius; x += starcoordLength)
    {
        for (float y = -galaxyRadius; y < galaxyRadius; y += starcoordLength)
        {
            //More likely to generate stars closer to the centre of the galaxy and
galaxy should have a double spiral shape
            if (1.35f * Mathf.Pow(Random.value * 0.5f, 2) > (x * x + y * y) /
(galaxyRadius * galaxyRadius) &&
                (Mathf.Abs((spiralDirection * Mathf.Atan2(y, x) + spiralAngle) %
Mathf.PI - 2 * Mathf.PI * Mathf.Sqrt(x * x + y * y) / galaxyRadius) > Random.value))
            {
                //Each star is assinged a unique cuboid in which it can generate

```

```

        Vector3 _starPosRelativeToGalaxy = new Vector3(x, 0, y) +
starcoordLength * Vector3.Scale(Random.insideUnitSphere, new Vector3(0.5f, 1, 0.5f));

        StarCircle star = new StarCircle(_starPosRelativeToGalaxy, 5000,
Random.value, Random.Range(0.5f, 1));
        Vector3 starPosRelativeToCamera = star.starPosRelativeToGalaxy +
galaxyCentre;
        Transform starInstance = Instantiate(starPrefab,
starPosRelativeToCamera.normalized * renderDistance,
Quaternion.LookRotation(starPosRelativeToCamera), almightyHolder).transform;

        starInstance.localScale =
GetCircleStarScale(star.starSphereRadius, starPosRelativeToCamera.sqrMagnitude);
        totalScale += starInstance.localScale.x;

        starInstance.GetComponent<MeshRenderer>().sharedMaterial = new
Material(starInstance.GetComponent<MeshRenderer>().sharedMaterial);

starInstance.GetComponent<MeshRenderer>().sharedMaterial.SetColor("_starColour",
Color.Lerp(star.starColour, star.sunColour, Mathf.Clamp01(starInstance.localScale.x /
closeScale)));

starInstance.GetComponent<MeshRenderer>().sharedMaterial.SetFloat("_brightness",
Mathf.Lerp(1, 5, Mathf.Clamp01(starInstance.localScale.x / 30_000)));

        Vector3 i01 = _starPosRelativeToGalaxy / galaxyRadius * 0.5f +
0.5f * Vector3.one;
        starMapColours[Mathf.FloorToInt(i01.x * starMap.width) *
starMap.height + Mathf.FloorToInt(i01.z * starMap.height)] = star.starColour;

stars.Add(starInstance, star);

System.Random prng = new System.Random(stars.Count);
string starCode;
do
{
    starCode = alphabet[prng.Next(0, 26)].ToString() +
alphabet[prng.Next(0, 26)].ToString() + alphabet[prng.Next(0, 26)].ToString();
}
while (starCodes.ContainsKey(starCode) || starCode == "HME" ||
starCode == "END" || starCode == "GUL");

starCodes.Add(starCode, stars.Count - 1);

//visitedSystems.Add(stars.Count - 1);

starPosRelativeToGalaxy.Add(_starPosRelativeToGalaxy);
starSphereRadii.Add(5000);
starRenderPos.Add(starInstance.position);
starRenderLocalScaleX.Add(starInstance.localScale.x);

if (starPosRelativeToCamera.sqrMagnitude < minCamSqrDist)
    (closestStar, minCamSqrDist) = (starInstance,
starPosRelativeToCamera.sqrMagnitude);

if (_starPosRelativeToGalaxy.sqrMagnitude < minGalSqrDist)
    (HME, minGalSqrDist) = (stars.Count - 1,
_starPosRelativeToGalaxy.sqrMagnitude);
if (_starPosRelativeToGalaxy.sqrMagnitude > maxGalSqrDist)
    (EDN, maxGalSqrDist) = (stars.Count - 1,
_starPosRelativeToGalaxy.sqrMagnitude);

```

```

        }
    }
    if (stars.Count > approxStars * 1.25f)
    {
        Debug.LogError("Error: too many stars");
        break;
    }
}

List<string> binCodes = new List<string>(3);
foreach (string starCode in starCodes.Keys)
{
    if (starCodes[starCode] == HME)
        binCodes.Add(starCode);
    else if (starCodes[starCode] == EDN)
        binCodes.Add(starCode);
    else if (starCodes[starCode] == Mathf.FloorToInt(GUL * stars.Count))
        binCodes.Add(starCode);
}
foreach (string binCode in binCodes)
    starCodes.Remove(binCode);
starCodes.Add("HME", HME);
starCodes.Add("EDN", EDN);
starCodes.Add("GUL", Mathf.FloorToInt(GUL * stars.Count));

float meanScale = totalScale / stars.Count;
probabilityScale = updateStarsEachFrame / (Mathf.Pow(meanScale / closeScale,
2) * stars.Count);

this.starPosRelativeToGalaxy = starPosRelativeToGalaxy.ToArray();
this.starSphereRadii = starSphereRadii.ToArray();
this.starRenderPos = starRenderPos.ToArray();
this.starRenderLocalScaleX = starRenderLocalScaleX.ToArray();

for (int i = 0; i < stars.Count; i++)
{
    float sqrDist = (starPosRelativeToGalaxy[i] + galaxyCentre).sqrMagnitude;

    if (sqrDist < 25 * starcoordLength * starcoordLength &&
!visitedSystems.Contains(i))
        visitedSystems.Add(i);
}

DrawVisitedGalaxy();
ApplyToSmallMap();
}

public void PathBetween(int startStarIndex, int endStarIndex, Color color)
{
    DrawVisitedGalaxy();

    Dijkstras pathFinder = new Dijkstras(starPosRelativeToGalaxy, 10 *
starcoordLength);
    pathFinder.ShortestPath(startStarIndex, endStarIndex, out List<int>
pathIndices, out _);
    Color32[] starMapColours = starMap.GetPixels32();

    Vector3 previ01 = starPosRelativeToGalaxy[pathIndices[0]] / galaxyRadius *
0.5f + 0.5f * Vector3.one;
    for (int i = 1; i < pathIndices.Count; i++)
    {

```

```

        Vector3 i01 = starPosRelativeToGalaxy[pathIndices[i]] / galaxyRadius *
0.5f + 0.5f * Vector3.one;
        //Draw line between previous node and current node on path
        for (float j = 0; j <= 1; j += 0.1f)
        {
            Vector3 lerpi01 = Vector3.Lerp(previ01, i01, j);

            starMapColours[Mathf.FloorToInt(lerpi01.x * starMap.width) *
starMap.height + Mathf.FloorToInt(lerpi01.z * starMap.height)] = color;
        }
        previ01 = i01;
    }
    starMap.SetPixels32(starMapColours);
    starMap.Apply();
    starMat.SetTexture("_starTexture", starMap);

    ApplyToSmallMap();
}

private Vector3 GetCircleStarScale(float starRadius, float sqrStarDist)
{
    //stars are at least rendered by minPixelStarSize pixels, with help from:
https://stackoverflow.com/questions/21648630/radius-of-projected-sphere-in-screen-space
    if (minAt1Dist == -1)
        minAt1Dist = minPixelStarSize / Mathf.Sqrt(Mathf.Pow(Screen.height * 0.5f
/ Mathf.Tan(0.5f * Camera.main.fieldOfView * Mathf.Deg2Rad), 2) + minPixelStarSize *
minPixelStarSize);

    float horizonRadius = Mathf.Max(minAt1Dist, Mathf.Sqrt(sqrStarDist -
starRadius * starRadius) * starRadius / sqrStarDist);

    float scale = 2 * horizonRadius * renderDistance;

    return new Vector3(scale, scale, 1);
}
public bool UpdateStarColour(int index)
{
    Transform starInstance = transform.GetChild(0).GetChild(index);
    float close01 = Mathf.Clamp01(starInstance.localScale.x / closeScale); //1 is
close

    starInstance.GetComponent<MeshRenderer>().sharedMaterial.SetColor("_starColour",
Color.Lerp(stars[starInstance].starColour, stars[starInstance].sunColour, close01 *
close01));

    starInstance.GetComponent<MeshRenderer>().sharedMaterial.SetFloat("_brightness",
Mathf.Lerp(1, 5, close01 * close01));
    //starInstance.GetComponent<MeshRenderer>().enabled ^= true;

    return true;
}
// Update is called once per frame
void Start()
{
    handle = starUpdateShader.FindKernel("UpdateStars");
    GenerateStars(GenPurpose.starting);
    InvokeRepeating(nameof(ApplyToSmallMap), 3, 3);
}
private void Update()

```

```

{
    if (PauseMenu.selectedPauseMenuOption == 0)
        GenerateStars(GenPurpose.wiping);
    else if (PauseMenu.selectedPauseMenuOption == 1 || CameraState.isDead &&
Time.realtimeSinceStartup > genTime + 5)
        GenerateStars(GenPurpose.respawning);

    //Mark the player's position
    starMat.SetVector("_playerPos", -galaxyCentre / galaxyRadius * 0.5f + 0.5f *
Vector3.one);
    starMat.SetFloat("_engineOn", InventoryUI.shipEngineOn01);
    starMatSmall.SetFloat("_engineOn", InventoryUI.shipEngineOn01);

    int count = stars.Count;

    ComputeBuffer starPosRelativeToGalaxyBuffer = new ComputeBuffer(count,
sizeof(float) * 3);
    ComputeBuffer starSphereRadiiBuffer = new ComputeBuffer(count, sizeof(float));
    ComputeBuffer starRenderPosBuffer = new ComputeBuffer(count, sizeof(float) *
3);
    ComputeBuffer starRenderLocalScaleXBuffer = new ComputeBuffer(count,
sizeof(float));

    starPosRelativeToGalaxyBuffer.SetData(starPosRelativeToGalaxy);
    starSphereRadiiBuffer.SetData(starSphereRadii);
    starRenderPosBuffer.SetData(starRenderPos);
    starRenderLocalScaleXBuffer.SetData(starRenderLocalScaleX);

    starUpdateShader.SetBuffer(handle, "starPosRelativeToGalaxy",
starPosRelativeToGalaxyBuffer);
    starUpdateShader.SetBuffer(handle, "starSphereRadii", starSphereRadiiBuffer);
    starUpdateShader.SetBuffer(handle, "starRenderPos", starRenderPosBuffer);
    starUpdateShader.SetBuffer(handle, "starRenderLocalScaleX",
starRenderLocalScaleXBuffer);

    starUpdateShader.SetFloat("numStars", count);
    starUpdateShader.SetVector("galaxyCentre", galaxyCentre);
    starUpdateShader.SetFloat("minAt1Dist", minAt1Dist);
    starUpdateShader.SetFloat("renderDistance", renderDistance);

    starUpdateShader.Dispatch(handle, Mathf.CeilToInt(count / 128f), 1, 1);

    starRenderPosBuffer.GetData(starRenderPos);
    starRenderLocalScaleXBuffer.GetData(starRenderLocalScaleX);

    starPosRelativeToGalaxyBuffer.Dispose();
    starSphereRadiiBuffer.Dispose();
    starRenderPosBuffer.Dispose();
    starRenderLocalScaleXBuffer.Dispose();

    Transform newClosestStar = closestStar;
    float minSqrDist = float.MaxValue;

    for (int i = 0; i < count; i++)
    {
        Transform starInstance = transform.GetChild(0).GetChild(i);

        float sqrDist = (starPosRelativeToGalaxy[i] + galaxyCentre).sqrMagnitude;

        if (sqrDist < 25 * starcoordLength * starcoordLength &&
!visitedSystems.Contains(i))
            visitedSystems.Add(i);
    }
}

```

```

        if (!(starInstance == closestStar && generatedAtStar) && (Random.value <
probabilityScale * Mathf.Pow(starRenderLocalScaleX[i] / closeScale, 2)))
            UpdateStarColour(i);

        //Determine if the star is within the camera's view frustum
        Vector3 viewPoint = Camera.main.WorldToViewportPoint(starRenderPos[i]);
        if (starInstance != closestStar && (viewPoint.x < -0.1f || viewPoint.x >
1.1f || viewPoint.y < -0.1f || viewPoint.y > 1.1f || viewPoint.z < 0))
        {
            //If not, don't bother rendering it
            starInstance.gameObject.SetActive(false);
            continue;
        }

        //If it is, enable its game object and update its position, rotation, and
scale
        starInstance.gameObject.SetActive(true);
        starInstance.SetPositionAndRotation(starRenderPos[i],
Quaternion.LookRotation(starRenderPos[i]));
        starInstance.localScale = new Vector3(starRenderLocalScaleX[i],
starRenderLocalScaleX[i], 1);

        if (sqrDist < minSqrDist)
        {
            newClosestStar = starInstance;
            minSqrDist = sqrDist;
        }
    }
    if (closestStar != newClosestStar)
        generatedAtStar = false;
    closestStar = newClosestStar;

    UpdateStarColour(closestStar.GetSiblingIndex());

    if ((stars[closestStar].starPosRelativeToGalaxy + galaxyCentre).sqrMagnitude <
starcoordLength * starcoordLength)
    {
        if (!generatedAtStar)
        {
            SaveTheGalaxy();
            generatedAtStar = true;
            LoadSun();

            if (!visitedSystems.Contains(closestStar.GetSiblingIndex()))
                visitedSystems.Add(closestStar.GetSiblingIndex());
            DrawVisitedGalaxy();
            ApplyToSmallMap();
        }
        else
            //Hide the 2D sprite that has been replaced with the 3D sun
            closestStar.GetComponent<MeshRenderer>().enabled = false;
    }
    else if (generatedAtStar)
    {
        SaveTheGalaxy();
        generatedAtStar = false;
        UnloadSun();
    }
}

```

```
private void LoadSun()
{
    sun.gameObject.SetActive(true);

    foreach (Planet celestialBody in sun.celestialBodies)
        PhysicsUpdate.RemoveWeight(celestialBody.GetComponent<Weight>());
    PhysicsUpdate.RemoveWeight(sun.GetComponent<Weight>());

    float close01 = Mathf.Clamp01(closestStar.localScale.x / closeScale); //1 is
close
    Color settledSunColour = Color.Lerp(stars[closestStar].starColour,
stars[closestStar].sunColour, close01 * close01);
    sun.Generate(kingSeed + closestStar.GetSiblingIndex(),
stars[closestStar].starPosRelativeToGalaxy + galaxyCentre,
stars[closestStar].starSphereRadius, settledSunColour);
}

private void UnloadSun()
{
    closestStar.GetComponent<MeshRenderer>().enabled = true;
    sun.gameObject.SetActive(false);

    foreach (Planet celestialBody in sun.celestialBodies)
    {
        celestialBody.gameObject.SetActive(false);
        PhysicsUpdate.RemoveWeight(celestialBody.GetComponent<Weight>());
    }
    PhysicsUpdate.RemoveWeight(sun.GetComponent<Weight>());
}
```

SunGenSystem.cs

```

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

public class SunGenSystem : MonoBehaviour
{
    [SerializeField] private TMPro.TextMeshProUGUI systemText;
    public int numPlanets;
    [SerializeField] private int numMoons;
    public Planet[] celestialBodies;
    public GameObject Grass;
    public GameObject Stone;
    public Vector2 minMaxDist;
    public int lordSeed;
    private System.Random masterPrng;

    private void Start()
    {
        //transform.position = Random.insideUnitCircle * 10000000000;
        //lordSeed = Random.Range(0, 999999);
        Segment.InstantiateFoilage(this);
    }
    public void Generate(int seed, Vector3 position, float radius, Color sunColour)
    {

        transform.GetChild(1).GetComponent<MeshRenderer>().sharedMaterial.SetColor("_sunColour", sunColour);

        transform.GetChild(1).GetComponent<MeshRenderer>().sharedMaterial.SetColor("_sunColour", sunColour);
        Generate(seed, position, radius);
    }
    public void Generate(int seed, Vector3 position, float radius)
    {
        PhysicsUpdate.AddWeight(GetComponent<Weight>());

        lordSeed = seed;
        Random.InitState(lordSeed);
        masterPrng = new System.Random(lordSeed);

        numPlanets = Random.Range(0, 5);

        transform.position = position;
        transform.GetChild(1).localScale = 2 * radius * Vector3.one;

        for (int i = 0; i < celestialBodies.Length; i++)
        {
            if (i < numPlanets)
                StartCoroutine(Init(i));
            else
                celestialBodies[i].gameObject.SetActive(false);
        }
    }

    IEnumerator Init(int i)
    {
        systemText.enabled = true;
    }
}

```

```
yield return new WaitForSeconds(Time.deltaTime);
Planet planet = celestialBodies[i];
planet.gameObject.SetActive(true);

Random.InitState(masterPrng.Next(-9999, 9999));

float speed = Mathf.Sqrt(GetComponent<Weight>().mass * Weight.gConst);
Vector3 orbitNormal = Vector3.Slerp(Vector3.up, Random.onUnitSphere, 0.2f);
Vector3 startDir = Vector3.Cross(orbitNormal, Random.onUnitSphere).normalized;

//Such that the planet orbits the sun in an approximate circle on a plane
similar to the galaxy's
Vector3 initialVelocity = Vector3.Cross(orbitNormal, startDir).normalized *
speed;

planet.Create(transform.position + startDir * Random.Range(minMaxDist.x,
minMaxDist.y), initialVelocity, masterPrng);
PhysicsUpdate.AddWeight(planet.GetComponent<Weight>()));

systemText.enabled = false;
}
}
```

SurpriseAndroid.cs

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

public class SurpriseAndroid : MonoBehaviour
{
    [Range(0, 1)]
    [SerializeField] private float lerp = 0;
    private float lerpTime = 1.5f;

    private List<Transform> sittingParts;
    private List<Transform> movingParts;
    private List<Transform> lerpingParts;

    // Start is called before the first frame update
    void Start()
    {
        sittingParts = new List<Transform>();
        movingParts = new List<Transform>();
        lerpingParts = new List<Transform>();

        //The rig of the robot when sitting
        foreach (Transform part in
transform.GetChild(0).GetComponentsInChildren<Transform>(true))
            sittingParts.Add(part);
        //The rig of the robot when standing
        foreach (Transform part in
transform.GetChild(1).GetComponentsInChildren<Transform>(true))
            movingParts.Add(part);
        //The rig of the robot that will interpolate between sitting and standing
        foreach (Transform part in
transform.GetChild(2).GetComponentsInChildren<Transform>(true))
            lerpingParts.Add(part);

        lerp = 0;
    }
    private void OnValidate()
    {
        Start();
        LerpBetween();
    }
    // Update is called once per frame
    void Update()
    {
        if (lerp == 1)
            return;

        if ((Camera.main.transform.position - transform.position).sqrMagnitude < 25)
            lerp += Time.deltaTime / lerpTime;
        else
            lerp -= Time.deltaTime / lerpTime;

        lerp = Mathf.Clamp01(lerp);

        transform.GetChild(0).gameObject.SetActive(lerp == 0);
        transform.GetChild(1).gameObject.SetActive(lerp == 1);
        transform.GetChild(2).gameObject.SetActive(lerp > 0 && lerp < 1);
    }
}
```

```
    if (lerp > 0 && lerp < 1)
        LerpBetween();
}
void LerpBetween()
{
    for (int i = 0; i < lerpParts.Count; i++)
    {
        lerpParts[i].position = Vector3.Slerp(sittingParts[i].position,
movingParts[i].position, lerp);
        lerpParts[i].rotation = Quaternion.Slerp(sittingParts[i].rotation,
movingParts[i].rotation, lerp);
        lerpParts[i].localScale = Vector3.Slerp(sittingParts[i].localScale,
movingParts[i].localScale, lerp);
    }
}
```

Tetris.cs

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

public class Tetris
{
    public Material displayMat;
    public Texture2D font;

    [SerializeField] private int pixelSize = 3;

    private Vector2Int mainStartPos = new Vector2Int(10, 0);
    [SerializeField] private Vector2Int holdStartPos = new Vector2Int(5, 13);
    [SerializeField] private Vector2Int nextStartPos = new Vector2Int(21, 13);

    private Texture2D mainScreen;
    private Texture2D textScreen;

    private int[,] Tiles = new int[20, 10];
    private int[,] activeCoords = new int[4, 2];

    public bool gameOver = false;

    private const float defaultMoveSpeed = 0.15f;
    private const float moveSpeedMultiplier = 1.2f;
    private float autoMoveSpeed = 0.125f;
    private float levelUpBoundary = 2;

    private float Speed = 1;
    private float updateTime = 1;
    private float nextPress = 0;
    private float coyoteTime = 0;

    public int scoreValue { get; private set; }
    private int tetrisCombo = 0;
    private int scoreCombo = 0;

    private int brickType;
    private int heldBrickType = 0;
    private bool allowHold = true;
    private int[] bankBricks;
    private int[] nextBricks;
    private int nextBrickIndex = 1;

    private int activeX = 4;
    private int activeY = 0;
    private int activeRot = 0;

    private enum GameState { start, end, playing, paused }

    private GameState gameState = GameState.start;

    private string row0 = "0123456789 ";
    private string row1 = "abcdefghijklmnopqrstuvwxyz";
    private string row2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    private string row3 = "!*()-_=+;:'@\\|,<.>/?";
```

```

void Write(string sentence, Color textColour, int lineNo)
{
    Texture2D addition = SentenceTexture(sentence);

    for (int x = 0; x < addition.width; x++)
    {
        for (int y = 0; y < addition.height; y++)
        {
            textScreen.SetPixel(x, textScreen.height - (lineNo + 1) * addition.height
+ y + 1, Color.Lerp(textColour, Color.black, addition.GetPixel(x, y).grayscale));
        }
    }
}
Texture2D SentenceTexture(string word)
{
    Texture2D wordTexture = new Texture2D(6 * 2 * word.Length, 6 * 2);

    int i = 0;

    if (word.Length == 0)
        return null;

    foreach (char character in word)
    {
        CharIndex(out int fontX, out int fontY, character);

        int xOffset = i * 6 * 2;

        for (int x = 0; x < 6; x++)
        {
            for (int y = 0; y < 6; y++)
            {
                for (int x0 = 0; x0 < 2; x0++)
                {
                    for (int y0 = 0; y0 < 2; y0++)
                    {
                        wordTexture.SetPixel(xOffset + x * 2 + x0, y * 2 + y0,
font.GetPixel(fontX + x, fontY + y));
                    }
                }
            }
        }
        i++;
    }
    wordTexture.Apply();
    return wordTexture;
}
bool CharIndex(out int fontX, out int fontY, char character)
{
    if (row0.Contains(character.ToString()))
    {
        fontY = 0;
        fontX = row0.IndexOf(character) * 6;
    }
    else if (row1.Contains(character.ToString()))
    {

```

```

        fontY = 6;
        fontX = row1.IndexOf(character) * 6;
    }
    else if (row2.Contains(character.ToString()))
    {
        fontY = 12;
        fontX = row2.IndexOf(character) * 6;
    }
    else if (row3.Contains(character.ToString()))
    {
        fontY = 18;
        fontX = row3.IndexOf(character) * 6;
    }
    else
    {
        fontY = 0;
        fontX = row0.IndexOf(' ') * 6;
        return false;
    }
    return true;
}
private void ResetScreen()
{
    textScreen = new Texture2D(240, 135);
    for (int x = 0; x < textScreen.width; x++)
    {
        for (int y = 0; y < textScreen.height; y++)
        {
            textScreen.SetPixel(x, y, Color.black);
        }
    }
}

public void GameStart(Material _displayMat, Texture2D _font)
{
    scoreValue = 0;

    font = _font;
    ResetScreen();
    mainScreen = new Texture2D(30 * pixelSize, 20 * pixelSize);

    bankBricks = GetBrickOrder();
    nextBricks = GetBrickOrder();

    brickType = nextBricks[0];

    displayMat = new Material(_displayMat);

    displayMat.SetTexture("_screen", mainScreen);
    displayMat.SetTexture("_text", textScreen);

    Write(" HOLD      NEXT", Color.white, 0);
    Write("           SCORE", Color.white, 6);

    mainScreen.Apply();
    textScreen.Apply();

    RenderUiElement(holdStartPos, 0);
    RenderUiElement(nextStartPos, nextBricks[1]);
}

void Start()

```

```

    }

}

public void Update()
{
    if (Input.GetKeyDown(KeyCode.E))
    {
        if (gameState == GameState.start)
        {
            gameState = GameState.playing;
            GameStart(displayMat, null);
        }
        else if (gameState == GameState.paused)
            gameState = GameState.paused;
        else if (gameState == GameState.paused)
            gameState = GameState.playing;
    }

    if (gameState == GameState.playing)
        GameUpdate();
}

void FixedUpdate()
{
    if (gameState == GameState.playing)
        GameFixedUpdate();
}

public void GameUpdate()
{
    Write("          " + new string('0', 5 - scoreValue.ToString().Length) +
Mathf.Min(scoreValue, 99999).ToString(), Color.white, 7);

    textScreen.Apply();

    if (gameOver)
        return;

    int prevActiveX = activeX;
    int prevActiveY = activeY;
    int prevActiveRot = activeRot;

    if (Input.anyKeyDown)
        autoMoveSpeed = defaultMoveSpeed;

    if (Input.GetKeyDown(KeyCode.LeftArrow) ||
Input.GetKeyDown(KeyCode.RightArrow))
    {
        activeX = activeX == -1 ? 0 : activeX;

        int moveRot;
        if (Input.GetKeyDown(KeyCode.LeftArrow))
            moveRot = (activeRot + 1) % 4;
        else
            moveRot = (activeRot + 3) % 4;

        if (!CheckCollided(brickType, activeX, activeY, moveRot, Tiles))
            activeRot = moveRot;
    }
}

```

```

        else
    {
        int moveY = activeY;

        for (int i = 0; i < 4; i++)
    {
        moveY = Mathf.Max(0, moveY - 1);

        if (!CheckCollided(brickType, activeX, moveY, moveRot, Tiles))
    {
        activeRot = moveRot;
        activeY = moveY;
        updateTime = Time.time * Speed + 1;
        break;
    }
}
}

//Not else if here to allow player to rotate and move
if (Input.GetKeyDown(KeyCode.A) || (Input.GetKey(KeyCode.A) && Time.time >= nextPress))
{
    autoMoveSpeed = Mathf.Max(0.05f, autoMoveSpeed / moveSpeedMultiplier);
    nextPress = Time.time + autoMoveSpeed;

    int moveX = Mathf.Max((brickType == 1 && activeRot % 2 == 0) || (brickType == 2 && activeRot == 3) ? -1 : 0, activeX - 1);

    activeX = !CheckCollided(brickType, moveX, activeY, activeRot, Tiles) ?
moveX : activeX;
}
else if (Input.GetKeyDown(KeyCode.D) || (Input.GetKey(KeyCode.D) && Time.time >= nextPress))
{
    autoMoveSpeed = Mathf.Max(0.05f, autoMoveSpeed / moveSpeedMultiplier);
    nextPress = Time.time + autoMoveSpeed;

    int moveX = activeX + 1;

    activeX = !CheckCollided(brickType, moveX, activeY, activeRot, Tiles) ?
moveX : activeX;
}
else if (Input.GetKeyDown(KeyCode.S) || (Input.GetKey(KeyCode.S) && Time.time >= nextPress))
{
    autoMoveSpeed = Mathf.Max(0.05f, autoMoveSpeed / moveSpeedMultiplier);
    nextPress = Time.time + autoMoveSpeed;

    int moveY = activeY + 1;

    if (!CheckCollided(brickType, activeX, moveY, activeRot, Tiles))
    {
        activeY = moveY;
        updateTime = Time.time * Speed + 1;
    }
}

else if (Input.GetKeyDown(KeyCode.W))
{
    int moveY = activeY;
}

```

```

        while (moveY < 20 && !CheckCollided(brickType, activeX, moveY, activeRot,
Tiles))
            moveY++;

        activeY = moveY - 1;

        coyoteTime = Time.time - 0.5f;
    }
    else if (Input.GetKeyDown(KeyCode.LeftShift) && allowHold)
    {
        //audioSource.PlayOneShot(hold);

        allowHold = false;

        if (heldBrickType == 0)
        {
            (brickType, heldBrickType) = (nextBrickIndex == 0 ? nextBricks[0] :
nextBricks[nextBrickIndex - 1], brickType);
            NewBrick();
        }
        else
        {
            (brickType, heldBrickType) = (heldBrickType, brickType);

            activeX = 4;
            activeY = 0;
            activeRot = 0;
        }
        RenderUiElement(holderStartPos, heldBrickType);
    }

    //Push back from the right edge if necessary
    int[,] simulation = GetActiveCoords(brickType, activeX, activeY, activeRot);
    while (Overflow(simulation) && activeX != 0)
    {
        if (CheckCollided(brickType, activeX - 1, activeY, activeRot, Tiles))
//Only if pushing back from edge doesn't result in overlapping bricks
            break;

        activeX--;
        simulation = GetActiveCoords(brickType, activeX, activeY, activeRot);
    }
}

public void GameFixedUpdate()
{
    if (gameOver)
        return;

    int[,] activeCoords = GetActiveCoords(brickType, activeX, activeY, activeRot);

    Tiles = SetCoords(brickType, activeCoords, Tiles);

    Render(Tiles);

    //Clear the active brick from Tiles to make collision detection with other
bricks easier
    Tiles = SetCoords(0, activeCoords, Tiles);
}

```

```

    if (CheckGrounded(activeCoords))
    {
        if (activeY == 0)
        {
            System.Array.Clear(Tiles, 0, Tiles.Length);
            Render(Tiles);
            gameOver = true;

            gameState = GameState.end;
            /*
            StreamReader reader = new StreamReader("Assets/HighScore.txt");
            int highScore = int.Parse(reader.ReadToEnd());
            reader.Close();

            if (scoreValue > highScore)
            {
                highScore = scoreValue;
                StreamWriter writer = new StreamWriter("Assets/HighScore.txt",
false);
                writer.WriteLine(highScore);
                writer.Close();
            }
            */
            //Score.GetComponent<TextMesh>().text = "Score: " + scoreValue +
"\nLevel: " + (1 + (Speed - 1) * 4) + "\nHigh Score: " + highScore;
        }

        //Allow the player to shuffle the active brick around on top of the others
        else if (coyoteTime == 0)
            coyoteTime = Time.time + 0.5f;

        //If the player has stopped shuffling fix brick in place and create new
brick at top
        else if (Time.time >= coyoteTime)
        {
            Tiles = SetCoords(brickType, activeCoords, Tiles);
            int lines_cleared;
            (Tiles, lines_cleared) = ClearFullLines(Tiles);

            if (lines_cleared == 0)
                scoreCombo = 0;
            else
            {
                if (lines_cleared != 4)
                    tetrisCombo = 0;
            }

            scoreCombo++;

            if (lines_cleared == 4)
                tetrisCombo++;

            int prevScoreValue = scoreValue;
            scoreValue += (int)Mathf.Pow(lines_cleared, 2) * Mathf.Max(1,
tetrisCombo) * scoreCombo * (int)(1 + (Speed - 1) * 2);

            Speed += (Mathf.Floor(scoreValue / (10 * (int)(1 + (Speed - 1) * 4))) -
Mathf.Floor(prevScoreValue / (10 * (int)(1 + (Speed - 1) * 4)))) / 4;

            if (Speed >= levelUpBoundary)
                levelUpBoundary = Speed + 1;
        }
    }
}

```

```
        NewBrick();

        coyoteTime = 0;
        updateTime = Time.time * Speed + 1;

        allowHold = true;
    }

}

else
{
    RenderDropPreview(brickType, activeX, activeY, activeRot, Tiles);
    coyoteTime = 0;
}

//Move the active block down by one
if (Time.time * Speed >= updateTime && coyoteTime == 0)
{
    updateTime = Time.time * Speed + 1;
    activeY++;
}

mainScreen.Apply();

}

public void NewBrick()
{
    //Yeah, more globals here :)
    autoMoveSpeed = defaultMoveSpeed;

    brickType = nextBricks[nextBrickIndex];
    if (nextBrickIndex % 7 == 0)
    {
        nextBricks = bankBricks;
        bankBricks = GetBrickOrder();
    }

    nextBrickIndex = (nextBrickIndex + 1) % 7;

    RenderUiElement(nextStartPos, nextBricks[nextBrickIndex]);

    activeX = 4;
    activeY = 0;
    activeRot = 0;
}

public (int[,], int) ClearFullLines(int[,] Tiles)
{
    int lines_cleared = 0;

    for (int line = 0; line < 20; line++)
    {
        bool clear_line = true;
        for (int x = 0; x < 10; x++)
        {
            if (Tiles[line, x] == 0)
            {
                clear_line = false;
            }
        }
        if (clear_line)
        {
            lines_cleared++;
            for (int x = 0; x < 10; x++)
            {
                Tiles[line, x] = 0;
            }
        }
    }
}
```

```

        break;
    }
}
if (clear_line)
{
    lines_cleared++;
    for (int y = line; y > 0; y--)
    {
        for (int x = 0; x < 10; x++)
            Tiles[y, x] = Tiles[y - 1, x];
    }
    for (int x = 0; x < 10; x++)
        Tiles[0, x] = 0;
    }
}
return (Tiles, lines_cleared);
}

public bool CheckGrounded(int[,] activeCoords)
{
    for (int coord = 0; coord < 4; coord++)
    {
        if (activeCoords[coord, 1] + 1 > 19 || Tiles[activeCoords[coord, 1] + 1,
activeCoords[coord, 0]] != 0)
            return true;
    }
    return false;
}

public bool CheckCollided(int brickType, int moveX, int moveY, int moveRot, int[,] Tiles)
{
    int[,] simulation = GetActiveCoords(brickType, moveX, moveY, moveRot);

    for (int coord = 0; coord < 4; coord++)
    {
        if (simulation[coord, 1] > 19)
            return true;
        else if (simulation[coord, 0] > 9)
            continue;
        else if (simulation[coord, 1] > 19 || Tiles[simulation[coord, 1],
simulation[coord, 0]] != 0)
            return true;
    }
    //Yeah I'm using a global variable here, I'm sorry but this saves too many
lines
    coyoteTime = Time.time * Speed + 1;
    return false;
}

public bool Overflow(int[,] simulation)
{
    for (int coord = 0; coord < 4; coord++)
    {
        if (simulation[coord, 0] > 9)
            return true;
    }
    return false;
}

public void RenderUiElement(Vector2Int startPos, int heldBrickType)
{

```

```

Color[] colours = new Color[] {
    new Color(0, 0, 0), //0 -> no brick here
    new Color(0, 1, 1), //1 -> cyan I
    new Color(0.5f, 0, 1), //2 -> purple T
    new Color(1, 0.5f, 0), //3 -> orange L
    new Color(0, 0, 1), //4 -> blue J
    new Color(1, 0, 0), //5 -> red Z
    new Color(0, 1, 0), //6 -> green S
    new Color(1, 1, 0) //7 -> yellow O
};

int[,] holdCoords = GetActiveCoords(heldBrickType, 0, 0, 0);

for (int y = 0; y < 4; y++)
{
    for (int x = 0; x < 4; x++)
    {
        for (int px = 0; px < pixelSize; px++)
        {
            for (int py = 0; py < pixelSize; py++)
            {
                mainScreen.SetPixel(startPos.x * pixelSize + x * pixelSize +
px,
                        startPos.y * pixelSize + 3 * pixelSize - y * pixelSize +
py, colours[0]);
            }
        }
    }
}

for (int i = 0; i < 4; i++)
{
    for (int px = 0; px < pixelSize; px++)
    {
        for (int py = 0; py < pixelSize; py++)
        {
            mainScreen.SetPixel(startPos.x * pixelSize + holdCoords[i, 0] *
pixelSize + px,
                    startPos.y * pixelSize + 3 * pixelSize - holdCoords[i, 1] *
pixelSize + py, colours[heldBrickType]);
        }
    }
}

public void RenderDropPreview(int brickType, int activeX, int previewY, int
activeRot, int[,] Tiles)
{
    int[,] activeCoords = GetActiveCoords(brickType, activeX, previewY,
activeRot);

    int moveY = previewY;

    while (moveY < 20 && !CheckCollided(brickType, activeX, moveY, activeRot,
Tiles))
        moveY++;
    previewY = moveY - 1;

    int[,] previewCoords = GetActiveCoords(brickType, activeX, previewY,
activeRot);

    for (int i = 0; i < 4; i++)

```

```

    {
        bool skip = false;
        for (int j = 0; j < 4; j++)
        {
            if ((previewCoords[i, 1], previewCoords[i, 0]) == (activeCoords[j, 1],
activeCoords[j, 0]))
            {
                skip = true;
                break;
            }
        }
        //if (!skip)
        //    transform.GetChild(10 * previewCoords[i, 1] + previewCoords[i,
0]).gameObject.GetComponent<SpriteRenderer>().color = new Color(0.2f, 0.2f, 0.2f);

        if (!skip)
        {
            for (int px = 0; px < pixelSize; px++)
            {
                for (int py = 0; py < pixelSize; py++)
                {
                    mainScreen.SetPixel(mainStartPos.x * pixelSize +
previewCoords[i, 0] * pixelSize + px,
                           mainStartPos.y * pixelSize + 19 * pixelSize -
previewCoords[i, 1] * pixelSize + py, new Color(0.2f, 0.2f, 0.2f));
                }
            }
        }
    }

    public void Render(int[,] Tiles)
    {
        Color[] colours = new Color[] {
            new Color(0, 0, 0),    //0 -> no brick here
            new Color(0, 1, 1),    //1 -> cyan I
            new Color(0.5f, 0, 1), //2 -> purple T
            new Color(1, 0.5f, 0), //3 -> orange L
            new Color(0, 0, 1),    //4 -> blue J
            new Color(1, 0, 0),    //5 -> red Z
            new Color(0, 1, 0),    //6 -> green S
            new Color(1, 1, 0)     //7 -> yellow O
        };

        for (int y = 0; y < 20; y++)
        {
            for (int x = 0; x < 10; x++)
            {
                for (int px = 0; px < pixelSize; px++)
                {
                    for (int py = 0; py < pixelSize; py++)
                    {
                        mainScreen.SetPixel(mainStartPos.x * pixelSize + x * pixelSize
+ px,
                               mainStartPos.y * pixelSize + 19 * pixelSize - y *
pixelSize + py, colours[Tiles[y, x]]);
                    }
                }
            }
        }
    }
}

```

```
public int[,] GetActiveCoords(int brickType, int x, int y, int rot)
{
    //cyan I
    if (brickType == 1)
    {
        if (rot % 2 == 0)
        {
            activeCoords = new int[4, 2] {
                {x + 1, y},
                {x + 1, y + 1},
                {x + 1, y + 2},
                {x + 1, y + 3}
            };
        }
        else
        {
            activeCoords = new int[4, 2] {
                {x, y + 1},
                {x + 1, y + 1},
                {x + 2, y + 1},
                {x + 3, y + 1}
            };
        }
    }

    //purple T
    else if (brickType == 2)
    {
        if (rot == 0)
        {
            activeCoords = new int[4, 2] {
                {x, y + 1},
                {x + 1, y + 1},
                {x + 2, y + 1},
                {x + 1, y + 2}
            };
        }
        else if (rot == 1)
        {
            activeCoords = new int[4, 2] {
                {x + 1, y},
                {x, y + 1},
                {x + 1, y + 1},
                {x + 1, y + 2}
            };
        }
        else if (rot == 2)
        {
            activeCoords = new int[4, 2] {
                {x + 1, y},
                {x, y + 1},
                {x + 1, y + 1},
                {x + 2, y + 1}
            };
        }
        else
        {
            activeCoords = new int[4, 2] {
                {x + 1, y},
                {x + 1, y + 1},
                {x + 2, y + 1},
                {x + 1, y + 2}
            };
        }
    }
}
```

```
        };
    }

//orange L
else if (brickType == 3)
{
    if (rot == 0)
    {
        activeCoords = new int[4, 2] {
            {x, y},
            {x + 1, y},
            {x + 2, y},
            {x, y + 1}
        };
    }
    else if (rot == 1)
    {
        activeCoords = new int[4, 2] {
            {x, y},
            {x + 1, y},
            {x + 1, y + 1},
            {x + 1, y + 2}
        };
    }
    else if (rot == 2)
    {
        activeCoords = new int[4, 2] {
            {x + 2, y},
            {x, y + 1},
            {x + 1, y + 1},
            {x + 2, y + 1}
        };
    }
    else
    {
        activeCoords = new int[4, 2] {
            {x, y},
            {x, y + 1},
            {x, y + 2},
            {x + 1, y + 2}
        };
    }
}

//blue J
else if (brickType == 4)
{
    if (rot == 0)
    {
        activeCoords = new int[4, 2] {
            {x, y},
            {x + 1, y},
            {x + 2, y},
            {x + 2, y + 1}
        };
    }
    else if (rot == 1)
    {
        activeCoords = new int[4, 2] {
            {x + 1, y},
            {x + 1, y + 1},
            {x + 2, y + 1}
        };
    }
}
```

```
        {x, y + 2},
        {x + 1, y + 2}
    };
}
else if (rot == 2)
{
    activeCoords = new int[4, 2] {
        {x, y},
        {x, y + 1},
        {x + 1, y + 1},
        {x + 2, y + 1}
    };
}
else
{
    activeCoords = new int[4, 2] {
        {x, y},
        {x + 1, y},
        {x, y + 1},
        {x, y + 2}
    };
}
}

//red Z
else if (brickType == 5)
{
    if (rot % 2 == 0)
    {
        activeCoords = new int[4, 2] {
            {x, y},
            {x + 1, y},
            {x + 1, y + 1},
            {x + 2, y + 1}
        };
    }
    else
    {
        activeCoords = new int[4, 2] {
            {x + 1, y},
            {x, y + 1},
            {x + 1, y + 1},
            {x, y + 2}
        };
    }
}

//green S
else if (brickType == 6)
{
    if (rot % 2 == 0)
    {
        activeCoords = new int[4, 2] {
            {x + 1, y},
            {x + 2, y},
            {x, y + 1},
            {x + 1, y + 1}
        };
    }
    else
    {
        activeCoords = new int[4, 2] {
```

```
        {x, y},
        {x, y + 1},
        {x + 1, y + 1},
        {x + 1, y + 2}
    };
}
}

//yellow 0
else
{
    activeCoords = new int[4, 2] {
        {x, y},
        {x + 1, y},
        {x, y + 1},
        {x + 1, y + 1}
    };
}

return activeCoords;
}

public int[,] SetCoords(int brickType, int[,] Coords, int[,] Tiles)
{
    for (int coord = 0; coord < 4; coord++)
        Tiles[Coords[coord, 1], Coords[coord, 0]] = brickType;

    return Tiles;
}

public int[] GetBrickOrder()
{
    int[] nextBricks = new int[7] { 0, 0, 0, 0, 0, 0, 0 };
    for (int i = 0; i < 7; i++)
    {
        int nextBrick = 0;
        bool unique = false;
        while (!unique)
        {
            nextBrick = Random.Range(1, 8);
            unique = true;
            foreach (int brick in nextBricks)
            {
                if (nextBrick == brick)
                {
                    unique = false;
                    break;
                }
            }
        }
        nextBricks[i] = nextBrick;
    }
    return nextBricks;
}
}
```

TextRender.cs

```

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

public class TextRender : MonoBehaviour
{
    [SerializeField] private GameObject prefabLetter;
    [SerializeField] private Color bgCol = new Color(0,0,0,0);
    List<Dialogue.Letter[]> letters;
    int sentence = 0;

    // Start is called before the first frame update
    void Start()
    {
        prefabLetter.SetActive(false);
        SetBgCol();
    }

    public void LoadConversation(string filename)
    {
        Dialogue dialogue = Dialogue.Load(filename);
        letters = dialogue.GetLetters();
        sentence = 0;
    }

    public bool NextSentence()
    {
        return RenderSentence(sentence++);
    }

    bool RenderSentence(int index)
    {
        for (int i = 1; i < transform.childCount; i++)
        {
            transform.GetChild(i).GetComponent<LetterEffect>().FadeOut();
        }
        if (index >= letters.Count)
            return false;

        float xOffset = 0;

        foreach (Dialogue.Letter letter in letters[index])
        {
            SadGirl.faceLight01 = Mathf.Clamp01(letter.freqSpeedAmpAmp.w / 0.05f);

            GameObject letterObj = Instantiate(prefabLetter, transform.position + xOffset
* transform.right, transform.rotation, transform);
            letterObj.SetActive(true);
            TextMesh letterMesh = letterObj.GetComponent<TextMesh>();
            letterMesh.text = letter.character;
            // 'o' has a reasonable width to use in the case of empty characters

            letterObj.GetComponent<TextMesh>().font.GetCharacterInfo(letter.character.Length > 0 ?
letter.character[0] : 'o', out CharacterInfo info, letterMesh.fontSize,
letterMesh.fontStyle);
            xOffset += info.advance * letterMesh.characterSize * 0.1f;
        }
    }
}

```

```
letterObj.GetComponent<LetterEffect>().Init(letter, xOffset, xOffset / 5);
// 2 + xOffset / 5
}

transform.GetChild(0).localScale = new Vector3(xOffset, 1, 1);

return true;
}

void SetBgCol()
{
    transform.GetChild(0).GetChild(0).GetComponent<MeshRenderer>().sharedMaterial.SetColor
("BaseColor", bgCol);
}
}
```

Tile.cs

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

public class Tile : MonoBehaviour
{
    public Vector3Int parentKey { get; private set; }

    private Vector2 tileSize;
    [SerializeField] private GameObject[] WallPrefabs;

    [SerializeField] private GameObject Shelf;
    [SerializeField] private GameObject Light;
    [SerializeField] private GameObject Door;
    [SerializeField] private GameObject[] OtherObjects;

    public Dictionary<Vector3Int, Wall> Walls;

    public void Init(Vector3Int pos, Vector3Int _parentKey, Vector2 _tileSize)
    {
        parentKey = _parentKey;
        tileSize = _tileSize;

        foreach (GameObject prefab in WallPrefabs)
            prefab.transform.localScale = new Vector3(tileSize.x, tileSize.y * 2,
tileSize.x);

        Walls = new Dictionary<Vector3Int, Wall>(5)
        {
            { Vector3Int.right, new Wall(Vector3Int.right, WallPrefabs[0], Shelf,
OtherObjects) },
            { Vector3Int.left, new Wall(Vector3Int.left, WallPrefabs[1], Shelf,
OtherObjects) },
            { Vector3Int.forward, new Wall(Vector3Int.forward, WallPrefabs[2], Shelf,
OtherObjects) },
            { Vector3Int.back, new Wall(Vector3Int.back, WallPrefabs[3], Shelf,
OtherObjects) },
            { Vector3Int.down, new Wall(Vector3Int.down, WallPrefabs[4], Shelf,
OtherObjects) }
        };

        transform.localPosition = new Vector3(pos.x * tileSize.x, pos.y * tileSize.y,
pos.z * tileSize.x);
    }

    public void Render()
    {
        Transform root = new GameObject("Tile root").transform;
        root.parent = transform;
        root.localPosition = Vector3.zero;

        if (tileSize == Vector2.zero)
        {
            Debug.LogError("Error: Tile not initialised");
            return;
        }
    }
}
```

```

        bool corridor = false;
        Vector3Int opposite = Vector3Int.zero;
        foreach (Vector3Int key in Walls.Keys)
        {
            if (!Walls[key].wallState.removed)
            {
                if (key == -opposite)
                {
                    corridor = true;
                    break;
                }
                else
                    opposite = key;
            }
        }

        //Add a door if the tile has two opposing walls
        if (Random.value < 0.3f && corridor)
        {
            Vector3Int[] dirs = new Vector3Int[4] { Vector3Int.forward,
Vector3Int.back, Vector3Int.left, Vector3Int.right };
            Vector3Int dir = dirs[Random.Range(0, dirs.Length)];

            //Locked if removing wall would make a hole in the hive
            bool locked = !Walls[dir].wallState.removed;

            //Replace wall with door

            Walls[dir].wallState.SetState(Wall.State.removed);

            Quaternion rot = Quaternion.Euler(0, 180, 0);

            if (dir == Vector3Int.back)
                rot = Quaternion.Euler(0, 0, 0);
            else if (dir == Vector3Int.left)
                rot = Quaternion.Euler(0, 90, 0);
            else if (dir == Vector3Int.right)
                rot = Quaternion.Euler(0, -90, 0);

            GameObject door = Instantiate(Door, transform.position, rot, root);
            door.GetComponent<OpenDoor>().locked = locked;
        }

        foreach (Wall wall in Walls.Values)
            wall.Render(root);

        if (Random.value < 0.1f)
            Instantiate(Light, transform.position, transform.rotation, root);
    }

    private IEnumerator showIfClose;

    private IEnumerator ShowIfClose(float time)
    {
        while (true)
        {
            yield return new WaitForSeconds(time);
            if (transform.childCount > 0)

```

```
transform.GetChild(0).gameObject.SetActive((Camera.main.transform.position -  
transform.position).sqrMagnitude < 2500);  
    }  
}  
public void Start()  
{  
    showIfClose = ShowIfClose(0.5f);  
    StartCoroutine(showIfClose);  
}  
public void OnDestroy()  
{  
    if (showIfClose != null)  
        StopCoroutine(showIfClose);  
}  
}
```

TowerGen.cs

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

#if UNITY_EDITOR
[ExecuteAlways]
#endif
public class TowerGen : MonoBehaviour
{
    public Vector3 normalUp = Vector3.up;
    [SerializeField] private GameObject basePrefab;
    [SerializeField] private GameObject flatPrefab;
    [SerializeField] private GameObject[] topPrefabs;
    private GameObject topPrefab;
    [SerializeField] private GameObject shelfPrefab;
    [SerializeField] private GameObject ceilLightPrefab;
    [SerializeField] private GameObject[] otherObjects;
    [Space()]
    [SerializeField] private int stories = 5;
    [SerializeField] private int startRot = int.MaxValue;

    private List<Flat> flats;

    public bool makeOnStart = false;

    [HideInInspector] public bool debugMake = false;

    private void Start()
    {
        debugMake = false;
        if (makeOnStart)
            Init(Random.Range(0, 2000));
    }

    public void Init(int seed)
    {
        normalUp = normalUp.normalized;
        GenerateTower(seed);
        makeOnStart = false;
    }

    public void GenerateTower(int seed)
    {
        Random.InitState(seed);
        stories = Random.Range(5, 15);
        topPrefab = stories < 10 ? topPrefabs[0] : topPrefabs[Random.Range(0, topPrefabs.Length)];
        if (transform.childCount == 1)
        {
            transform.GetChild(0).gameObject.SetActive(false);
            if (Application.isEditor)
                DestroyImmediate(transform.GetChild(0).gameObject);
            else
                Destroy(transform.GetChild(0).gameObject);
        }
        GameObject holder = new GameObject("Holder");
        holder.transform.parent = transform;
        holder.transform.localPosition = Vector3.zero;
```

```

    //Check if tower is on top of another tower (like the 3x3 on the 5x5 top
prefab)
    foreach (TowerGen towerGen in FindObjectsOfType<TowerGen>())
    {
        if (transform.IsChildOf(towerGen.transform))
            normalUp = towerGen.normalUp;
    }

    //transform.rotation = Quaternion.LookRotation(Vector3.Cross(normalUp,
Vector3.forward), normalUp);

    int prevRot = startRot;

    flats = new List<Flat>();

    for (int i = 0; i < stories; i++)
    {
        int rot;
        do { rot = Random.Range(0, 4) * 90; }
        while (rot == prevRot);

        GameObject prefab = i == 0 ? basePrefab : i == stories - 1 ? topPrefab :
flatPrefab;

        Flat flat = Instantiate(prefab, transform.position + i * 5 * normalUp,
Quaternion.LookRotation(Vector3.Cross(normalUp, Vector3.forward), normalUp) *
Quaternion.Euler(0, rot, 0), holder.transform).GetComponent<Flat>();
        flat.gameObject.layer = gameObject.layer;
        flat.Init(Random.Range(0, 2000), rot, i == 0 ? transform : flats[i -
1].nextGapPos);

        flats.Add(flat);

        prevRot = rot;
    }
    if (!debugMake)
        return;
    foreach (Flat flat in flats)
        flat.DebugMake();
}

public void DebugRealign()
{
    transform.rotation = Quaternion.identity;
}
}

```

TreeGen.cs

```

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

public class TreeGen : MonoBehaviour
{
    public GameObject BranchPrefab { get; private set; }
    [SerializeField] private Material leafMatPreset;

    public GameObject root { get; private set; }

    [Range(0, 20)]
    [SerializeField]
    private float consecutiveSplits;
    [SerializeField]
    private float trunkLength;
    [SerializeField]
    private float minBranchLength;
    [SerializeField]
    private float minBranchWidth;
    [SerializeField]
    private int leafDensity;
    public Vector3 planetNormal;
    public Vector3 relativePosition;

    private float sqrMaxNatureDist;

    private List<Branch> branches;

    public void SetMaterials(Planet planet, Vector3 _relativePosition)
    {
        sqrMaxNatureDist = planet.maxNatureDist * planet.maxNatureDist;
        relativePosition = _relativePosition;

        BranchPrefab = transform.GetChild(0).GetChild(0).gameObject;
        Material branchMat = new
Material(BranchPrefab.transform.GetChild(0).GetComponent<MeshRenderer>().sharedMaterial);
        branchMat.SetFloat("_minElevation", planet.planetMesh.elevationData.Min);
        branchMat.SetFloat("_maxElevation", planet.planetMesh.elevationData.Max);
        branchMat.SetTexture("_planetTexture",
planet.terrainMat.GetTexture("_planetTexture"));
        branchMat.SetVector("_position", relativePosition);

        BranchPrefab.transform.GetChild(0).GetComponent<MeshRenderer>().sharedMaterial =
branchMat;

        Material leafMat = new Material(leafMatPreset);
        leafMat.SetFloat("_minElevation", planet.planetMesh.elevationData.Min);
        leafMat.SetFloat("_maxElevation", planet.planetMesh.elevationData.Max);
        leafMat.SetTexture("_planetTexture",
planet.terrainMat.GetTexture("_planetTexture"));
        leafMat.SetVector("_position", transform.position);
        leafMat.SetFloat("_windDensity", planet.planetValues.windSpeed * 0.06f);

        for (int i = 0; i < BranchPrefab.transform.GetChild(1).childCount; i++)
    }
}

```

```

BranchPrefab.transform.GetChild(1).GetChild(i).GetChild(0).GetComponent<MeshRenderer>()
).sharedMaterial = leafMat;
}

public void Init(int _layer, int seed)
{
    BranchPrefab = transform.GetChild(0).GetChild(0).gameObject;

    gameObject.layer = _layer;
    BranchPrefab.transform.gameObject.layer = _layer;
    for (int i = 0; i < BranchPrefab.transform.GetChild(1).childCount; i++)

BranchPrefab.transform.GetChild(1).GetChild(i).GetChild(0).gameObject.layer = _layer;

    Random.InitState(seed);

    Randomise();
    GenerateTree();
    //RenderTree();

    Random.InitState(System.Environment.TickCount);
}
private void Randomise()
{
    consecutiveSplits = Random.Range(10, 15);
    minBranchLength = Random.Range(0.6f, 1.4f);
    trunkLength = Random.Range(4f, 8);
    minBranchWidth = Random.Range(0.001f, 0.02f);

    leafDensity = 0;
}
private void GenerateTree()
{
    if (transform.childCount > 1)
    {
        if (Application.isEditor)
            DestroyImmediate(transform.GetChild(1).gameObject);
        else
            Destroy(transform.GetChild(1).gameObject);
    }

    root = new GameObject("Root (1)");
    root.layer = gameObject.layer;
    root.transform.position = transform.position;
    root.transform.parent = transform;
    Branch.treeObj = this;
    Branch.trunkLength = trunkLength;
    Branch.planetNormal = planetNormal;
    Branch.leafDensity = leafDensity;

    Vector2 fullRange = new Vector2(-Mathf.PI, Mathf.PI);

    branches = new List<Branch>();

    Branch trunk = new Branch(Vector3.zero, trunkLength, 0.5f,
Branch.planetNormal, fullRange, 0);
    branches.Add(trunk);
    int k = 0;
    //While there are still branches generated left to split
    while (k < branches.Count)
}

```

```

    {
        float angleBuffer = Random.Range(-Mathf.PI, Mathf.PI);
        for (int i = 0; i < branches[k].splitsInto; i++)
        {
            //Break if the branch is too small to split or has reached the split
            depth limit
            if (branches[k].branchWidth / branches[k].splitsInto < minBranchWidth)
                break;
            else if (branches[k].consecutiveSplits > consecutiveSplits)
                break;

            Vector3 splitPoint;
            if (i == 0)
                splitPoint = branches[k].endPoint;
            else
                splitPoint = Vector3.Lerp(branches[k].startPoint,
branches[k].endPoint, Random.Range(Random.Range(0.25f, 0.5f), 1));

            float branchWidth = branches[k].branchWidth / branches[k].splitsInto;
            float branchLength = Random.Range(minBranchLength,
branches[k].branchLength);

            Vector2 directionRange = new Vector2(angleBuffer + 2 * Mathf.PI * i /
branches[k].splitsInto, 0);
            directionRange.y = directionRange.x + 2 * Mathf.PI /
branches[k].splitsInto;
            Branch branch = new Branch(splitPoint, branchLength, branchWidth,
branches[k].branchNormal, directionRange, branches[k].consecutiveSplits);
            branches.Add(branch);
        }
        k++;
    }
}

public void RenderTree()
{
    foreach (Branch branch in branches)
    {
        branch.ShowBranch();
        //branch.AddLeaves();
    }
}

private IEnumerator renderatLOD;

private IEnumerator RenderAtLOD(float time)
{
    while (true)
    {
        yield return new WaitForSeconds(time);

        float LOD = Mathf.InverseLerp(0, sqrMaxNatureDist, (transform.position -
Camera.main.transform.position).sqrMagnitude);

        foreach (Branch branch in branches)
        {
            int splits = branch.consecutiveSplits;
            //Show entire tree if close enough, or the trunk and leaf filled
            branches if a bit further away
            if (LOD < 0.2f || ((splits < 2 || splits > consecutiveSplits * 0.75f) &&
&& LOD < 0.95f))

```

```
        {
            branch.ShowBranch();
            //branch.AddLeaves();
        }
        else
        {
            branch.HideBranch();
        }
    }
}
void Start()
{
    renderatLOD = RenderAtLOD(0.5f);
    StartCoroutine(renderatLOD);
    //InvokeRepeating("RenderAtLOD", Random.value, 0.5f);
}
void OnDisable()
{
    StopCoroutine(renderatLOD);
}
void OnEnable()
{
    renderatLOD = RenderAtLOD(0.5f);
    StartCoroutine(renderatLOD);
    //InvokeRepeating("RenderAtLOD", Random.value, 0.5f);
}
}
```

TriggerHiveGen.cs

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

public class TriggerHiveGen : MonoBehaviour
{
    [SerializeField] private ShipDoorOpen trigger;
    [SerializeField] private GameObject hivePrefab;
    private bool comeFromHive = false;

    void Start()
    {
        CameraState.inHive = false;
    }

    void LateUpdate()
    {
        //SetPlanetActive(!CameraState.inHive);

        if (CameraState.inHive)
            return;

        //If the player is on the planet and the enterence door is closed
        if (!comeFromHive && trigger.lerp < 0)
        {
            CameraState.inHive = true;

            Instantiate(hivePrefab, transform.GetChild(3).position, transform.rotation,
            transform.root.GetChild(6));

            comeFromHive = true;
        }
        //If the player was in the hive but isn't now and hive has been generated
        else if (comeFromHive && transform.root.GetChild(6).childCount > 0)
        {
            Destroy(transform.root.GetChild(6).GetChild(0).gameObject);
        }
        //If the player was in the hive but isn't now and the enterence door is open
        else if (comeFromHive && trigger.lerp > 0)
            comeFromHive = false;
    }
}
```

UpdateStars.compute

```

// Each #kernel tells which function to compile; you can have many kernels
#pragma kernel UpdateStars

// Create a RenderTexture with enableRandomWrite flag and set it
// with cs.SetTexture
StructuredBuffer<float3> starPosRelativeToGalaxy
StructuredBuffer<float> starSphereRadii
RWStructuredBuffer<float3> starRenderPos
RWStructuredBuffer<float> starRenderLocalScaleX

float numStars
float3 galaxyCentre
float minAt1Dist
float renderDistance
float4x4 cameraMatrix

float GetCircleStarScale float starRadius float sqrStarDist
    float horizonRadius = max minAt1Dist sqrt sqrStarDist - starRadius * starRadius *
starRadius / sqrStarDist

    return 2 * horizonRadius * renderDistance

numthreads 128 1 1
void UpdateStars uint3 id : SV_DispatchThreadID

// TODO: insert actual code here!
if id.x >= numStars
    return

float close01 = clamp starRenderLocalScaleX id.x / 40000 0 1

//maybe overoptimisation? it was important for CPU...
//float3 viewPoint = mul(cameraMatrix, float4(starRenderPos[id.x], 1));
//if (viewPoint.x < 0 || viewPoint.x > 1 || viewPoint.y < 0 || viewPoint.y > 1 ||
viewPoint.z < 0)
//    return;

float3 starPosRelativeToCamera = starPosRelativeToGalaxy id.x + galaxyCentre
float sqrDistance = starPosRelativeToCamera.x * starPosRelativeToCamera.x +
starPosRelativeToCamera.y * starPosRelativeToCamera.y + starPosRelativeToCamera.z *
starPosRelativeToCamera.z

//Radius of the cross section of the 3D sphere (at the star's position) tangent to
the player's view
float horizonRadius = max minAt1Dist sqrt sqrDistance - starSphereRadii id.x *
starSphereRadii id.x * starSphereRadii id.x / sqrDistance

starRenderLocalScaleX id.x = 2 * horizonRadius * renderDistance - 1000 * close01
//Closer stars are rendered in front of further stars
starRenderPos id.x = normalize starPosRelativeToCamera * renderDistance - 1000 *
close01

```

Wall.cs

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

public class Wall
{
    public class StateMachine
    {
        public State state { get; private set; }

        public void SetState(State _state)
        {
            if (!removed)
                state = _state;
        }

        public bool removed { get { return state == State.removed; } }

        public StateMachine(State _value)
        {
            state = _value;
        }
    }

    private GameObject Prefab;
    private GameObject Shelf;
    private GameObject[] Objects;

    public enum State { visible, removed }

    public StateMachine wallState;

    public Vector3Int type { get; private set; }

    public Wall(Vector3Int _type, GameObject _Prefab, GameObject _Shelf, GameObject[]
    _Objects)
    {
        type = _type;
        Prefab = _Prefab;
        Shelf = _Shelf;
        Objects = _Objects;
        wallState = new StateMachine(State.visible);
    }

    public void Render(Transform parent)
    {
        if (wallState.removed)
            return;

        Transform wallObject = Object.Instantiate(Prefab, parent.position,
        parent.rotation * Prefab.transform.localRotation, parent).transform;

        wallObject.localScale = Prefab.transform.localScale;

        //wallObject.GetChild(0).GetComponent<MeshRenderer>().sharedMaterial = new
        Material(wallObject.GetChild(0).GetComponent<MeshRenderer>().sharedMaterial);
    }
}
```

```

//wallObject.GetChild(0).GetComponent<MeshRenderer>().sharedMaterial.SetColor("_tint",
wallState.locked ? Color.red : Color.gray);

    if (Random.value < 0.3f)
        SpawnObject(parent);
}

public void SpawnObject(Transform parent)
{
    Quaternion rot = Quaternion.Euler(0, 0, 0);

    if (type == Vector3Int.back)
        rot = Quaternion.Euler(0, 180, 0);
    else if (type == Vector3Int.left)
        rot = Quaternion.Euler(0, -90, 0);
    else if (type == Vector3Int.right)
        rot = Quaternion.Euler(0, 90, 0);
    else if (type != Vector3Int.forward)
        return; //no shelves on the floor

    if (Random.value < 0.6f)
    {
        //Spawn a random object
        Transform randObject = Object.Instantiate(Objects[Random.Range(0,
Objects.Length)], parent.position, Quaternion.identity, parent).transform;
        randObject.localRotation = rot;
    }
    else
    {
        //Spawn a shelf
        bool lowest = false;

        for (int i = 0; i < 2; i++)
        {
            if (Random.value < 0.5f || i == 1)
            {
                if (i == 0)
                    lowest = true;
                else if (i == 1)
                    lowest = !lowest;

                Transform shelf = Object.Instantiate(Shelf, parent.position +
Vector3.up * i, Quaternion.identity, parent).transform;

                shelf.localRotation = rot;
                //shelf.localPosition += 5 * (1 - scale) / 2 * (Random.value * 2 -
1) * shelf.right;

                shelf.GetComponent<Shelf>().lowest = lowest;
                shelf.GetComponent<Shelf>().height = i;
                shelf.GetComponent<Shelf>().Init();
            }
        }
    }
}
}

```

Weight.cs

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

public class Weight : MonoBehaviour
{
    //The sigWeight is the PLANET closest (most significant) to this weight, this is
    //important information for many scripts
    [HideInInspector] public Weight sigWeight = null;

    public static float gConst = 30f;
    public static int framesPerUpdate = 60;

    [HideInInspector] public Planet planet;
    protected Rigidbody rb;
    public Vector3 position { get { return rb.position; } set { rb.position = value; } }

    [SerializeField] protected float Mass;
    private void OnValidate() { Mass = Mathf.Max(1, Mass); }

    public virtual float mass { get { return Mass; } }
    public Vector3 initialVelocity;

    public Vector3 velocity;
    private Vector3 acceleration;

    private int counter = -1;

    public bool stationary = false;

    [Space]
    [Header("Simulation Settings")]
    public Color colour;
    [Range(0, 300)]
    public float secondsAhead;
    [SerializeField] private Weight relativeTo;
    private Vector3 simVelocity;
    private Vector3 simAcceleration;
    [HideInInspector] public List<Vector3> positionOverTime;

    public static float MassForSurfaceG(float g, float radius)
    {
        return g * radius / gConst;
    }

    public void Init()
    {
        sigWeight = null;
        velocity = initialVelocity;
        ConfigRigidbody();
        TryGetComponent(out planet);
    }

    public void SetMass(float g, float radius)
    {
        Mass = MassForSurfaceG(g, radius);
    }
```

```

private void ConfigRigidbody()
{
    if (!TryGetComponent(out rb))
        rb = gameObject.AddComponent< Rigidbody >();
    rb.useGravity = false;
    rb.isKinematic = true;
    rb.interpolation = RigidbodyInterpolation.Interpolate;
    //rb.freezeRotation = true;
}

public void GetGravity(out Vector3 displacement)
{
    velocity += GetAcceleration() * Time.fixedDeltaTime;

    displacement = velocity * Time.fixedDeltaTime;
}

public void Teleport(Vector3 displacement, bool force = false)
{
    if (stationary && !force)
        return;
    position += displacement;
}

private Vector3 GetAcceleration()
{
    counter++;
    if (counter % framesPerUpdate != 0)
        return acceleration;

    acceleration = Vector3.zero;

    float minSqrDist = 25_000_000; //5000^2 (maximum distance of rings)

    sigWeight = null;

    //Add acceleration due to gravity towards each active weight
    foreach (Weight weight in PhysicsUpdate.weights)
    {
        if (weight == this || weight.mass == 0)
            continue;

        Vector3 acc = gConst * weight.mass / (weight.position -
position).sqrMagnitude * (weight.position - position); //directly proportional to
distance (since 1 normalizes vector)
        acceleration += acc;

        if (weight.planet != null && (weight.position - position).sqrMagnitude <
minSqrDist)
        {
            minSqrDist = (weight.position - position).sqrMagnitude;
            sigWeight = weight;
        }
    }
    return acceleration;
}

public Vector3 GetAccelerationAhead(int frame, List< Weight > weights)
{
    Vector3 acceleration = Vector3.zero;
    foreach (Weight weight in weights)

```

```

    {
        if (weight == this)
            continue;
        acceleration += gConst * weight.mass / (weight.positionOverTime[frame] -
positionOverTime[frame]).magnitude * (weight.positionOverTime[frame] -
positionOverTime[frame]).normalized;
    }
    return acceleration;
}

public void Simulate()
{
    if (relativeTo == null)
        relativeTo = this;

    int frames = Mathf.RoundToInt(secondsAhead / Time.fixedDeltaTime);

    List<Weight> sigWeights = new List<Weight>();

    foreach (Weight weight in FindObjectsOfType<Weight>())
    {
        if (weight.mass == 0)
            continue;

        weight.positionOverTime = new List<Vector3>(frames);
        weight.positionOverTime.Add(weight.transform.position);
        weight.simVelocity = Application.isPlaying ? weight.velocity :
weight.initialVelocity;

        sigWeights.Add(weight);
    }

    for (int i = 0; i < frames; i++)
    {
        foreach (Weight weight in sigWeights)
        {
            if (i % framesPerUpdate == 0)
                weight.simAcceleration = weight.GetAccelerationAhead(i,
sigWeights);

            weight.simVelocity += weight.simAcceleration * Time.fixedDeltaTime;
            weight.positionOverTime.Add(weight.positionOverTime[i] +
(weight.stationary ? Vector3.zero : weight.simVelocity * Time.fixedDeltaTime));
        }
    }
#endif UNITY_EDITOR
    if (Application.isEditor)
    {
        foreach (Weight weight in sigWeights)
        {
            List<Vector3> lines = new List<Vector3>();
            for (int i = 0; i < frames; i += framesPerUpdate)
                lines.Add(relativeTo.transform.position -
relativeTo.positionOverTime[i] + weight.positionOverTime[i]);
            Handles.color = weight.colour;
            Handles.DrawPolyLine(lines.ToArray());
        }
    }
#endif
}
}

```

ZeroWeight.cs

```

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

public class ZeroWeight : Weight
{
    //Zero weight for a reason
    private void OnValidate() { Mass = 0; }
    public override float mass { get { return 0; } }

    [Space]
    [Header("Collision Detection Settings")]
    public CollisionDetection kinematicBody;
    public enum CapsuleAxis { xAxis, yAxis, zAxis }
    public CapsuleAxis capsuleAxis;
    public bool isShip { get; protected set; } = false;

    public virtual void Start()
    {
        kinematicBody.Init(this);
    }

    public virtual Vector3 MoveRelative(Vector3 displacement)
    {
        Vector3 motion = displacement / Time.fixedDeltaTime;
        kinematicBody.GetSafeMotion(motion, out Vector3 safeDisplacement, out _);

        if (CameraState.inHive && !isShip)
            sigWeight.transform.GetChild(6).position -= safeDisplacement;
        else
            Teleport(safeDisplacement);

        //We need friction as otherwise velocity will grow too large and 0weight
        //will clip, since we can't add advisedImpulse without losing control of the 0weight.
        if (kinematicBody.isGrounded && sigWeight != null)
            velocity = Vector3.Lerp(velocity, sigWeight.velocity, 0.1f);

        return safeDisplacement;
    }

    [System.Serializable]
    public class CollisionDetection
    {
        public float slopeLimit = 70;
        public float stepOffset = 0.3f;
        public float skinWidth = 0.02f;

        [SerializeField] public Vector3 m_center = Vector3.zero;
        [SerializeField] public float m_radius = 0.7f;
        [SerializeField] public float m_height = 3.23f;
        [SerializeField] private FabrikLeg[] legs;

        private Vector3 m_position;
        private Vector3 m_upDirection;
        private CapsuleAxis axis;
        private bool isShip = false;

        private ZeroWeight body;
    }
}

```

```

private CapsuleCollider collider;

private readonly Collider[] m_overlaps = new Collider[5];
private readonly List<RaycastHit> m_contacts = new List<RaycastHit>();

private const int MaxSweepSteps = 5;
private const float MinMoveDistance = 0f;
private const float MinCeilingAngle = 145;

public Vector3 velocity { get; set; }
public bool isGrounded { get; private set; }
public Vector3 center
{
    get { return m_center; }
    set
    {
        m_center = value;
        collider.center = value;
    }
}
public float radius
{
    get { return m_radius; }
    set
    {
        m_radius = value;
        collider.radius = value;
    }
}
public float height
{
    get { return m_height; }
    set
    {
        m_height = value;
        collider.height = value;
    }
}

public void Init(ZeroWeight _thisWeight)
{
    body = _thisWeight;
    isShip = body.isShip;
    axis = body.capsuleAxis;

    if (!body.TryGetComponent(out collider))
        collider =
body.gameObject.AddComponent<CapsuleCollider>();

    collider.center = m_center;
    collider.height = m_height;
    collider.radius = m_radius + Physics.defaultContactOffset;
    collider.direction = (int)axis;
}

public void GetSafeMotion(Vector3 motion, out Vector3 safeDisplacement,
out Vector3 advisedImpulse)
{
    m_position = body.position;
    m_upDirection = axis == CapsuleAxis.xAxis ? body.transform.right
: axis == CapsuleAxis.yAxis ? body.transform.up : body.transform.forward;
    velocity = motion;
}

```

```

m_contacts.Clear();
isGrounded = false;

//Calculate a safe displacement to move by
if (velocity.sqrMagnitude > MinMoveDistance)
{
    if (isShip)
    {
        CapsuleSweep(velocity.normalized,
velocity.magnitude * Time.fixedDeltaTime, 0);
    }
    else
    {
        Vector3 localVelocity =
body.transform.InverseTransformDirection(velocity);
        Vector3 lateralVelocity = new
Vector3(localVelocity.x, 0, localVelocity.z);
        Vector3 verticalVelocity = new Vector3(0,
localVelocity.y, 0);

        lateralVelocity =
body.transform.TransformDirection(lateralVelocity) * Time.fixedDeltaTime;
        verticalVelocity =
body.transform.TransformDirection(verticalVelocity) * Time.fixedDeltaTime;

        CapsuleSweep(lateralVelocity.normalized,
lateralVelocity.magnitude, stepOffset, MinCeilingAngle);
        CapsuleSweep(verticalVelocity.normalized,
verticalVelocity.magnitude, 0, 0, slopeLimit);
    }
}

//Adjust velocity here as a whole vector
if (m_contacts.Count > 0)
{
    foreach (RaycastHit contact in m_contacts)
    {
        float angle = Vector3.Angle(m_upDirection,
contact.normal);

        if (angle <= slopeLimit || isShip)
            isGrounded = true;

        if (isShip)

((ShipWeight)body).ApplyNormal(contact.normal);

        velocity -= Vector3.Project(velocity,
contact.normal);
    }
}

//Fix any overlaps (these cannot be fixed by CapsuleSweep)

///START OF NOT MY CODE

float capsuleOffset = m_height * 0.5f - m_radius;
Vector3 top = m_position + m_upDirection * capsuleOffset;
Vector3 bottom = m_position - m_upDirection * capsuleOffset;
int overlapsNum = Physics.OverlapCapsuleNonAlloc(top, bottom,
collider.radius, m_overlaps);

```

```

        if (overlapsNum > 0)
        {
            for (int i = 0; i < overlapsNum; i++)
            {
                //Don't want the ship capsule to detect collision
                //with its individual colliders or the player capsule (as then the ship would displace
                //away from the player and itself)
                //Don't want the player capsule to detect collision
                //with the ship capsule (as then the player couldn't enter the ship)
                bool layerMask = isShip &&
(m_overlaps[i].gameObject.layer == 6 || m_overlaps[i].gameObject.layer == 11 ||
m_overlaps[i].gameObject.layer == 12) || !isShip && m_overlaps[i].gameObject.layer ==
3;
                if (!layerMask && m_overlaps[i].transform !=
body.transform &&
                    Physics.ComputePenetration(collider,
m_position, body.transform.rotation, m_overlaps[i], m_overlaps[i].transform.position,
m_overlaps[i].transform.rotation, out Vector3 direction, out float distance))
                {
                    m_position += direction * (distance +
skinWidth);
                    velocity -= Vector3.Project(velocity, -
direction);
                }
            }
        }

///END OF NOT MY CODE

safeDisplacement = m_position - body.position;
advisedImpulse = velocity - motion;

//Update leg targets by subtracting safeDisplacement (remember
the player position is always zero so stuff like this needs to be done explicitly)
foreach (FabrikLeg leg in legs)
{
    leg.deltaPos = safeDisplacement;
    leg.prevTarget -= safeDisplacement;
    if (leg.walkCycle >= 2)
    {
        leg.midTarget -= safeDisplacement;
        leg.endTarget -= safeDisplacement;
    }
}
}

private void CapsuleSweep(Vector3 direction, float distance, float
stepOffset, float minSlideAngle = 0, float maxSlideAngle = 360)
{
    float capsuleOffset = m_height * 0.5f - m_radius;

    //Don't want the ship capsule to detect collision with its
    //individual colliders or the player capsule (as then the ship would displace away from
    //the player and itself)
    //Don't want the player capsule to detect collision with the ship capsule
    // (as then the player couldn't enter the ship)
    int layerMask = isShip ? ~(1 << 6 | 1 << 11 | 1 << 12) : ~(1 <<
3); //IF SHIP 1...1011111 ELSE 1...1111011

///START OF NOT MY CODE

for (int i = 0; i < MaxSweepSteps; i++)

```

```

        {
            Vector3 world_centre =
body.transform.TransformDirection(m_center.normalized) * m_center.magnitude;
            Vector3 origin = m_position + world_centre - direction *
m_radius;
            Vector3 bottom = origin - m_upDirection * (capsuleOffset -
stepOffset);
            Vector3 top = origin + m_upDirection * capsuleOffset;

            if (Physics.CapsuleCast(top, bottom, m_radius, direction,
out RaycastHit hitInfo, distance + m_radius, layerMask))
            {
                float slideAngle = Vector3.Angle(m_upDirection,
hitInfo.normal);
                float safeDistance = hitInfo.distance - m_radius -
skinWidth;
                m_position += direction * safeDistance;
                m_contacts.Add(hitInfo);

                if ((slideAngle >= minSlideAngle) && (slideAngle <=
maxSlideAngle))
                    break;

                direction = Vector3.ProjectOnPlane(direction,
hitInfo.normal);
                distance -= safeDistance;
            }
            else
            {
                m_position += direction * distance;
                break;
            }
        }

        ///END OF NOT MY CODE
    }
}
}

```

Testing

Tests

Note: Actual test outcomes different from the expected are documented under the “*Changes following failed tests*” heading.

Note: Timestamps are hyperlinked to the time stated in the video but may not be completely accurate, especially when I perform multiple tests simultaneously. Timestamps are also provided in the description of each video for your convenience.

Video ID	Covers tests	Corresponding video link
#0	Introduction	https://youtu.be/oMuZyeEmu6M
#1	P1-P9	https://youtu.be/FQRFEGU_Yjw
#2	P10-P17	https://youtu.be/z9XHqQmJfcY
#3	S1-S9	https://youtu.be/9q35CuTRpcY
#4	F1-G4	https://youtu.be/j8dW7_3HgpQ
#5	H1-T3	https://youtu.be/KKrlyXNq2GM
#6	I1-D5	https://youtu.be/DI1jPTKya60
#7	U1-U2	https://youtu.be/6xddMYI725I
#8	Re-testing	https://youtu.be/gWimOdiCSQM

Test ID	Video ID, Timestamp, Primary objective test code(s)	Expected outcome / test purpose	Test procedure	Pass / Fail (P/F)
P1	#1 00:00:00 1.7	When the player walks around a planet, trees and ore are generated at a constant rate.	Verify each planet segment has between the specified range of trees and ore generated on it when the player is nearby.	P
P2	#1 00:11:16 1.8.1	When the player walks around a planet, grass and pebbles are generated at a constant rate.	Verify grass and stone prefabs are popped from the global supply per segment at a constant rate. The global supply should be replenished when foliage is no longer required by a segment.	P
P3	#1 00:17:46	When the player approaches the planet, any towers generated are visible.	Random chance segment colliders are temporarily enabled to check if terrain suitable for tower generation, where this is true tower placeholder are produced and rotated to planet normal.	P
P4	#1 00:27:47	Collecting multiple ore in short time span modifies (not creates) metal count.	Collect 5 metal, notification “acquired 5 metal” collect 5 more and notification updates to “acquired 10 metal” before disappearing.	P
P5	#1 00:36:43 6.3	Rotation is clamped between +90° & -60° on landing, but not in air.	Perform front and back flip to check that on landing rotation is clamped.	P
P6	#1 00:44:44 6.2	Left and right leg step one after the other, at constant rate and naturally.	Look at legs while walking in ship, on planet terrain.	P
P7	#1 00:51:49 6.1	Cannot change local horizontal displacement in air, only on ground.	Use WASD with effect on ground, with no effect in air.	P
P8	#1 00:56:58 9.1	Ship legs correctly deploy on terrain when grounded.	Land ship on ground, leave ship, check if ship legs touch ground. Check legs retract when flying again.	P
P9	#1 01:03:51	Ship opens door if robot is soft locked.	No-clip out of ship, verify that ship opening sequence occurs.	P
P10	#2 00:00:00 1.8.2	Dust is generated rapidly on terrain in front of player.	Walk on terrain in many directions, verify that dust always present.	P
P11	#2 00:11:08 1.8.2	Cloud followers generate at a constant rate, randomly and at about a fixed distance from the player.	Walk on terrain in many directions, verify that cloud followers always present.	P
P12	#2 00:16:05 1.8.2	Cloud surround generates at a fixed distance from the player, updates with player movement.	Walk on terrain in many directions, verify that clouds always surround player.	P

P13	#2 00:18:01 1.8.2	Clouds desaturate at dusk.	Observe a decrease in saturation of clouds sun sets over terrain.	P
P14	#2 00:30:10 1.3.2	Ocean has lip lining shore, moves up and down.	Note the distinction between ocean layers and observe the noise determining wave height.	P
P15	#2 00:44:32 1.3.6	Entrance into and out of atmosphere should be seamless	Fly ship in and out of both atmosphere types slowly.	P
P16	#2 00:54:04 1.1, 1.2, 1.3	Variety of planets can be generated, range of colours, terrain, appearances.	Each time New Generate is pressed in Editor, unique and distinct planet is created from a change in the planet's parameters.	P
P17	#2 01:34:06 1.5	At most two branches split off from each branch, tree branches grow up away from planet.	Observe a variety of trees and their structure.	P
S1	#3 00:00:00 9.2.2	The ships digital thermometer displays the temperature. The temperature calculated is appropriate to the player's location.	Temperature gauge increases when in planet, in proportion to planet temperature, ship altitude and sun dot.	P
S2	#3 00:18:21	Wind noise is lower pitch in ship than out and is lower pitch depending on temperature.	Observe change in pitch on ship exit, enter. Compare pitch between cold and hot planet.	P
S3	#3 00:24:12	Ship rotates to match local terrain normal.	Crash into planet, land gently, on flat terrain, on jagged terrain, while flying, while not flying ship.	P
S4	#3 00:39:01 9.2.1	Both compasses point north on planet.	Fly at range of altitudes in range of orientations, verify compass direction is direction to north pole projected onto display.	P
S5	#3 00:59:42	Ship seat drops automatically after jumped off.	Fly ship, exit flying, lower seat. Fly ship, exit flying, jump off seat.	P
S6	#3 00:50:55 7.1	Stars are added to map when close to them.	Fly close to unmapped stars, they become visible on full map.	F
S7	#3 00:47:24	Map correctly rotates so that up on map is always global forward.	Fly in range of orientations, verify stars on the upper-middle of the map are those in front of the ship (when projected onto the horizontal plane).	P
S8	#3 01:06:23 7.1	Path correctly drawn between closest system and destination using Dijkstra's algorithm (where the weight between stars is the distance between them squared).	Test that Dijkstra's algorithm works correctly on small graph. Test that path to system HME (which is the star closest to the centre of the galaxy) is thus drawn correctly.	P
S9	#3 01:15:47	Lights, interior exterior, and displays (excluding computer	Turn engine off, observe dimming of lights and displays. Monitor ship fuel	P

	9.3	interface) dim when engine turned off. Fuel is not consumed by ship when engine turned off.	gauge (& variable) to verify it does not decrease.	
F1	#4 00:00:00 9.2.3.1	Fuel gauge depletes when ship is on, especially when flying ship and even more so when flying in boost mode.	Monitor ship fuel gauge (& variable) to check rate of consumption when ship is on standby, flying and boosting. Consumption should be larger when flying and larger still when boosting.	P
F2	#4 00:07:06 9.2.3.2	Fuel gauge replenishes when solar panel dot sun is high and close enough to sun.	Rotate ship so that solar panels face sun, note fuel gauge showing charging. Fly ship far from sun, note fuel gauge showing not charging.	P
F3	#4 00:17:03 9.2.4	Velocity meter increases with ship velocity, ship engine pitch increases with rev meter.	Accelerate ship and note these changes, decelerate ship, and note these changes. Relate to graph.	P
F4	#4 00:34:26	Velocity streaks point in direction of ship motion, increase in intensity with ship speed.	Change ship velocity and observe these changes occur.	P
F5	#4 00:39:00 6.4.1	Toggling R changes rotation mode and visual indicator.	Moving mouse horizontally changes ship yaw, press R, aural and visual indication of change, moving mouse horizontally now changes ship roll.	P
G1	#4 00:47:09 5.1	Star positions update when ship moves.	Verify that ship moves relative to planets (ship does not move relative to global space). Confirm that this movement correctly changes the position of stars on the celestial sphere and changes their size based on their distance to the ship.	P
G2	#4 01:01:05 5.1	Star sprite colour and brightness interpolate to their respective sun colour and brightness when approached.	Fly towards and away from stars of various colour, observe this transition.	P
G3	#4 00:47:09 5.1	Sun sphere translated in front of closest star sprite and at correct distance when closest star system loading.	Fly towards and away from stars, note that the closest star becomes 3D.	P
G4	#4 01:09:39 5.2	Planet orbits are approximately circular.	Generate a variety of systems, note orbital paths of their planets.	P
H1	#5 00:00:00 2.1.1	There are no holes in or unjustified walls in any hive generated by the hive generation algorithm.	Test the hive generation algorithm has been implemented correctly by viewing each corridor and the order in which each was created. No tiles are generated on the same coordinate where they may be expected to. No walls between adjacent tiles.	P

H2	#5 00:17:15 2.1.2	Props are generated one per wall and at a constant rate in the hive.	Each square wall has at most one prop on it. Throughout the hive, the local ratio of props to tiles visible is approximately constant	P
H3	#5 00:22:36 2.1.2.1	Collectables are generated one per shelf slot on a shelf prop.	Collectables do not overlap on any shelf observed in hive.	P
H4	#5 00:34:05	Ship remains in current local position when in hive and when far away.	Ship position is fixed when player is not in ship, even when the terrain collider beneath the ship is disabled.	P
H5	#5 00:43:49 2.1.4	Mobile android finds a path to the player where possible and follows it cutting corners along way.	Path displayed for a variety of cases between android and player. Android skips unnecessary detours.	P
H6	#5 00:43:49	Doors open automatically for android.	Wherever android path intersects door, door opens for android.	P
H7	#5 00:54:58 2.1.3.1	Doors stop opening or closing if blocked by player.	Test this is the case.	P
T1	#5 01:01:33 2.2.3	Flats are generated (placeholder replaced) only within a certain distance of the player.	Find the boundary at which the placeholder is substituted in/out for ground, middle, and top floors.	P
T2	#5 01:01:33 2.2.1	Windows rendered on flat placeholders match those on flat generations.	Verify for ground, middle, and top floors.	P
T3	#5 01:12:57 2.2.2	Flats are randomly rotated, and a hold is made in the floor & ceiling of adjacent flats for traversal between.	Check this is the case for a sample of ground, middle and top floors.	P
I1	#6 00:00:00 3.2	Erroneous input characters are ignored.	Test range of invalid symbols, including non-ascii.	P
I2	#6 00:09:19 3.3	Only valid files can be opened.	Test for all valid files, variations on their filenames to verify they are not recognised as valid.	P
I3	#6 00:18:02	Can only withdraw and deposit metal and fuel so that no store contains less than 0 or more than its capacity.	Try withdrawing and depositing too much fuel and metal, check value clamps to boundary. Try for boundary amounts, check it works. Try for otherwise valid amounts, check it works.	P
I4	#6 00:31:32 1.4	Map of planet accurately generated and tower and player positions accurate.	Verify by identifying features on map that correspond to planet features. Player position is at top of map when player at north pole.	P
D1	#6 00:54:15 4.2,3,4	After conversed, Eva disappears on ship leave or when flying ship, reappears	Test these circumstances to verify this is the case.	P

		after 5 mins and after player died with new conversation.		
D2	#6 01:01:48	Conversing with robots can only be done in proximity.	Try conversing at a distance, try conversing when close.	P
D3	#6 00:57:39 4.1	Conversations progress correctly across saves.	Converse with Eva, quit game and reopen note the new conversation available.	P
D4	#6 01:01:48 8.1.1	Upgrades are correctly saved.	Examine updated save file after player gets an upgrade.	P
D5	#6 01:01:48 8.3	Can buy 1 upgrade max with 100 metal from a constructor robot.	Interact with constructor robot with at least 200 metal, only one upgrade should be purchasable.	P*
U1	#7 00:00:00 8.5	Controls button toggles control visibility and controls match the current actions available to the player.	Turn on controls, read controls when flying ship, walking, playing Tetris, turn off controls.	P
U2	#7 00:09:07 8.3	Self-destruct button respawns player in random galaxy location.	Press self-destruct, observe Eva continue prior state, observe ship metal remaining, player metal resetting to zero, ship fuel and player fuel replenishing.	P
U3	#7 00:19:05 8.4	Self-destruct button works in hive, on planet, flying ship, interacting with command line and Eva.	Test self-destructing in these circumstances, function should work correctly the same for all.	P*
U4	#7 00:09:07 8.4	Wipe-save button starts a new game.	Button functions like self-destruct, except Eva resets progression, ship metal resets to zero.	P
U5	#7 00:04:06 8.4	Quit button quits game, continue button resumes game.	Test that they do.	P
U6	#7 00:23:11	The screen pixelates in proportion to the force experienced by the player.	Observe pixelisation on jump, land, and collision with walls.	P
U7	#7 00:30:50	Loading notification appears while a solar system is loading.	Check it does for a sample of systems.	P
U8	#7 00:30:50 3.1	Player can interact with levers when facing them and when close.	Try interacting with lever at range of distances and angles.	P*
U9	#7 00:52:24 8.2.2	When the player runs out of oxygen, their fuel burns faster. When the player runs out of fuel, metal is consumed. When the player runs out of metal and fuel, they die.	Observe the warnings that appear when the oxygen & fuel meters are depleted.	P

Changes following failed tests.

Some aspects of some tests (labelled P* in the Pass / Fail column) went unexpectedly without being sufficient to fail the entire test.

Here are these aspects:

Test ID	Unexpected outcome	Re-test video ID, Timestamp
U8	Ship inner switch was set as type “wing”. Ship wing switches were in antiphase.	#8 00:04:45
U3	Self-destruct on CLI means cannot interact with CLI on respawn.	#8 00:00:00
D5	Metal taken from player <i>before</i> robot announces confirmation.	#8 00:04:45

Test S6 was initially failed as there was missing functionality for displaying stars near the player’s new position after respawning. After adding said functionality during the recording of the test, test S6 was passed.

Evaluation

Evaluation of objectives

For clarity: if a test ID is given, for example “P1”, it may be read as “(test) P1”. If an objective number is given, for example “1.1”, it may be read as “(objective) 1.1”.

Objectives restated.

Every objective from the Analysis section is copied below:

1. Generate Procedural Planets
 - 1.1. Generate Planet Terrain
 - 1.1.1. Generate a Quad Sphere mesh.
 - 1.1.2. Add layered noise functions onto the Quad Sphere mesh for terrain. Different layers of noise functions for different terrain types: Continent noise, mountain noise, crater noise, seabed noise. Offset continent noise with additional noise for more variety. Run this algorithm on the GPU for faster execution.
 - 1.2. Generate Planet Colours
 - 1.2.1. Generate the planet’s colour pallet, randomly based on the temperature of the planet.
 - 1.2.2. Generate a gradient of colours corresponding to terrain height, such that higher terrain gets “colder” colours. Blend a different colour into steep terrain.
 - 1.2.3. Add biomes with latitudinal boundaries, offset with noise that have different yet similar “gradients of colours” (1.2.2)
 - 1.3. Add Ocean and Atmosphere

- 1.3.1. Ocean & atmosphere colour inherits from planet's colour pallet.
 - 1.3.2. The ocean shader should give the spherical ocean visual waves, by oscillating each sphere vertex's local scale.
 - 1.3.3. The atmosphere shader should give the atmosphere an optical density or "thickness", thinner atmospheres are more affected by the sun (stronger warm colour interpolations for sunset and sunrise). Atmosphere facing away from the sun is black.
 - 1.3.4. The cloud shader is a variant of the atmosphere shader, it has a thick atmosphere layer but inside is a hollow sphere, with cloudy noise covering its interior.
 - 1.3.5. Atmospheres can have colour bands; specific latitudes can be assigned different colours like for (1.2.3).
 - 1.3.6. Entrance and exit into and out of the regions of all these shaders should be seamless.
 - 1.3.7. Create a custom render pass with URP to render the atmospheres and oceans to the screen according to their distance from the player.
 - 1.4. Mini map generation
 - 1.4.1. Use terrain data from the planet's mesh to produce an equirectangular projection of the planet's terrain. Colour this map using the biome colours (1.2.3).
 - 1.4.2. The player should be able to view this map on their ship on request.
 - 1.5. Tree generation (as described by my Tree Generation Algorithm)
 - 1.6. Ore generation
 - 1.6.1. Zero to three pieces of ore are generated together, they can be collected to restore the player's fuel and increase their metal count.
 - 1.7. Each planet segment generates a random number of trees and ore within a given range.
 - 1.8. Foliage and cloud followers
 - 1.8.1. Grass and stones should be generated efficiently within proximity to the player.
 - 1.8.2. Clouds should make the planet seem 'alive', that is, a dynamic environment.
2. Generate the Hive (and Towers)
- 2.1. Hive ("dungeon" or tile map) generation
 - 2.1.1. Randomly generate a "dungeon" (Hive) composed of corridors, stairs and prefabricated rooms following my Hive tile Generation algorithm.
 - 2.1.2. Decorate the Hive with prefabricated props (boxes, papers, planks etc.)
 - 2.1.2.1. Add randomly generated shelves (they have 1 or 2 layers, 0 to 3 dividers, that naturally hold either an interactable computer or random prefabricated props)
 - 2.1.3. Insert doors in corridors such that the door can only be seen from in front or behind.
 - 2.1.3.1. Doors should have an opening animation, the reverse for closing and stop their motion if the player is in the way.
 - 2.1.4. Implement my own algorithm custom Pathfinding algorithm that allows NPCs to traverse between positions in the Hive, such as itself and the player, as described in the Problem modelling section.
 - 2.2. Tower generation
 - 2.2.1. Build a tower in any direction at a position, each floor of the tower is prefabricated, floors should seamlessly connect. Windows are randomly added to walls on the exterior of the tower.
 - 2.2.2. Each floor in the tower is either made of a 3x3 or 5x5 tile grid. Ground floors have entrances, the top floor if it's a 5x5 this is either another 3x3 tower, a spire, or an air traffic control style top floor, where the player's ship can land and enter the tower from through a lever-controlled system, if it's a 3x3 this is a flat roof or a garden roof with a tree on top from my tree generation algorithm.

- 2.2.3. For performance purposes, floors should only be instantiated when the player is close enough, a placeholder block should be in their place until this.
3. Make and render a CLI, like command prompt.
 - 3.1. The player can press E to start their interaction with the computer.
 - 3.2. Add a system to convert a string to pixels on a render texture, the opening text prompt should be: "ENTER *HELP* TO GIVE A LIST OF COMMANDS"
 - 3.3. Commands are stored in a dictionary and if the player types and enters a valid command, said command is executed. One command, GENERATE_MAP, is only available on the ship and displays a mini map of the planet (1.4); another, OPEN TETRIS.EXE, starts a game of Tetris.
 4. Dialogue system
 - 4.1. Save progress of a series of conversations between Eva (on the ship) and the player. Conversations should be available every 5 minutes and once per life of the robot.
 - 4.2. Each sentence should be composed of 3 phrases each assigned a mood which affects how the phrase is rendered. Letters oscillate according to sine waves of a frequency dependant on the mood of the phrase they are in. This adds personality to the dialogue system.
 - 4.3. Sentences should transition in and out of view aesthetically by fading up at a rate according to an interpolation function $f(t) = t^n$ where $0 < t < 1$ and n is a tuneable parameter.
 - 4.4. Conversations should be stored and read from json files, annotated with each phrase's mood. Mood properties are also read from json files. This allows for streamlined creation of conversations by me.
 5. Galaxy generation (described in the Problem modelling section)
 - 5.1. For performance purposes, all stars except the closest to the player are represented as 2D sprites, they should transition seamlessly to 3D models when the player is close.
 - 5.2. Planets should orbit the sun using equations of motion.
 6. Player movement
 - 6.1. The player should be able to move in 3 dimensions.
 - 6.2. Players legs animate movement by implementing the FABRIK algorithm (described in the Problem modelling section)
 - 6.3. Mouse movement left and right controls yaw rotation, up and down controls pitch, pitch is to be limited by tuneable parameters, if the player is on a planet. If they are not grounded, the player's body will rotate pitch, not just their head and this rotation will not be limited, so the player could do a backflip for example.
 - 6.4. The player should be able to control the ship
 - 6.4.1. The ship has toggleable wings that change whether left right mouse movement controls yaw or roll rotation. These wings may be useful on planet's where the ship's roll is adjusted so that the ship aligns with the planet's normal at the ships position.
 7. Star mapping system
 - 7.1. Each visited star will become visible on a star map the player can view. The player can select a coordinate in this star space and Dijkstra's algorithm will produce a path with refuelling points as nodes to get there from the player's current coordinate. This path then becomes part of the ship's HUD during flight.
 8. Gameplay cycle, menu, options
 - 8.1. Produce a functional start menu with options to exit, start a new game or continue.
 - 8.1.1. The game saves the star map, dialogue state, inventory, and the player's star space coordinate. The game saves each time the player dies or enters a new coordinate in star space. Upon death, the player respawns at a random star space coordinate losing their inventory but retaining their dialogue state and star map.

- 8.2. Oxygen and fuel deplete over time when the player is not in their ship.
 - 8.2.1. Oxygen is replenished in the ship and hive.
 - 8.2.2. When the player runs out of oxygen, their fuel burns faster. When the player runs out of fuel, metal is consumed. When the player runs out of metal and fuel, they die.
- 8.3. Upgrades to the players statistics (such as fuel burn rate) can be acquired by talking to a constructor robot while possessing 100 metal to spend on it.
- 8.4. Pause menu has options to start a new game, self-destruct (force start of death sequence), view controls and quit the game.
- 8.5. Controls viewed should be specific to the actions the player can currently take.
- 9. Ship functionality
 - 9.1. Ship legs deploy onto terrain to give the appearance they are supporting the ship.
 - 9.2. Ship widget meters
 - 9.2.1. Compass widget, aid fully for navigation, points to the north pole of the closest planet
 - 9.2.2. Ship thermometer displays the calculated temperature of the ship.
 - 9.2.3. Ship fuel depletes over time if the engine is on.
 - 9.2.3.1. Ship fuel depletes faster if boost mode enabled.
 - 9.2.3.2. Ship fuel replenishes via solar panels.
 - 9.2.4. Velocity meter displays velocity and made-up RPM of ship.
 - 9.3. Ship widgets and lights dim when engine off.

Every specific objective above is discussed in this evaluation.

Objective 1: Generate Procedural Planets

As demonstrated in P16, a variety of natural looking planets can be generated. More specifically, P16 demonstrates how the max function is used to layer noise functions onto the planet's quad-sphere mesh and how warp noise makes continent noise more intricate (adjectives "knotted" and "warped" are used in the test). The algorithm is also written in a compute shader, improving performance. These points together satisfy 1.1.2 to a great extent. The structure of the planet's quad-sphere mesh is also explained in P16, but the mesh is utilised and referenced by features tested in P1 & I4. Overall, the fact that the mesh had 6 sides, as opposed to an icosphere, made the map generation tested in I4 simpler to implement, so was the right choice, satisfying 1.1.1 fully.

Additionally, P16 verified over a range of planets that biomes were generated correctly, and hotter planets' planet textures were composed of redder hues and colder planets of bluer. This proved effective at visually marking the temperature of the planet. Moreover, the observed textures generated were distinct, with each biome being distinct from others on the planet surface, demonstrating the success of 1.2.1 and 1.2.3. In I4 it was explained how the colours of steep terrain differed from the surrounding colours given by the planet texture, especially as steep terrain colour was shown to be excluded from planets maps generated, allowing a comparison to be made. P16 also showed that higher altitudes were assigned cooler colours and S1 verified that this corresponded to a decrease in temperature, 1.2.2 was therefore met and embedded into the game's mechanics.

Although not explicitly stated in P16, 1.3.1 was satisfied because atmosphere and ocean colours on planets generated were observed to be of the same pallet as the terrain colours. P15 demonstrates that the solution satisfies 1.3.3 in a meaningful way as the closer the player got to the edge of the atmosphere, the more visible the surrounding stars became and vice versa, this also specifically

showed fulfilment of 1.3.6. P16 highlighted an error in the satisfaction of 1.3.3 as the code to modify the atmosphere colour for sunrise was commented out, with the re-addition of this code 1.3.3 was fully met. P16 and P13 also showed how the atmosphere facing away from the sun was notably darker. The inclusion of the cloud shader as described in 1.3.4 is shown in P15 specifically but also throughout the testing process. Unlike the regular atmosphere, the cloud atmosphere has a visible interior, and this was shown to transition seamlessly from visible to hidden on exit of the planet and vice versa on enter. By testing 1.3.4 with the criteria from 1.3.3 for “optical density” its full implementation was demonstrated to a great extent. P16 highlighted and explained the inclusion of colour bands on the atmosphere with a range of examples, thoroughly demonstrating the satisfaction of 1.3.5. In P15, 1.3.7 was also mentioned and its functionality verified. P14 was successful in verifying the satisfaction of 1.3.2, the additional inclusion of a visible lip lining shore on the ocean proved useful, as intended, in identifying the presence of an ocean on planets in other tests. Therefore, the utility of 1.3.2 was demonstrated beyond the scope of the test.

Mini map generation for each planet was covered exclusively by I4. In the test, how the equirectangular projection was produced was covered and the relative positions of the player and towers on the planet were shown to correspond to their relative positions on the map. The colours of the map were shown to be accurate to the planet, however, inclusion of steep colours in the map as described above was not implemented. Although 1.4.1 does not specify the inclusion of steep colours in the map, it is fair to say this is necessary for a proper map meaning 1.4.1 was met to some extent. When I revisit this game in the summer, this is an update I intend to develop as it can be confusing as shown in I4 when the terrain surrounding the player is of a colours inconsistent with those on the map. 1.4.2 was implemented successfully as the player can enter the command “MAP_PLANET” into the CLI and the map is then generated for viewing.

Tree generation was tested in P17, the following of the rule: “from each tree branch can extrude 1 or 2 new branches”, was supported with multiple examples, and visually this proved an effective rule at creating a diverse yet controlled range of trees. In P17 it was also explained and shown how branches were biased to extrude in the direction of the planet normal. P17 showed visual confirmation of more leaves being generated at higher levels of branches, and of decreasing width of higher levels of branches, hence 1.5 was well met.

Ore generation was tested in P4, and with an explanation and multiple examples (including examples from U9) it was shown that 1.6.1 was fully satisfied.

Tree and ore instantiation, 1.7, was tested directly by P2, where on a decent sample of terrain segments the number of trees and ore was counted and their counts were all within their ranges specified for the planet. How these ranges differ from a planet with and without an ocean was explained and shown successfully.

This was also the case for 1.8.1, which was shown to be met by P2 in a similar fashion to 1.7. Additionally for 1.8.1, the implementation of the requirement for efficient instantiation of grass and stones was shown as grass was popped from the sun’s stack of instances onto the terrain surrounding the player and pushed when the player moved away. This approach was implemented for efficiency as there were lots of grass and stones to be rendered so they impact the performance of the game significantly. That there was not a noticeable drop in frame-rate due to grass generated between planets with lots of grass and planets without during P2 demonstrates the satisfaction of this requirement to a great extent. In order to reduce the subjectivity of testing 1.8.2, its fulfilment was split between four tests: P10, P11, P12, and P13 targeting different aspects of the objective. The dynamism of the clouds was demonstrated in P11 where cloud followers moved over the terrain

close to the terrain in all directions following the player, P11 showed successfully this gave the impression of wind on the planet blowing great storms of dust across the planet as the cloud followers were indeed generated at a constant rate. This dynamism is important in creating the illusion of the planet being ‘alive’. P10 emphasised the realistic nature of these clouds, as smaller dust clouds were shown to correctly crawl along the terrain in front of the player. However, P10 revealed that these dust clouds were sometimes too transparent for the player to see, lessening their impact on 1.8.2 and so reducing the extent to which 1.8.2 was met. P12 showed consistency in the ‘alive’ environment, as cloud surrounds were shown to be generated in a complete circle surrounding the player hence applying a realistic background for the cloud followers to demonstrate the planets dynamism on. P13 verified that the cloud colour is also affected by the light intensity of the planet terrain, improving the realism of the clouds. Overall, 1.8.2 was therefore met to a decent extent, limited by the invisibility of some clouds in the dark and some dust clouds in general.

Objective 2: Generate the Hive (and Towers)

By successfully demonstrating and explaining why there were no holes or unjustified walls in any hive generated by the hive generation algorithm in H1, 2.1.1 was shown to be fully met as the ordering of the tiles generated was indicative of a successful implementation of the Hive generation algorithm (tiles were generated one corridor at a time, each corridor started with a tile adjacent to a previously generated corridor), and this was a necessary step to demonstrating the absence of holes and unjustified walls. H2 directly showed that 2.1.2 was also satisfied, and the rate at which props were generated was verified to be constant, so the Hive neither felt bare nor overwhelming. H3 verified that shelves could hold prefabricated props, one per division, and there were different types of dividers for each shelf to implement in their creation. However, H3 did not verify that interactable computers could be generated on shelves, although their programming was covered in I2, as by chance there were no computers generated on any of the tested shelves. Therefore, 2.1.2.1 was shown to be met to some extent, but not fully. Additionally to 2.1.2.1 however, robots were also added as another option for a top shelf layer occupant. In H1, it was also explained and verified that doors are locked if the wall they replace “isn’t already removed”, which meets 2.1.3 to some extent, the code for inserting doors on tiles with “two opposing wall” is also shown, although isn’t explained during the test nor tested, the success of this conditional insertion is verified by the footage in H1, but since it was not explicitly tested, 2.1.3 has only provably been met to some extent. Their opening and closing animations are vigorously demonstrated in H7, fully satisfying 2.1.3.1. Although H5 provides substantial evidence for completion of 2.1.4, it would have been better in retrospect to have made a point of thoroughly testing the algorithm as there is lots of potential for error (as shown when the android gets stuck behind the door). With account of this potential, it is fair to say 2.1.4 has been satisfied to a decent extent. Therefore, another of the things I intend to update in the summer is a more polished version of this algorithm that accounts for collision between the robot and the props.

Towers were shown to be generated in any direction by P3 (and other tests also), where towers are seen pointing away from their planet a different latitudes and longitudes. Although not specifically tested, the success of this aspect of 2.2.1 simply requires a translation and rotation after the tower is generated so the fact that it was observed for a range of examples is sufficient to deduce the requirement has been met. T2 demonstrated that windows were randomly added to the exterior walls of each tower, and additionally, these windows were consistent between the placeholder and the real tower. This requirement, 2.2.3, which shown to be met fully by T1, was necessary for the game because the rendering the real towers significantly reduces the performance of the game, so any flats out of the player’s reach are replaced with less detailed placeholders as demonstrated in

T1. In T3 it was shown that the flats / floors connect seamlessly at run time with a hole in the floor of the flat above. These points together demonstrate the great extent to which 2.2.1 was met. The different types of towers are shown throughout testing but specifically in T1, although the 3x3 tower with the tree on top was never generated, the tree was shown in P17 (it is the brown tree specifically). The “lever-controlled system” of the “air traffic control style top floor” is shown, described, and used in D5 but it was never explained or explicitly tested. Therefore, the testing for 2.2.2 was performed to a weaker extent hence it is still clear 2.2.2 was met but this was demonstrated without proper rigour.

Objective 3: Command Line Interface

In each of the tests I1, I2, I3 & I4 it is demonstrated that interaction with the CLI is successfully started by pressing E. This meets 3.1 fully.

As part of I1 it was necessary to explain and demonstrate how the string of characters the player enters is converted to pixels on the CLI display texture, characters that did not have mappings via the sprite sheet were also shown to be successfully ignored and in the case that they were not ignored (by forcing the characters into the user input string variable) they were rendered with a space character instead, mitigating the error. Hence 3.2 was met completely, as it was observed that the opening text prompt was indeed: “ENTER *HELP* TO GIVE A LIST OF COMMANDS”.

Multiple valid commands were successfully executed during the testing of I1, I2, I3 & I4, including “MAP_PLANET” (which replaced command “GENERATE_MAP” in 3.3) and “OPEN TETRIS.EXE”. “MAP_PLANET” was shown in the code to only be available to a computer with the Boolean variable “shipComputer” enabled but this was never verified in testing. “OPEN TETRIS.EXE” makes use of a filename parameter and different options for the parameter such as “OPEN HISTORY.LOG” were also executed successfully, invalid filenames “OPEN hitory.log” threw the appropriate error in the CLI as showing I2. Withdrawing and depositing fuel also worked successfully as shown in I3, as the user could not perform a transaction that resulted in negative values or values in holders greater than their respective capacities, one issue with this presented itself in testing that in some circumstances this validation unnecessarily rounds the transaction value down, weakening the extent to which 3.3 was met as the command was not executed as expected. There was one error with Tetris where the display was not visible, this was fixed during the recording. Overall, the execution of this wide range of commands and the error handling of invalid commands demonstrates the near full extent to which 3.3 was met.

Objective 4: Dialogue System

This objective was overall tested weakly, simply because most of the requirements were either met or they weren’t so verifying their functionality was as simple as observing dialogue in conversations with Eva and the upgrade robot. However, without explicit tests, it is difficult to properly demonstrate that the objective requirements have been met. During testing, the conversations with Eva were skipped over, but the conversations with the upgrade robot in D5 are worth analysing (both in the initial and re-test). There it is clear that each line can contain three phrases, for example the text “Right, now, I can upgrade your **ship’s solar charge efficiency**.”, where the first portion is white, the specific upgrade variable is red, and the final full stop is white. These correspond to different moods, where the frequency of oscillation of the red text and white text is different. This analysis is sufficient to conclude that the 4.2 was met, although this has not been tested specifically during recording.

Similarly, this conversation also demonstrated multiple times that sentences faded in and out smoothly meeting 4.3, though the code for this was not explained.

Due to lack of time, it was not shown that conversations were available every 5 minutes and once per life of the robot, as to prove this is the case would mean waiting for five minutes while recording, although I tried to do other tests in the meantime, five minutes of consecutive gameplay was never achieved. However, the simplicity of the code required to implement this feature and its careful explanation during D1 means it is fair to say that this aspect of 4.1 was met although only provably to a weak extent. D3 demonstrated that conversation progress was saved, so 4.1 overall was met to a sufficient extent.

The final objective of the dialogue system, 4.4 was met fully, the json files I created are included in the Technical solution section, and they are formatted as described in 4.4.

Objective 5: Galaxy Generation

Another way in which performance of the solution was improved was by rendering all stars except the closest as 2D sprites. This significantly reduced the number of vertices that needed to be rendered, making playing the game feasible. The success of G1 demonstrates that each 2D star sprite is rendered on the celestial sphere at the appropriate position and scale. In G2, the colour of the star is shown to interpolate to a more saturated colour as it gets closer, because the colour of the sun is saturated. The transition between 2D sprite and 3D mesh was demonstrated to be seamless in G3 as the mesh perfectly overlays on the sprite, but the colours were not exactly the same, causing the swap to be noticeable. This is because of how the sprites are rendered as transparent but the sun is rendered as opaque, so the appearance of the colours changes slightly. However, this is not so much an issue because the frame the swap occurs is the same frame the planets are loaded, which requires a few seconds of loading time, so the immersion is inevitably broken anyway. That is to say, 5.1 was met to a decent extent, but the transition could be improved in the future to be completely seamless.

It was demonstrated in G4 that equations of motion are used successfully to derive the initial velocity of each planet, as the path that each planet takes around the sun was shown to be approximately a circle. It was not, however, shown that the circular path was formed due to the equations of motion, although this was demonstrated by the fact that the initial velocity did result in a circular path, as predicted by the equations of motion. Therefore, 5.2 was fully met, but the testing for it could have been more thorough.

Objective 6: Player Movement

The entire testing process demonstrated 6.1 was fully met rigorously, but especially P5 & P7, which properly demonstrated jumping.

The use of the FABRIK algorithm to animate the legs was explained and shown to successfully work in P6 on a range of terrain including steep, bumpy, and flat. The legs were also shown to synchronised after every step, satisfying 6.2 to a great extent.

It was clearly demonstrated in P5 that 6.3 was fully met. During P5 the player performed multiple back/front flips in the air and their rotation smoothly transitioned back to being clamped when they landed every time.

Throughout all of the tests, but especially F1, F2, F3, F4 & F5, the player was able to exhibit a high level of control over the ship, satisfying 6.4. Moreover, the player could press R to change which axis

the ship rotated around when the mouse was moved horizontally. This was extensively tested in F5, and the result was that 6.4.1 was completely met.

Objective 7: Star Mapping System

This singular objective was covered by a singular test, S8. In S8, Dijkstra's algorithm was shown to work on a smaller complete graph and the smaller nature meant it was possible to verify by inspection that the shortest path had been successfully found, implying a correct implementation of the algorithm. Then in the game, in the larger graph of 2000 nodes the algorithm then appeared to work correctly, finding what seemed to be the shortest (where distance between nodes is squared so does not obey the triangle inequality) path from the player to the centre of the galaxy (HME). The distance is squared to produce a path that maximises refuelling points. Additionally, in S8 it is shown that the path becomes visible on the star map (green dotted line) while flying the ship. Together, these points demonstrate thorough testing to show that 7.1 was fully met.

Objective 8: Gameplay, Menu, Options

The pause menu has five options: "New Game, Self-Destruct, Continue, Controls, Quit" each of which's functionality was tested by U4; U2 & U3; U1, U2, U3, U4 & U5; U1 & U2; and U5 respectively. Note that while "Continue" was not tested directly, its successful functionality was demonstrated alongside the tests for the other buttons, where it was used extensively. The controls button was shown in U1 to work correctly in the entire range of contexts: walking, interacting, flying, playing Tetris, this fully satisfies 8.5. The "Self-Destruct" button also worked in a wide range of contexts including walking, interacting, flying, in the hive. However, there was an issue raised in U3 with "Self-Destruct" where if the button was pressed while interacting with the CLI, the player could not interact with the CLI on respawn. This was resolved in the re-testing section. There was also an issue raised in S6 where the star map did not update on "Self-Destruct" or "New Game", this was resolved immediately during the recording of S6. After these issues had been resolved, the tests U1, U2, U3, U4 & U5 demonstrated that the menu functioned as intended and 8.4 was met fully, however 8.1 was not met as a start menu was not included in the solution. In order to demonstrate the difference between "Self-Destruct" and "New Game", the star map and Eva conversation count were shown in U4 to be reset on "New Game", but not for "Self-Destruct". It was not tested during recording that the game saves each time the player dies or enters a new coordinate in star space, nor that they lose their inventory. However, there is evidence that this is the case during P14, where due a bug which was then immediately explained and fixed, after dying with 37 metal and respawning, the player's metal counter was reset to 0. Therefore, overall, 8.1.1 was met fully, although ideally there should have been more tests to verify what the game resets when the player dies.

In U9, the player's oxygen and fuel meters are shown to deplete when the player is on the planet. Throughout all tests that occur in the ship or hive, it can also be observed that the player's fuel meter does not decrease, and the player's oxygen meter replenishes, this is most notable in H3 when the player enters the hive. This is sufficient to satisfy 8.2 and 8.2.1 fully. Additionally, U9 specifically tested and verified that when the player runs out of fuel, metal is consumed. When the player runs out of metal and fuel, they die. Therefore 8.2.2 was also fully met.

When the player interacts with the upgrade / constructor robot, if they have 100 metal, they can spend it on a randomly selected upgrade. D4 verified that said upgrade is then saved to a json file, and D5 demonstrated that the player cannot get an upgrade if they are holding less than 100 metal and can only get one upgrade even if they are holding at least 200 metal. There was an issue raised in D5 that the robot takes metal off of the player before they confirm to the player that the upgrade

has been completed. This may confuse the player, so it was fixed in the re-testing section. When I return to this project in the summer, I would like to change the conversation structure so that the player can choose which upgrade they can get, as that way getting an upgrade is more rewarding. Additionally, I added unique dialogue for each upgrade and hints as to how use the upgrades effectively as shown in D5. Therefore, 8.3 has been met to a great extent.

Objective 9: Ship Functionality

The ship has four legs which deploy onto terrain to give the appearance they are supporting the ship. This functionality is tested repeatedly in P8 with success. It is noteworthy that the legs do not retract smoothly for the reason stated during P8 that it is impossible for the player to see the legs retract anyway. Therefore, 9.1 has been fully achieved, demonstrated by thorough testing.

The ship has multiple widgets. Both compass widgets are tested in S4, which demonstrated that both point north of the planet the player is within, including when the ship is close to directly above the north pole. This fully meets the requirement of 9.2.1. When the player is not within a planet the compass widget is replaced by the velocity meter widget. The velocity meter and its RPM component are tested in F3, where it is shown that the velocity meter increases in proportion to the velocity of the ship and is full when the ship is at maximum hyper velocity. F3 also demonstrates that the pitch of the engine noise and the RPM meter are synchronised, making acceleration and deceleration more impactful for the player. As 9.2.4 states, this RPM meter is made-up but does change with the velocity of the ship. Hence, 9.2.4 is fully met as shown by F3. In S1 it was explained how the sun angle, planet temperature and planet altitude contributed to the temperature of the player, each factor was tested and its effect on the calculated temperature was observed. All observations were as expected and an additional observation during H3 verified that on a redder-so-hotter planet the temperature was significantly higher. This temperature determines how quickly oxygen is consumed by the player on a planet and is indicated to the player via the temperature widget, hence 9.2.2 was fully met. Fuel is consumed by the ship when the engine is on, this was shown to be the case by F1. F1 also demonstrated that fuel was burned at approximately twice the rate when the ship was flying, and then ten times the rate when the ship was flying in boost mode. Therefore, F1 demonstrated that 9.2.3 and 9.2.3.1 have been fully achieved. Fuel is replenished by the ship when the intensity of light on the solar panels is great enough. This is approximated by the square distance between the sun and the ship and the dot product between the solar panel's normal and the sun direction. F2 verified this by toggling the solar panel's normal between facing towards and away from the sun and observing that when the solar panels were facing towards the sun, the ship fuel meter has a charging symbol and the fuel meter was increasing, and when they were facing away, the ship instead burned fuel with not charging symbol displayed. Hence the ship fuel replenishes realistically via solar panels so 9.2.3.2 has been met to a great extent.

Finally, when the engine is turned off, 9.3 requires that the lights dim as well as the ship's widgets. This was observed to be the case during S9, where it was also shown that fuel is not burned when the engine is off, this was verified because the charging symbol appeared on the fuel meter, made easier to spot because the fuel meter was slightly more visible than the rest of the widgets (intended specifically to signal to the player that this is the case). Hence, the requirements of 9.3 have been fully met.

Brief summary of objective evaluation

Overall, the vast majority of the objectives have been fully met. Some of the objectives were lacking proper documented testing such as those for objective 4. Additional features have been tested that were not objectives such as F4.

Feedback on solution

Responses to questions

In addition to my objective evaluation, it is important that my target audience evaluates my game with responses to questions targeting its more subjective aspects. Therefore, I asked two people familiar with first person 3D games who are interested in exploration focused games to respond to the questions below:

What was your favourite aspect of the game?

1. The towers appealed to me as there was an aspect of tension when searching for metal in them. I particularly liked when the robot was chasing me.
2. I found flying the ship to be quite entertaining, it was fun to fly really fast, especially on planets where a high level of control was required.

What was your least favourite aspect of the game?

1. The interaction with Eva was one sided so it felt a bit stiff. Perhaps it would have been better for the player to contribute to the conversation.
2. The ground movement was quite slow and did not encourage the player to explore the planets. Adding more movement options would make exploration more entertaining.

How did the rate at which your fuel was burned impact your experience?

1. I was constantly checking how much fuel I had remaining while exploring the planets. This was quite stressful, but it did maintain the pace of the game. It also caused me to die a few times at first, but I soon learned that my time on planets was supposed to be limited.
2. Initially, I found it to be quite frustrating but then it became an enjoyable challenge to get as much metal as possible and return to the ship before I ran out of fuel.

Was flying the ship intuitive?

1. Yes, although flying the ship was less responsive player movement, this was intuitive as this is natural. Boost mode enabled me to fly between star systems quickly which was useful. It was slightly difficult to land on planets but the “Press space to match velocity” prompt made slowing down intuitive.
2. Yes, I had a high level of control over the ship, and it was fun to manoeuvre between mountains on the planets. I kept wanting to turn my ship during boost mode, but this would also disincentivise turning it off, which perhaps isn’t a good thing.

How was your experience with the computer on the ship?

1. I didn’t spot it at first, but it was quite easy to interact with and its functions were useful like withdrawing fuel. I didn’t like having to type EXIT every time I finished using the computer, maybe pressing escape would be better.
2. Positive. The error messages were useful at informing me how to properly enter commands. Tetris was fun to play despite it being a bit out of place. Also, it was nice to be able to press up arrow and re-print the command above, I pressed it sub-consciously but was surprised that the functionality was implemented.

Were you ever confused about what you were supposed to be doing?

1. After interacting with Eva, it was clear that my goal was to go to HME. I assumed that I wasn't supposed to understand how this could be achieved immediately so decided to explore the planets available to me to find answers. The ship computer drew a path to HME which was in the middle of the map and so I knew to follow the path there.
2. No, I knew I was supposed to go to HME, but I chose to fly around the planets.

Would you consider these elements of my game (considered following feedback in the analysis section) to have been implemented successfully in my game?

- *Quality, engaging & extensive exploration,*
 - *Focus of the difficulty of the game around its exploration.*
 - *An engaging story to go alongside the gameplay.*
 - *Upgrades to make resource gathering more effective.*
 - *Include upgrades to allow for exploration of extreme locations.*
 - *Include settlements of NPCs to interact with.*
1. The exploration was engaging because each planet had distinguishable terrain. Collecting metal was difficult because of fuel limits and robots in the towers but this difficulty make exploration enjoyable. The story was very minimal and so not engaging although it did motivate enough follow the path to HME. The upgrades were well explained although they didn't improve my abilities very much. It was fun interacting with other robots in the towers, they felt alien.
 2. Mostly yes, the exploration was certainly extensive and mostly engaging but because the planets were procedural this limited the quality of exploration. There were upgrades available to explore extreme locations and improve resource gathering, but I didn't have time to try them out. The Rebels of Entropy were fun NPCs to interact with and rewarding to reach.

Evaluation of responses

Note: how I intent to improve my game over summer following these responses is explained in the future improvements of solution section.

The ability of the players to quickly pick up the controls, both of the ship and computer and know what they wanted to do demonstrated that my controls button was implemented to great effect and the interactions with Eva were meaningful. Flying the ship and exploring the towers were listed as the best aspects of the game, my prediction is that the tension and flexibility in the approach to both is what made them enjoyable. This is also what I intended to implement in the planet terrain exploration but the response to this was less positive. The slow movement and minimal movement options of the player clearly made exploring the terrain less engaging. To some extent, it is evident this was compounded by the fast rate at which fuel burned preventing the Player 1 from freely exploring the terrain, but to some extent, this also counteracted the reduction in engagement as it shifted the objective of exploration from freedom to efficiency, as described by Player 2. It is also clear that the conversations lacked player input, which was a missed opportunity for engagement, despite this though they were positively responded to.

Future improvements of solution

In the summer I intend to revisit my game to improve and add a range of features. This is in response to the feedback I received above but also my own experience of playing the game. The

primary intent behind my revisiting is to enhance the player's engagement, and this can be broken down into two major tasks. The first is to make the planets more engaging to explore and the second is to make the story / objective of the game more streamlined and better paced. This second task is in slight contradiction to the open-ended nature of my game discussed in the analysis section, but from the responses above it is clear that players naturally have a desire to explore so do not need additional motivation.

One improvement I will make to as part of the first task is to make the environment the player explores more immersive. More specifically, I want to replace the current trees on my planet with a new type of tree generated from Bezier curves instead of cylinders (I discussed this during P17), and also the current grass on my planet with fields composed of individual blades of grass (also discussed during P17). I have already created these, so their implementation in summer is realistic, although dynamically generating grass to follow the terrain and keeping the frame rate up may prove challenging. This would improve immersion with the game as it would make the planet feel more alive. Additionally, having different types of ore may encourage the player to explore different places on the planets.

Another is updating the planet mapping system, which was not mentioned in the player feedback, so I assume it was not utilised effectively. During U8, I also demonstrated difficulty using the planet map to navigate the planet. I therefore want to modify the planet map system to update the player's position on the map regularly, so that the player can gain immediate feedback as to whether they are heading in their desired direction or not. I also want the map to be visible to the player on the ship while they are exploring the planet. This could be achieved by automatically replacing the star map with the planet map when the player enters a planet. That way, the player is encouraged to use the map as they do not have to spend time entering the command on the computer, and this makes exploring the planets easier so more engaging. Also, this encourages the player to explore the planet in the ship, addressing the constructive criticism I received in the feedback section above regarding the slow player movement on the planet by reducing the time the player is intended to spend exploring on foot. However, automatically replacing the star map with the planet map would add overhead to the loading time of the star system, by about 1.5 seconds per planet (on my computer), it follows the map generation algorithm should also be optimised. For example, the alpha channel that sends player and tower position data to the planet map shader could be applied to another texture, so that the other texture can be updated quickly while the planet map texture does not need to be re-drawn every time the map updates. This would enable the map to update in real time, after it has been generated.

In H5, the android, during pathfinding to the player, gets stuck behind a door. This was not accounted for in the pathfinding algorithm I implemented, which by default assumes all the tiles are empty such that if there is an obstacle blocking the path through the current tile, the android does not detect it as such. I intend to implement a solution to this problem. One approach may be to keep the middle forward, back, left, right and centre of each tile empty of props, restrain the android to only move in these areas, and remove collision detection between the android and the props (and doors, which already automatically open for the android) to mitigate the error in the case the android accidentally enters prop-filled regions.

As part of the second task, I want to encourage the player more to get upgrades, and to make the upgrades more desirable. In the current build of the game, the upgrade the player gets is selected for them at random, and improves one of the players statistics, each of which can be improved at most three times. It was stated in the feedback above that these upgrades did not have significantly noticeable effects. One way of making them more interesting would be adding unlockable widgets

for the ship. Rather than this, it may be easier to require the widgets (specifically the compass, velocity meter, star map, planet map and temperature gauge) the ship has in the current build of the game to be unlocked in the updated build. By doing this, the player can learn the mechanics of each individually when they unlock them, and not be overwhelmed when they first play the game. These unlocks are more interesting than the current upgrades as they are visual features in the game so they would give the player more motivation to continue playing. This, of course, is only the case if the player knows what they can buy with their metal, so adding an interface for the player to select the upgrade they want is also a feature I want to add to my game in summer.

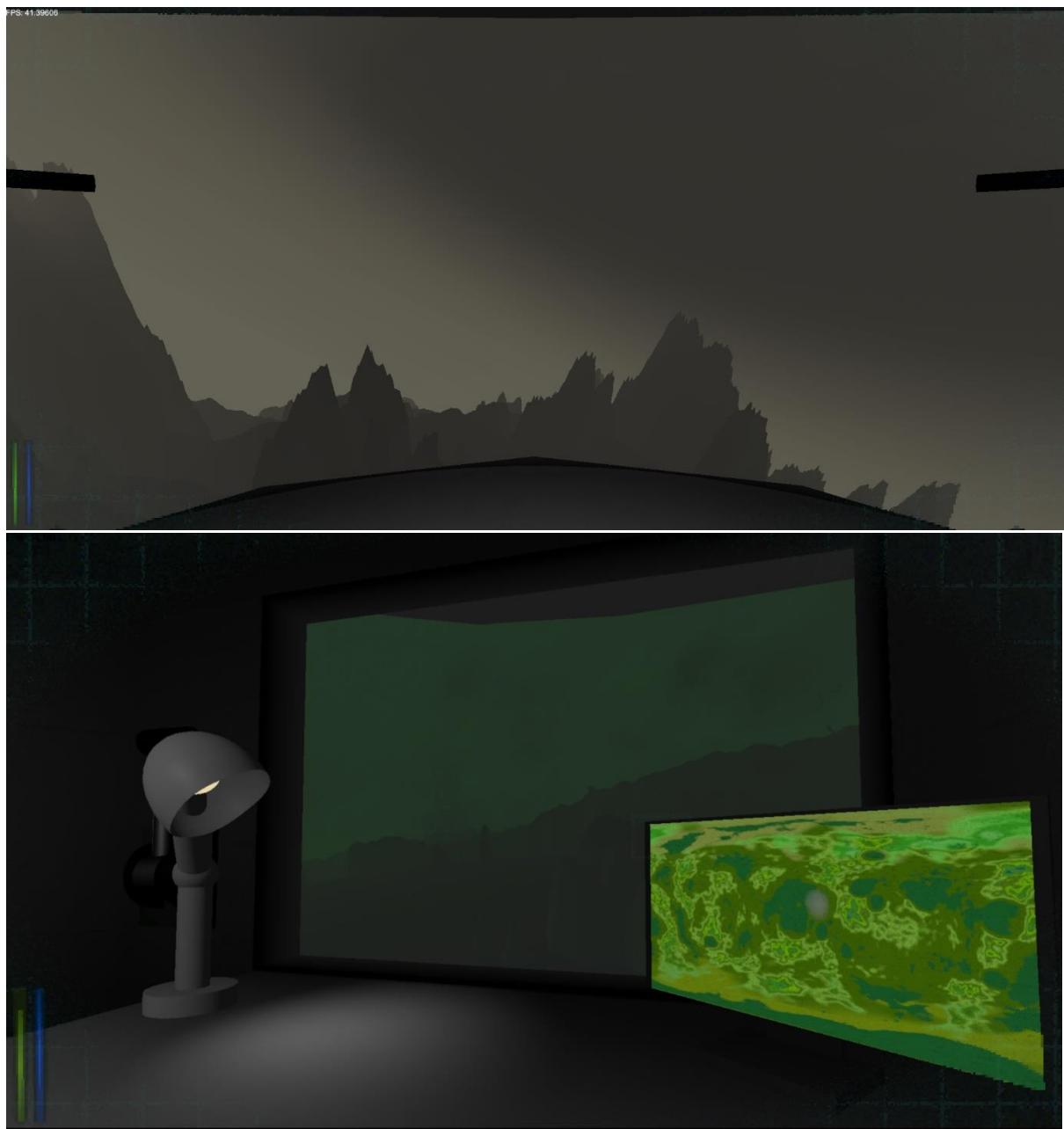
Finally, I want to add special star systems that are not procedurally generated at run-time but instead are fixed for every playthrough. These can act as checkpoints and will be places along the path between the player and HME, they would replace star systems that would otherwise be procedurally generated. At these special star systems, because they have been prefabricated, I can add linear story elements to Third Law (the title of the game), to improve the game's pacing and storyline. This feature will take time to add to the game, which is why it wasn't implemented in the current build of Third Law, but it is not technically difficult to include so is a realistically achievable improvement I would like to make.

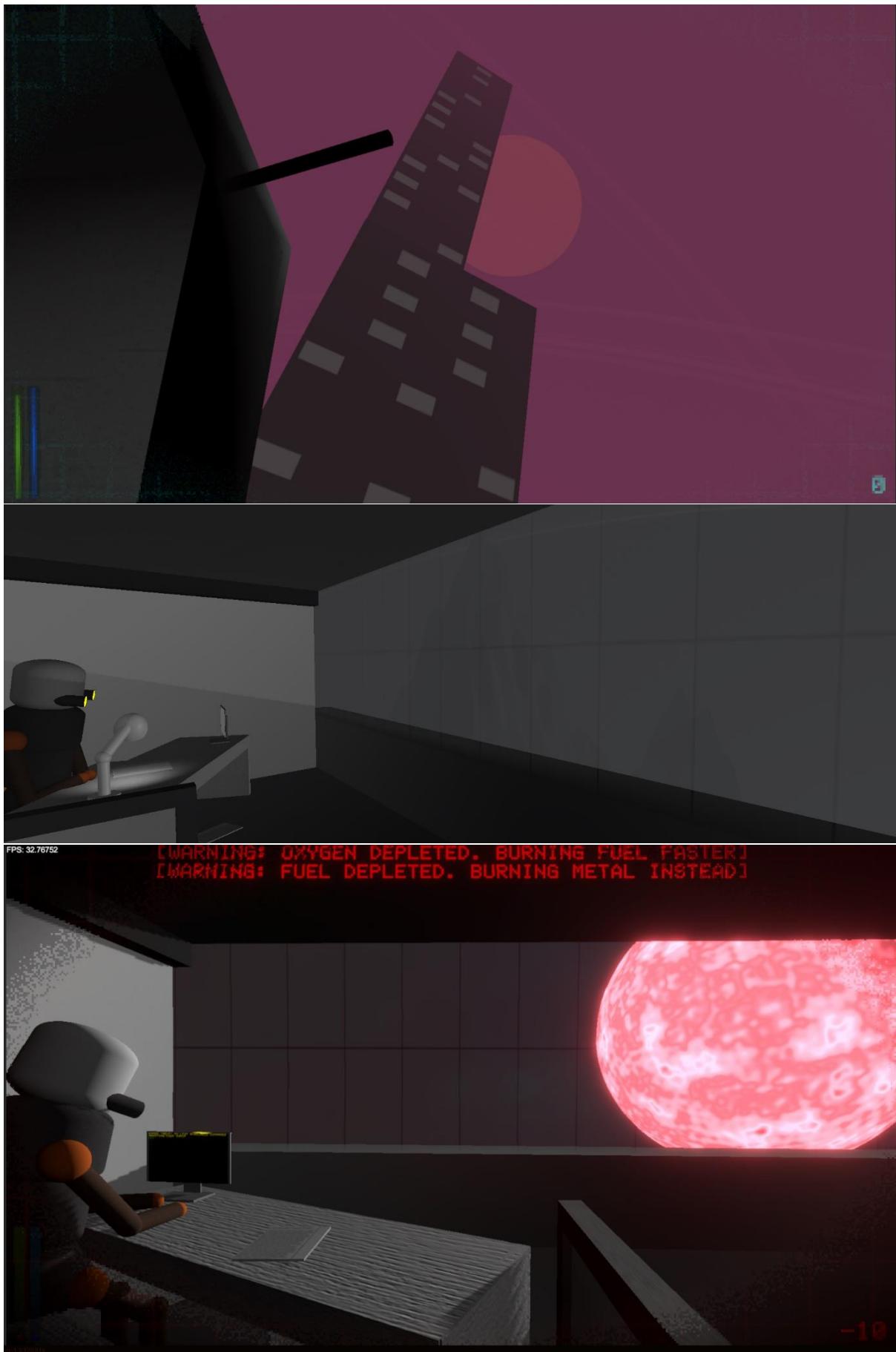
Select screenshots of the solution

I wanted to include these screenshots as a visual summary of Third Law, the game I have created.











89



