# Intro to Game Design: Coursework - Documentation

Lyall Campbell
S1429996

# Contents



# Synopsis

This game follows the journey of a worker of a space exploration team, who is abruptly awoken on a large island on a foreign planet with no other signs of life around. With no immediate communication with other life, the character must discover and explore what happened to the other workers whilst also taking into consideration that there is no natural oxygen and oxygen supplies are finite. The character later meets an exploration assistant robot which must assist the character on the journey and discoveries on the island.

# Gameplay and Story

## Wake Up Call (Intro Scene)

The introduction to the game and the island should be one of the most important parts of the game. The player and the character must be on an equal level in terms of whereabouts and situation since this is a game that revolves around mystery.

### Sleep Pod Room

The game begins with a black screen on the screen for 5 seconds before being abruptly removed, revealing what should be the sleep pod room to the player.

This room is in one of the wings of one of the space buildings situated in an isolated rocky area of the island. The entire building contains no windows therefore this room also contains no windows in order to preserve the mystery of what is outside to the player. This room, along with the entire building, uses the same clean white coloured structure that all the buildings use. However, this room should be lit up red by emergency alarm lights on the ceiling. A convincing alarm noise should be blaring for the duration of the time that the player is in the room to signify an emergency.

The cause of emergency is due to a lack of reserve oxygen being supplied to the room as a result of a system failure. The door will be sealed shut as a result of this, trapping the player in this room. This should be indicated to the player via the information feed bar in their HUD. The message will indicate that the player's suit oxygen is dangerously low, with only 35 seconds of suit oxygen left and that they should find an oxygen supply immediately. The player will be able to hunt around in the room for some way of regaining oxygen in this room, however there is no way of them regaining oxygen – it is scripted that their supply depletes. Using a script which updates the remaining time on screen, the remaining time should count down and then reach 0, indicating that the player has no oxygen at all.

At this point, the red lights should turn off, leaving the player suffocating in a dark room. 5 seconds later a green light will abruptly turn on above the door, signifying that the door is seemingly functioning now. 5 seconds later, an animation should be played which opens the door allowing the player to quickly escape into the airlock. Therefore, the entire door opening sequence should take 10 seconds.

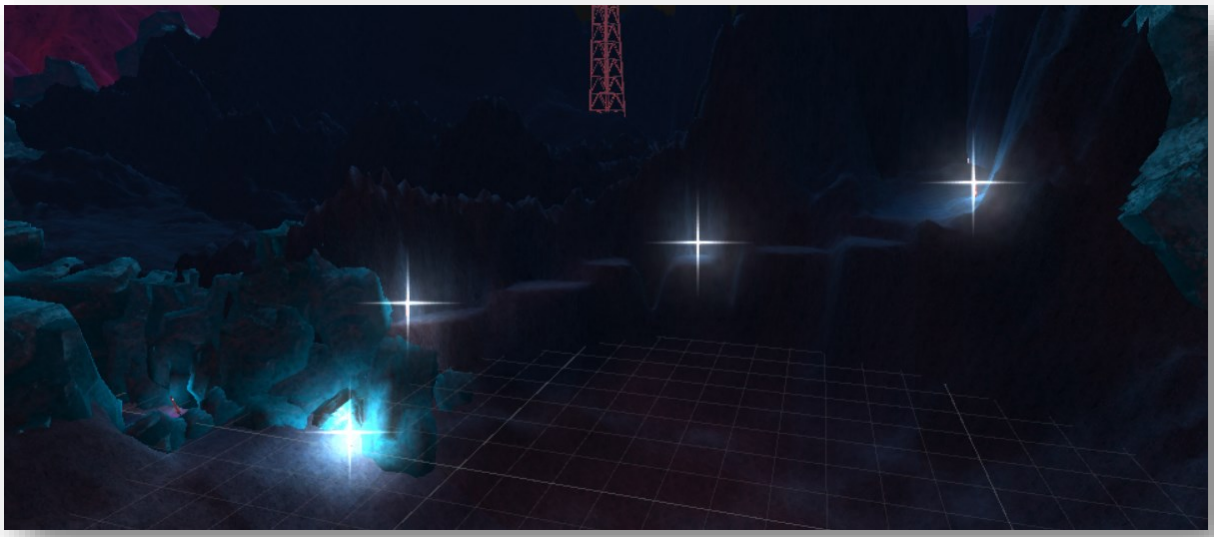
When the player enters the airlock, by using a trigger, the door should be closed promptly behind them, restoring oxygen to the player. This will leave the player in the airlock and there should be an oxygen canister item (which vanishes upon going next to it) on a chest inside the airlock. Indication that the oxygen can be picked up should be provided via the HUD in the form of a message since the oxygen timer is no longer needed.

The airlock's other door will open by playing its animation, allowing the player to enter the main hall room of the building. The player could explore the small interior of this area, before heading to the main door and activating it, leading them into another airlock. This airlock will follow the same timed procedure as the previous airlock before allowing the player to go outdoors into the section of the island.

## In The Open (Exploration)

When the player exits the building, they will discover they are in a small crater-like area with a narrow pathway in front of them. The pathway will have large blue cliff walls on either side, which make up the contour of the paths. The player should follow the path forward, until they reach an obstruction. The player can then turn right, and head up a slope to get around the obstruction.

At the top of the slope, the player will be standing at a point where they can look into and across a large crater below them. Light sticks are positioned around the opposite edge of the crater, leading up to an entrance at the other side of the crater. The player should head towards these, and climb up a sloped rock onto a part of the crater that sticks out. Platforms, as such, are positioned around this edge of the crater to which the player can progressively jump up in order to reach the entrance.



Once they reached this entrance, they must follow the long winding cliff path forward, leading to the final area of the level.

### The Fall (Level Ending)

After navigating through the winding pathway, the player arrives at a more open piece of land, separated by a large river stretching from left to right. Across the river is a single bridge - and at the other side a cluster of lights - which the player should be drawn towards. When they get around halfway across the bridge, their HUD should be abruptly changed and explosives on the bridge will detonate, dropping the bridge into the water, along with the player.

These explosives should be created as a prefab which uses an explosive particle effect along with a sound effect for the explosion. A trigger should be placed just before halfway along the bridge which should trigger the explosions and enable physics on the bridge.

A trigger should be placed on the surface of the water, so that when the player hits the trigger, the screen fades to black.

## Mechanics

### Player

#### Heads-Up Display (HUD)

A customised overlay, created in Photoshop, will be used to give the player a somewhat futuristic HUD visor which will be used to convincingly display information to the player and to show that the player is wearing a helmet as part of their suit. The overlay should be a semi-transparent image which will be applied across the user's screen via a HUD script component which makes use of the OnGUI method and stretches the image across the player's screen with screen.width and screen.height.

An important feature of the HUD should be three information lines, two at the top and one at the bottom. These should be managed via a HUD script and triggers which makes use of three public string variables which can be changed via the triggers where required throughout the level. In order to clear whatever information is displayed, by using the script's variables, the strings contents should simply be "".

The first top string should be used to let the player know their objective, location or information which is justified as it would later be revealed that the robot character is able to utilise the main character's HUD to give them instructions and objectives. This string should be positioned in the top middle of the screen, and should be relatively small in size since it shouldn't intrude too much on the player's screen space.

The bottom string should be used to let the player known immediate information and warnings. Again, this is justified since the suit, robot character and buildings can utilise this a way of providing immediate information to the character. Similar to the top string, it will be centred, but instead it will be placed down lower on the screen. Since this should be used to attract the player's attention, this text should be a lot larger and bolder than the top string since these are only temporary unlike objectives.

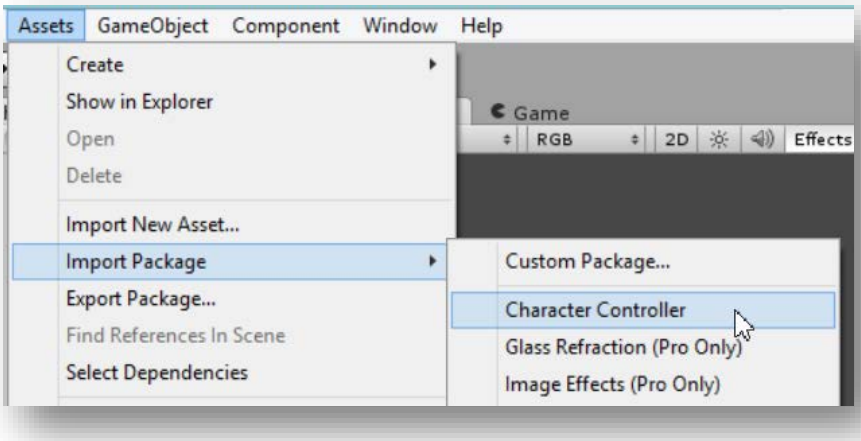
Another string, positioned directly below the top string, should indicate the character's remaining oxygen as a percentage. It should be noted that this is ultimately a pseudo representation of the character's remaining oxygen. The amount of oxygen will be adjusted throughout the game using specific triggers which simply change the string at various points in the game. This is to ensure that the player does not waste time which would completely deplete the remaining oxygen too early on which would result in the player not being able to advance further in the game. Essentially the purpose of the oxygen amount is to eventually give the player the objective of finding oxygen later on in the game.

During the opening scene and at the very end of the level, the player's screen will be blacked out. For the intro, this is achieved by drawing a black texture across the whole screen while a certain condition is met; when the timer is greater than the 35 seconds before when the player should wake up. For the end scene, this is simply done by changing to a new scene where the camera only displays black in its skybox settings. For if the game was going to continue directly from the fall, this would use the same system of drawing black on the screen and removing it after x amount of seconds, revealing the new location.

Altogether, the HUD, making use of 3D text objects which are parented to the camera and drawing a visor overlay over the screen, the HUD looks like:


## Low Gravity & Jumping

Handling the low gravity on the island is a simple procedure. This is achieved by making use of the First Person Controller prefab which can be acquired by importing Unity's built-in Asset Package called "Character Controller"




When the First Person Controller prefab is imported into the world, the prefab comes with the Character Motor script which handles specific movement features of the First Person Controller. This script conveniently uses public variables so that the First Person Controller can be tweaked with ease. In order to produce the low gravity aspect of the planet's island, the Gravity and Base Height had to be changed so that the player can jump higher but also fall slower. Additionally, the Max Forward Speed has been adjusted so that the player doesn't walk too slowly nor too fast. Changes indicated by bold.


This adds a reasonable change compared to standard first person movement therefore this mechanic is later used for puzzles and obstacles.

## Interaction

As a feature of the game, the player should be able to use certain objects. Traditionally games with this mechanic often use “E” on the keyboard to interact therefore this should be also used in this game too since player interaction is an important feature.

This can be achieved using a separate script component on the player which makes use of a raycast when the player uses the interaction key. Any interactive objects will need to have a script that confirms whether or not the object can be interacted with and its function.

Unfortunately this is not implemented fully in the created level.

Instead, excessive use of trigger zones is used and function simply by when the player enters an area.

For example, the oxygen script works using this method:

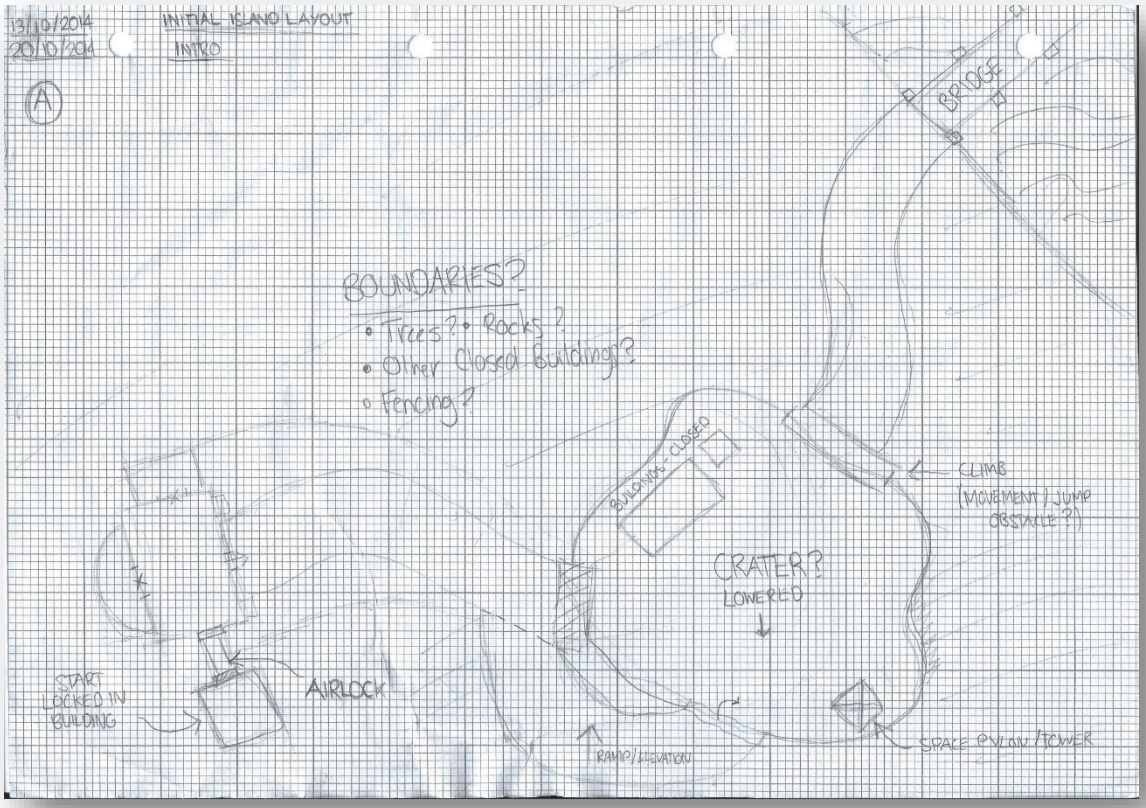
```
void OnTriggerEnter(Collider other)
{
        if (other.gameObject.tag == "Player")
        {
                GameObject userPlayer = GameObject.Find("First Person Controller");
                HUD hudScript = userPlayer.GetComponent<HUD>();
```

```
                hudScript.textTop.GetComponent<TextMesh>().text = "...";
                hudScript.textOxygenTop.GetComponent<TextMesh>().text = "[OXYGEN: 82%]";
                hudScript.textBottom.GetComponent<TextMesh>().text = "O2 REFILLED TO 82%";

                Destroy (playerHeartbeatSound.gameObject);
                Destroy (gameObject);
            }

        }
```

# Construction

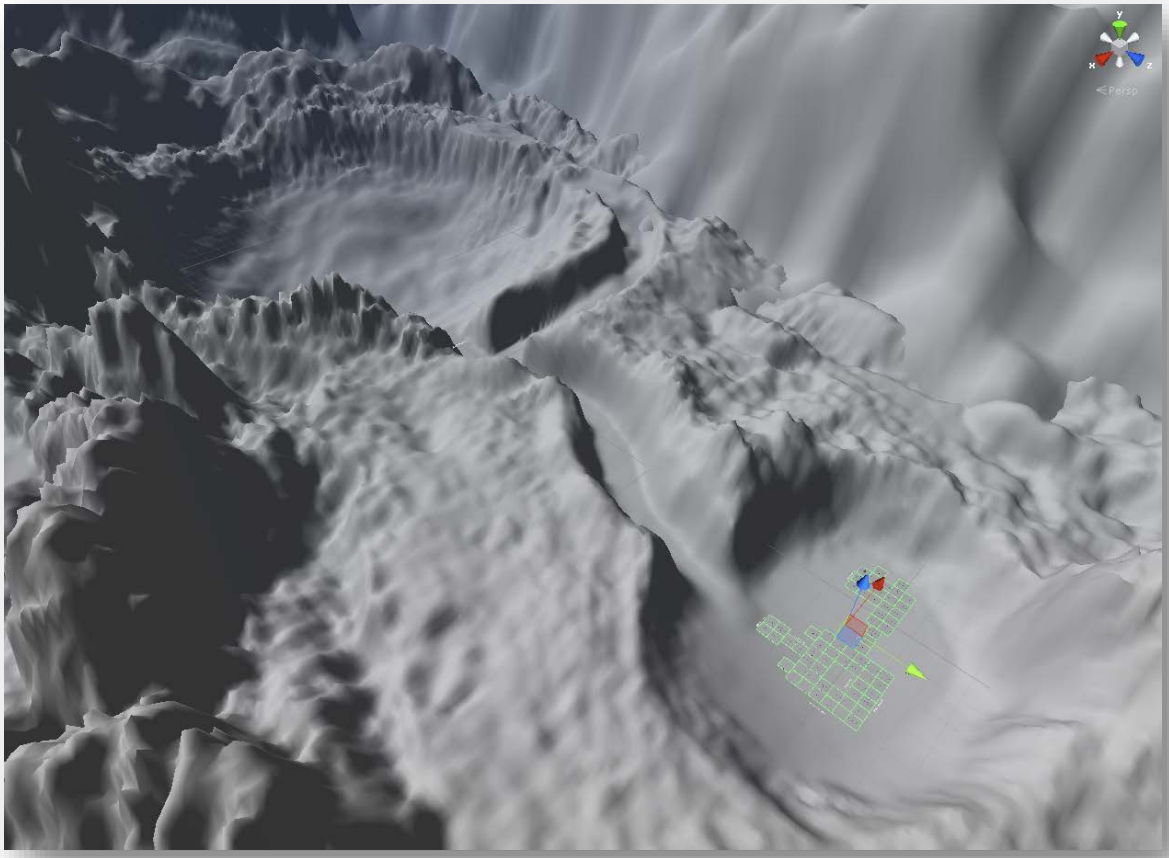
## Terrain

The base terrain of the level makes use of Unity's built-in terrain editor. The initial terrain – the terrain in which the player can navigate through should be a new terrain asset with default settings. Before beginning to edit the terrain, the floor for the intro room should be placed to ensure that the scale for the level is correct. This is achieved by importing the floor model in the assets, and quickly laying it out to match the layout plan. By using the layout plan for the level, the terrain should be based on the following layout sketch:




(*The buildings located inside the crater should not be included as part of the final construction.)*

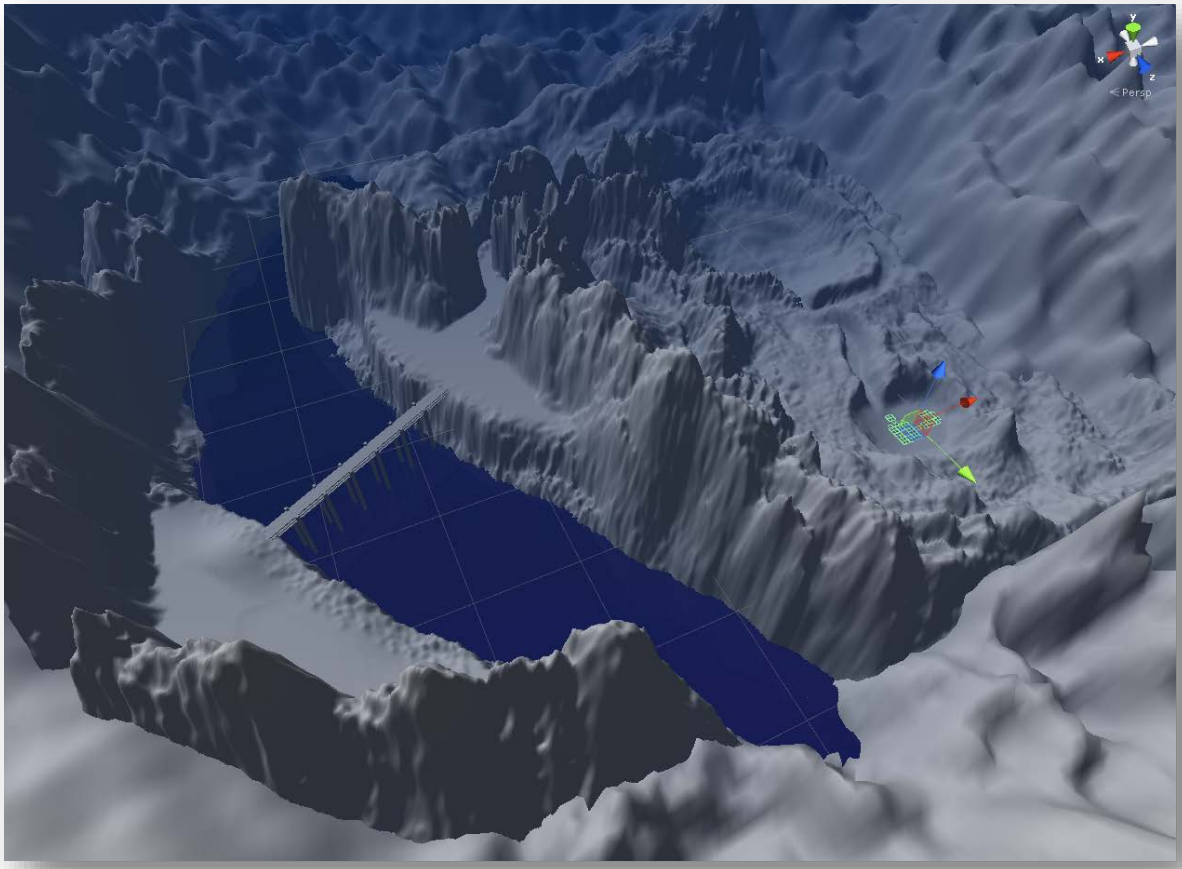
The path in which the player will follow should be reasonably flat so that the player can navigate it with ease. The rounded area in which the intro building is situated in should be reasonably tight, but allow a little space for when detail assets are introduced. The crater should be the largest open space of the level and should gradually dip towards the middle.

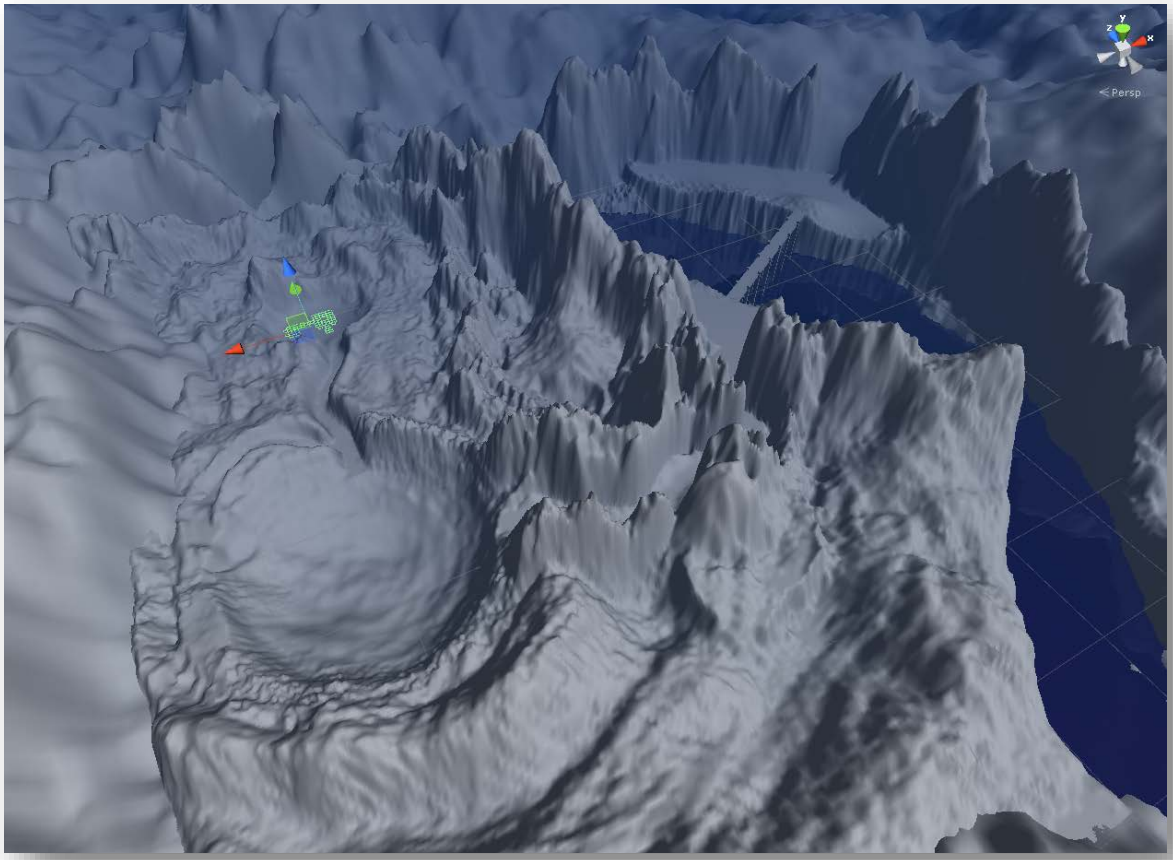


The highlighted part in the image indicates the area which is walkable inside the intro building. Once the terrain looks like the layout, extra detail can be added to it. This is achieved by painting the height on the outside of the paths and by creating large and tall mountains to hide areas that the player shouldn't immediately see too early on. This applies for the river since the player should not be able to see the river or the bridge until they actually arrive at it. As a result, a line of mountains is created on the outside of the path and along the river which blocks it from sight until you reach the river. The river itself should be done by lowering the height significantly so that a water plane can be added inside the gap later. To break up the terrain and to make it look more natural, non-rounded brushes can be applied very lightly (low opacity and strength) around the flat terrain.

When the first terrain (the foreground) is finished, a second terrain (the background) can be introduced. This is done by creating another new terrain and by adjusting the size in terrain settings so that the terrain is triple the size of the foreground terrain. The terrain should be manually repositioned using the transform tool so that the foreground terrain is positioned in the centre of the background terrain. Once this is done, mountains and rougher landscape should be created using the paint tool. It is important however that the background terrain does not clip into the foreground terrain therefore this should be done carefully. The background terrain should, however, merge slightly with the foreground terrain to avoid any obvious unrealistic gaps.

At an early stage, the terrain eventually looks similar to the following two images:

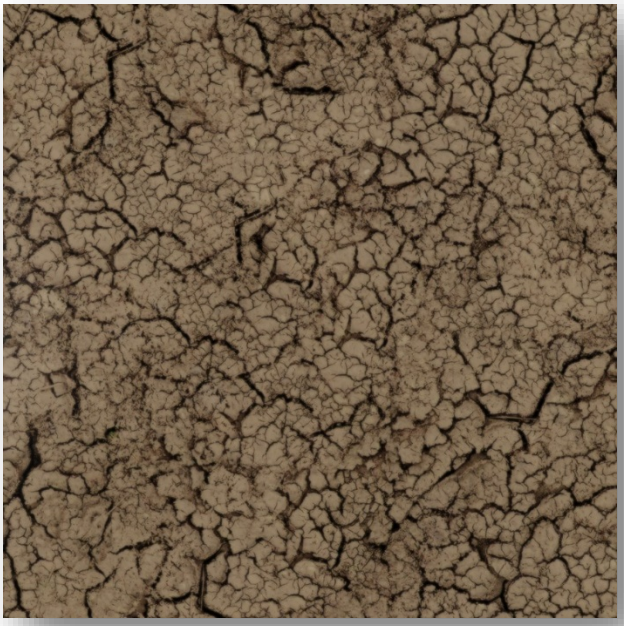

Persp



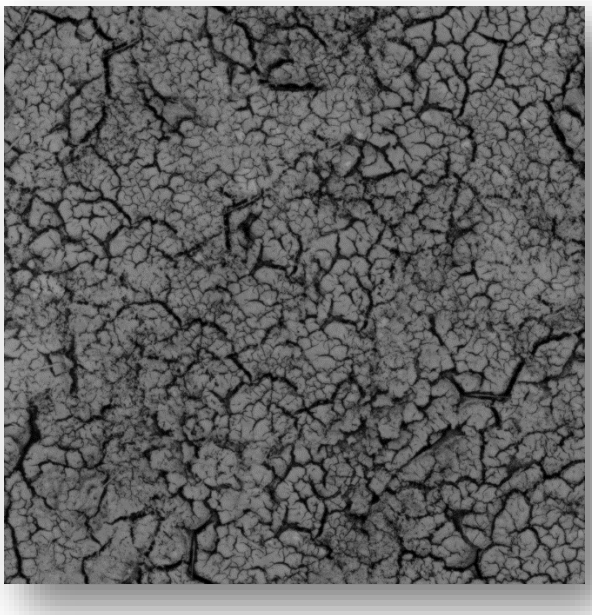

## Texturing

The texturing for this section of the island's terrain is somewhat straightforward but involves the use of Photoshop or a similar package which allows gradient maps to be used. Almost all the terrain texture assets which are used were taken from a free download of ground textures on the Unity Asset Store (credited at the end) which were later quickly modified in Photoshop to meet the colour scheme.

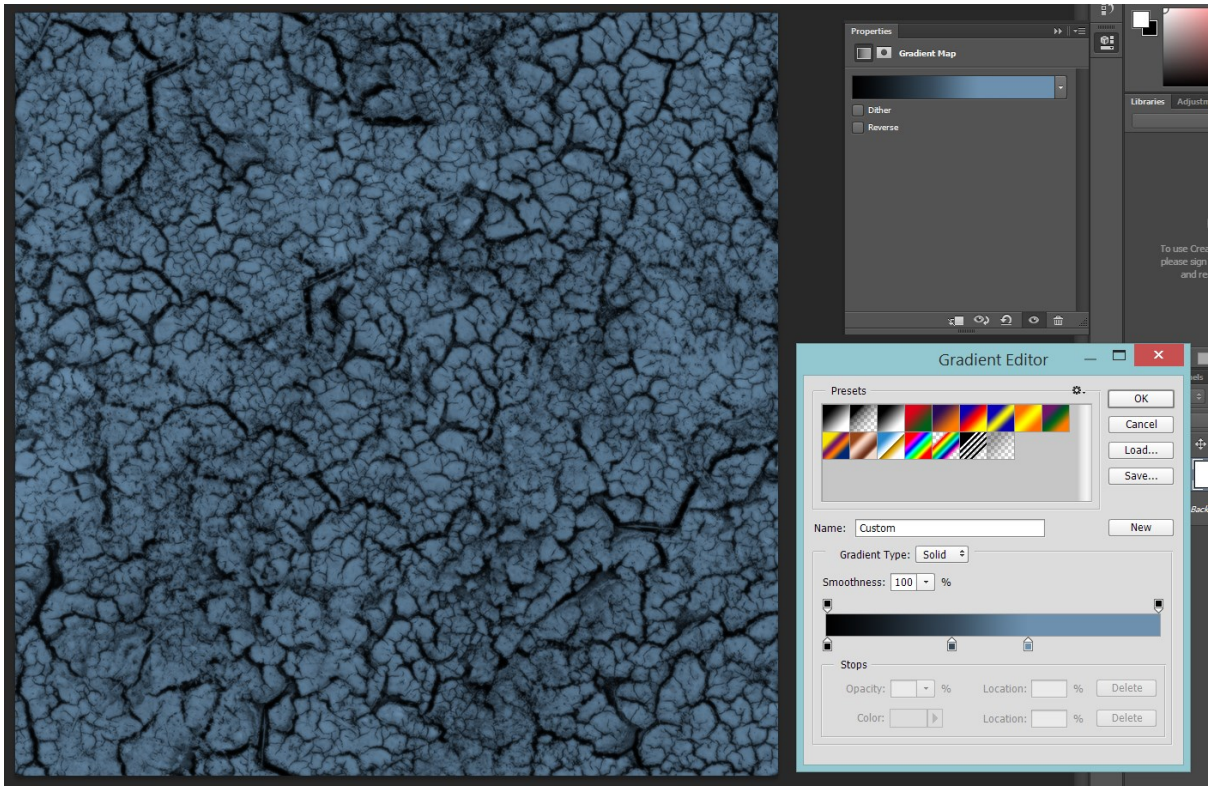
For example, this is the default texture that was downloaded:



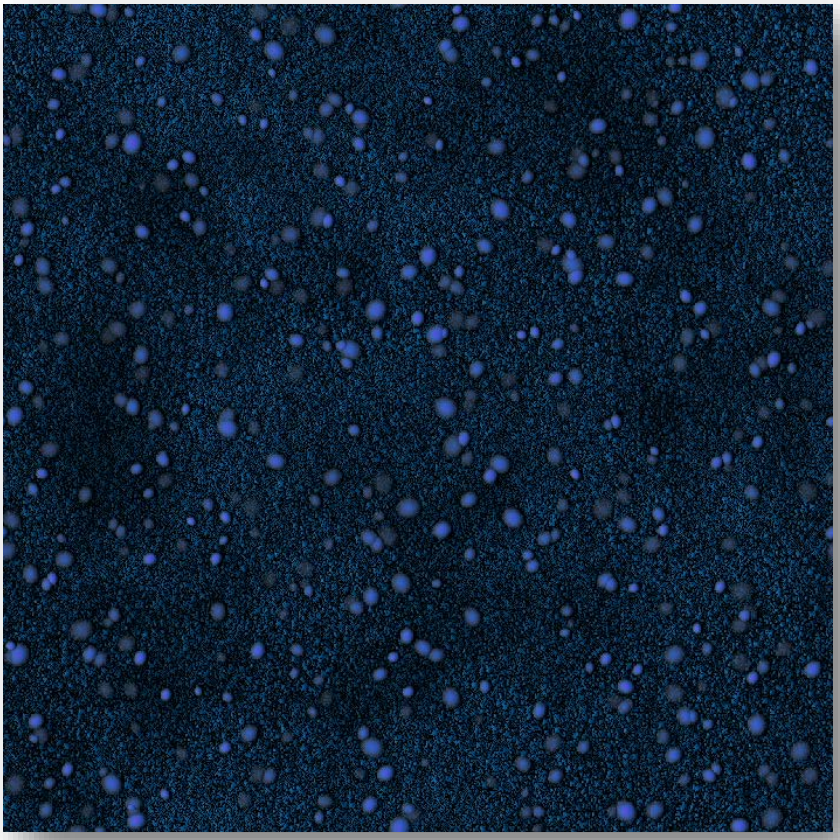
A gradient map was applied to this image in Photoshop, which essentially converts the image into greyscale, allowing the user to input where colours should be replaced between the black to white colours. Shades of blues were used for this texture which changes it from looking like a natural Earth-like ground to one that is more fitting with the island's design and colour scheme:

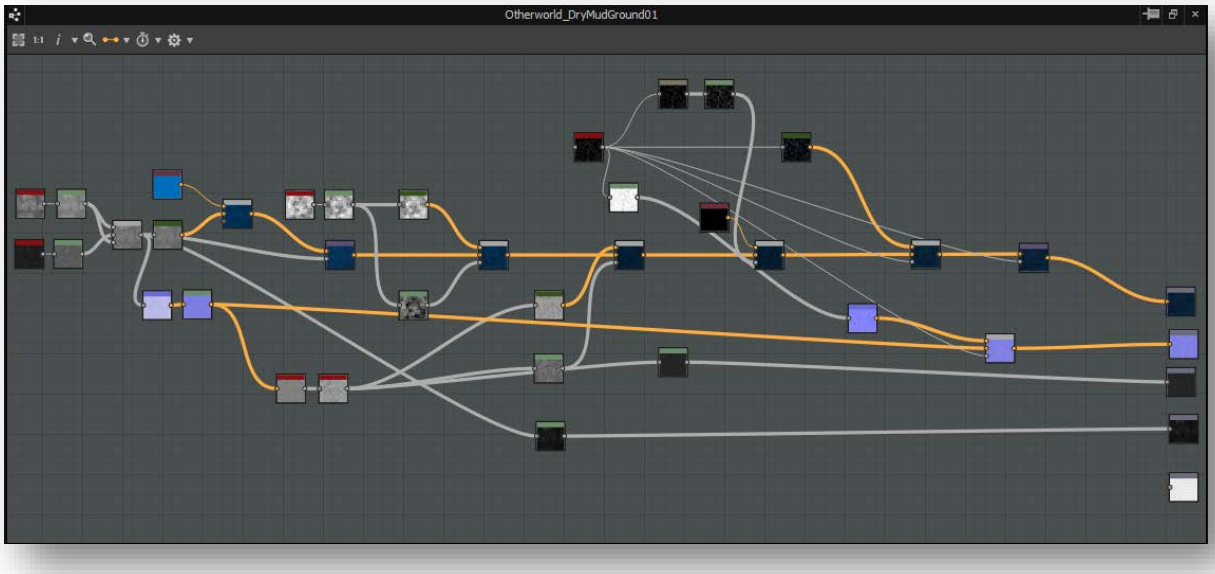


Once the gradient map colours were set, this particular texture looked like the following image. This procedure was done a few of the other ground textures assets that came with the downloaded pack. Not all of them were suitable for this particular terrain, therefore only a dry ground, ground and moss and another ground texture was used in the final textured terrain.




One texture was specially created for the island and for this terrain, which was created from scratch using Allegorithmic's Substance Designer. The reason for using this tool is that although the colour change scheme somewhat gives the impression that this world is not Earth, it needed more unnatural terrain to make it more fitting. As a result, by using Substance Designer, new textures can be created by utilising and combining various nodes to produce texture outputs. Substance Designer allows normal maps and PBR compatible maps to be exported, however, due to the limitations of Unity's terrain texturing, only the diffuse can be used. If need be, Substance Designer allows Substance files – which are compatible directly with Unity – to be exported and imported into Unity where materials and textures can be adjusted in real time inside the Unity editor.





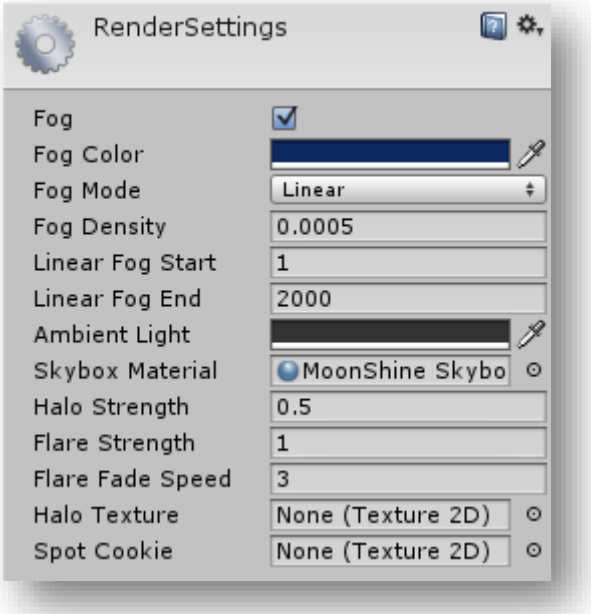

In total, four textures were used to paint the terrain. The texture that was created in Substance Designer was applied as the base material for the terrain. Afterwards, the dry rock texture was applied around the edges where the rock walls meet the terrain. It is also lightly applied in the crater to break up the textures through slight blending.


In addition to this, the terrain was elevated slightly along the edge of the rocks so that the meet more nicely.

Overall, through use of the textures on the terrain, it eventually looks like this:


## Fog & Skybox

To break off the terrain in the distance, coloured fog is enabled under Render Settings using the following values:




Since this is set during the night, a suitable skybox had to be imported. This was achieved by importing Unity's build-in skybox asset pack to which the "MoonShine Skybox" was chosen for this level.

## Detail Objects

To populate the level, detail objects are introduced and used at various points throughout the level. Some of these can be acquired for free from the Unity Asset Store, however, for particular specific objects, they have to be modelled manually.

### Rocks & Cliffs

The rocks and cliffs models used are potentially one of the most important assets for the construction of this section of the island. In order to give an impression that this was a more secluded and remote area of the island, a general mountainous landscape had to be created. Additionally, these assets also had to fit in with the colour scheme.

The rocks and cliffs used in the scene were downloaded and imported from a free download on the Unity Asset Store. By default, however, these models were textured with natural colours and therefore immediately unsuitable for the island. Thankfully, the same technique for changing the colour of the terrain textures can also be used for this.

The diffuse textures for the particular models that were going to be used were brought into Photoshop where gradient map was used once again to change the textures blue. This image was saved and automatically reimported back into Unity where it could be used.

The construction of the pathways and giant overhanging cliff was achieved by reusing a few of the same models repetitively. These models were copied and pasted many times, then repositioned to fit the terrain or to create obstructions. In order to make it look more natural and to make it appear as if the rocks are somewhat unique, many were either rescaled or rotated and overlapped with other rocks.

Eventually, after using many instances of these models throughout, the scene was finally given a dense rocky environment that broke apart some of the bland areas of terrain.


### *Obstruction*

These rocks are used as walls for the pathway, therefore the player is unable to go outside of the path. However, since the player is expected to be able to see into the crater but be obstructed, smaller rocks are used in clusters to create this obstruction. Whilst this may provide an obstruction that fits with the environment, the player will be able to get over them due to how gravity and movement works.

As a workaround, an invisible wall must be created at these rocks. This allows the player to climb up the rocks as much as they can but ultimately prevents them from getting fully through:


By creating a new cube object, resizing it and disabling its Mesh Renderer, it acts as an invisible barrier to the player.

## Oxygen Container

The oxygen container is one of the objects that had to be created specifically for this environment. Since the player has to acquire an oxygen refill of some form, it was vital that there was an object to represent this. A canister shaped model with specific connectors and top and bottom was created in Blender and later imported into Allegorithmic's Substance Painter in order to quickly texture it. Once the textures and mesh was exported out and imported into Unity, the material was quickly created by creating a new material that uses the "Diffuse" shader and by placing the diffuse texture in the respective box:

## Crates

The crates used in this level are from a free download pack on the Unity Asset Store. This pack provides around 8 different low-poly crates which can be resized and positioned to appropriately fit the scene.

## Grating

The grating that is used on the walls and on the flooring of the airlock rooms was created by simply using the downloaded crates pack. This pack comes with a crate which has a grate-like texture. By bringing in this particular model into the scene, it can be resized to make it long and flat so that it can be repositioned into the square indents on the flooring and the walls.

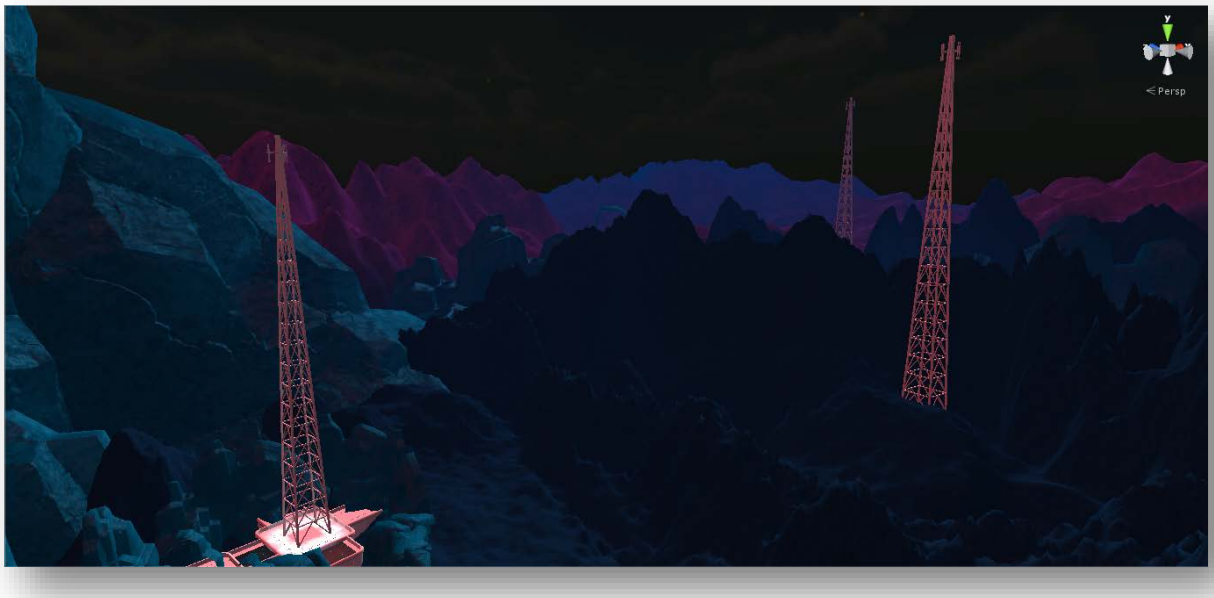
## Light Poles

The light poles are another asset that was specifically made for this level. These too were created in Blender and imported into Substance Painter in order to texture it quickly. The diffuse + gloss map and normal map was exported out this time, and the material later created in Unity makes use of the “Bumped Specular” shader. The diffuse + gloss map and normal map was assigned to the material in its respective slot. The material was applied to the imported mesh inside of Unity and rescaled so that it can be used in the scene.

A point light object was parented to each instance of this object and the light was repositioned towards the top section of the model. The intensity and range of the lights were adjusted per instance of the object with respect to the area in which they are positioned in. Afterwards, the lens flare package – which Unity has by default – was imported into the project. The “Small Flare” was applied to this object.

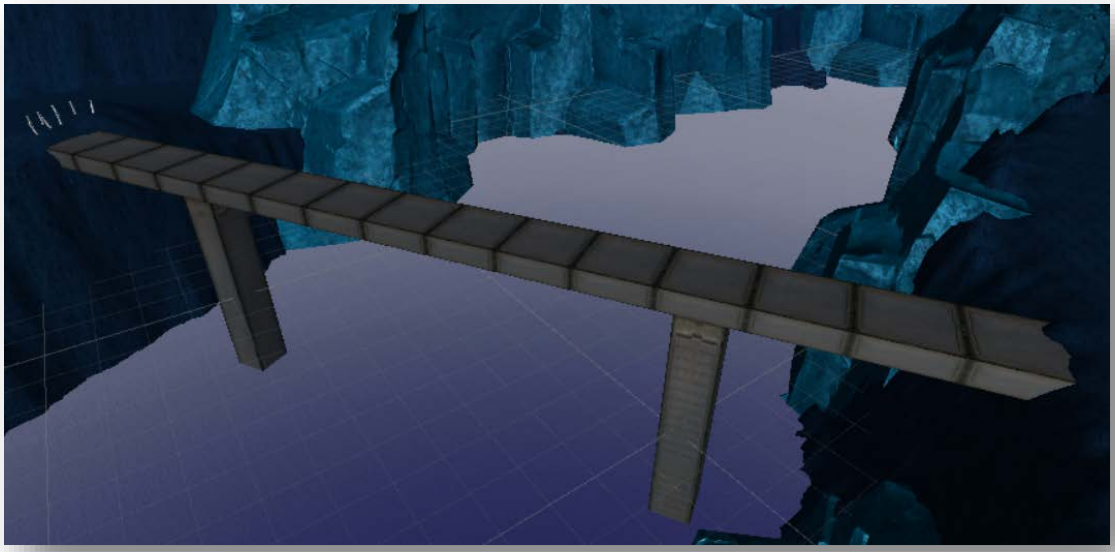
### Communication Towers

The communication / radio towers used in the scene were downloaded from the Unity Asset Store. When brought into the scene, by default, they have a striped red and white texture. Since this does not fit with the overall white theme of the buildings and architecture, the texture was simply deleted in the assets browser. These were introduced into the scene for background detail and to potentially show that communication was possible between buildings on the island / planet.



### Bridge

The bridge is the last important detail object of the level, which is used as a mechanic for concluding the level. The bridge was constructed using long flattened out instances of one of the crates in the previously mentioned package. This was duplicated several times across the length of the river, and two tall thin instances were created and placed appropriately. The explosives are to be placed on these supports so that the middle section of the bridge is dropped into the water. This is achieved by adding a rigidbody through a script when the player enters a trigger. The addition of the rigidbody means that the bridge parts are affected by physics and thus fall down along with the player. The explosives are created by importing the built-in Particles package into Unity. The explosives by default are disabled, and are later activated using SetActive in the bridge script.



## Space Buildings

### Structure

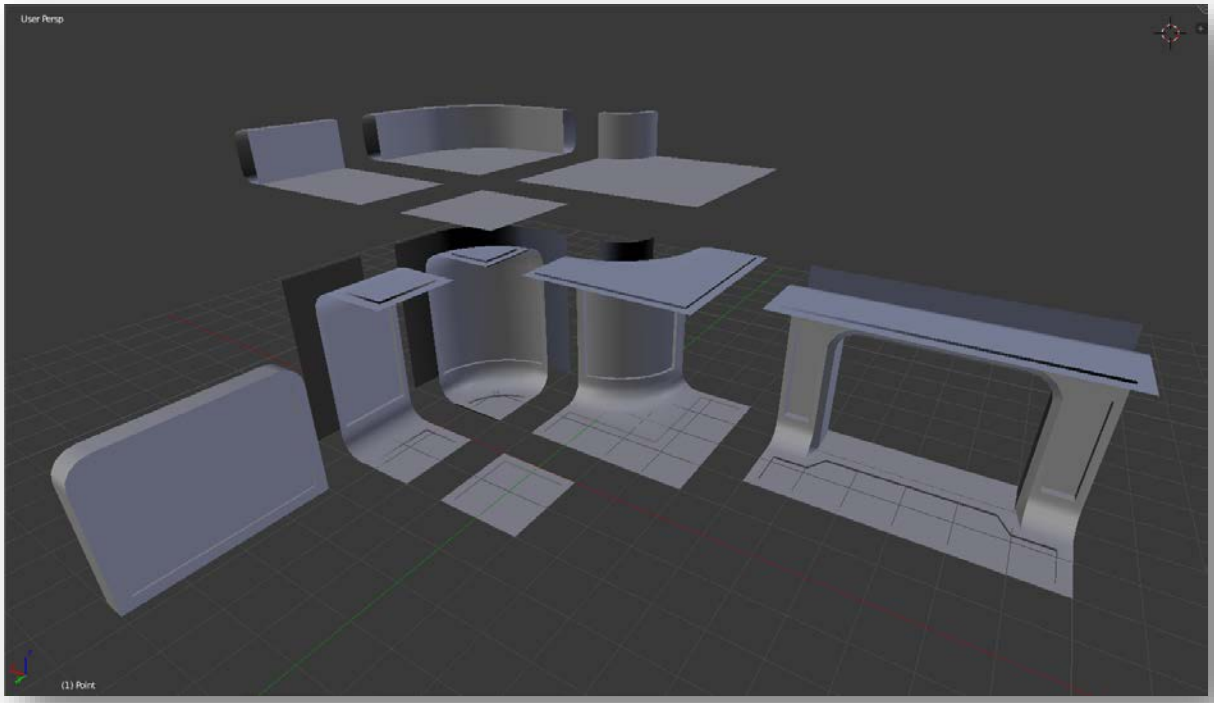
The quick layout of the intro building was created to initially preserve scale for the terrain, but also to give an idea of how the building should be formed. Fast tile floor models were used to create this layout. Afterwards, it was labelled appropriately:


In order to save time and to make it more convenient for the project, custom modular meshes should be created to form any basic space buildings for the level. These can be created in most 3D modelling packages. For this example, Blender was used.

The meshes were simply created by building singular walls, inner corners, outer corners, door frames and flooring by sticking to Blender's grid and by creating these pieces as separate objects that were able to fit seamlessly together. To ensure that they did fit together, the editing process should be repeated almost exactly the same for each piece. For example, if the singular wall was extruded up by 2 units at the side, then all other parts which connect onto this wall by the side should also be extruded up 2 units. A similar approach was repeated once again for adding minor detail to the walls by using the inset function on the ground face and the wall face. Whilst using inset, the same values were used across all the pieces that had it, and the inset faces were also extrude inwards by the same amount to ensure consistency.

Afterwards, roof pieces were created which would be placed on top of the walls and corner pieces so that the player cannot see through the roof at points where they can see the top of the buildings. Additionally, if the player was required to stand on top of a building, they would be able to do so. These roof pieces were produced by copying the respective wall piece, flattening it out and adding a thick border around the outside. The reason for the thick border is to give the building a sense of robustness and the roundness of these borders helps work towards the space feel of the building.

Once the pieces were finished, they were appropriately UV mapped so that they could be textured if required. However, due to the nature of these particular meshes and their use, a solid material will suffice later in Unity.




As shown by the meshes in the picture, they are all created to fit the grid. The sizes of each piece are as follows:

- Single Wall: 1 x 2
- Floor: 1 x 1
- Outer Corner: 2 x 2
- Inner Corner: 2 x 2
- Door Frame: 3 x 2
- Door: 2 x 1 *the door's positioning has been adjusted to fit the door frame*

(The top pieces are also the same size of their respective counterpart.)

When the pieces were all acceptably finished, they were repositioned one by one to the origin in Blender and then exported out individually and named as "sb_" followed by its type. For example, sb_wall, sb_innercorner, sb_outercorner_top, etc. From there, they were each imported into Unity into a folder containing all of the space building assets. Since the meshes, by default, are extremely small, each piece had to have their scale adjusted to 2 in the properties menu.

Once this was done, the meshes could be brought into the level. One important thing to note is that when the objects are brought into the level, they will not be positioned correctly and thus they will not be able to snap and fit effectively. In order to fix this, when the object has been brought into the world, the object's transform location must be set to a whole number for its x, y and z values. This allows the objects to fit to Unity's grid and therefore objects can be moved around and snap to the grid whilst holding ctrl.

Since space walls are traditionally clean and white, these meshes do not need to be textured – however, they can if the buildings are to be given a more polished and realistic feel. Instead, a new material can simply be created in Unity.

Very quickly, by using this approach, quick mock-ups of buildings can be created:

Due to the sizes of these pieces, the initial layout was not able to be recreated exactly. Nevertheless, the general layout and features were still implemented where size allowed.

## Interior

Once the building had been formed, work could be focussed on the interior and its lighting since a good amount of scripted gameplay is implemented in this area.

### *Sound Effects*

To add to the feel of the scene, audio should be introduced into the scene. For the intro scene, a total of 3 different sound effects are used. Each is an empty game object with an audio source component attached to it. The three sound objects should be:

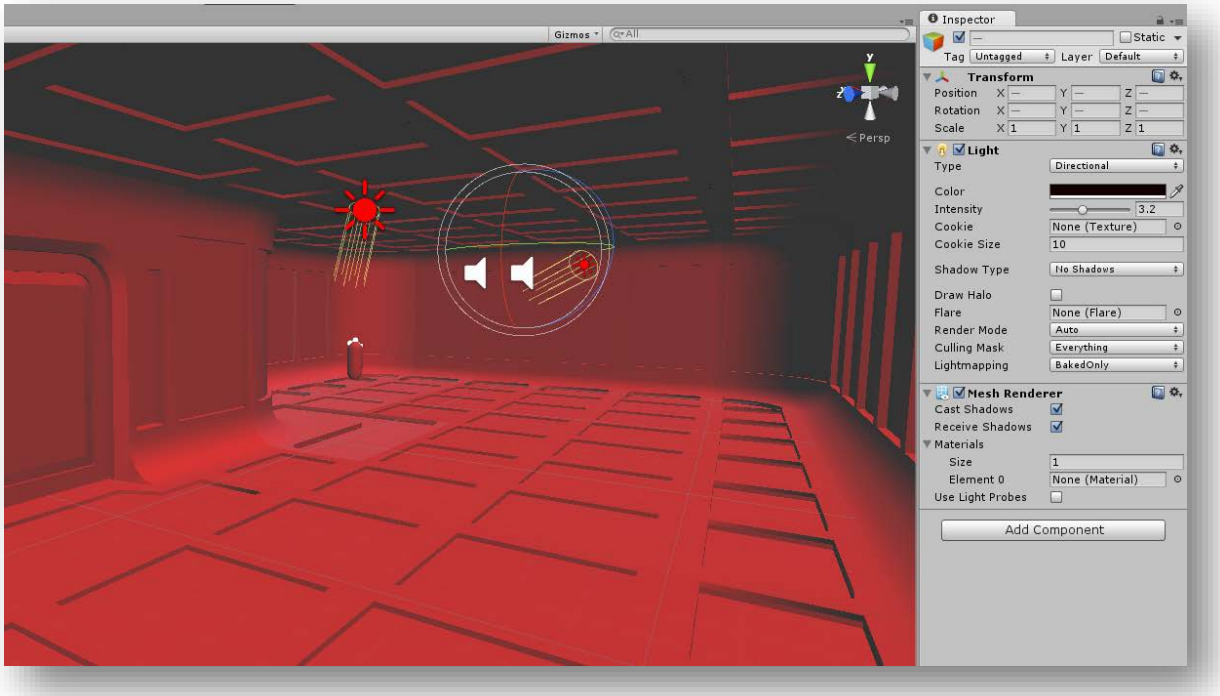
- "AlarmSound" with alarm sound clip:
    - 3D sound
    - Play on Awake
    - Loop
    - Volume: 0.4
- "PlayerHeartbeat" with the heartbeat sound clip:
    - 2D sound
    - Play on Awake
    - Loop
    - Volume: 0.4
- "PlayerBreathing" with the air swoosh sound clip:
    - 2D sound

- Play on Awake
- Loop
- Volume: 1

These are set up so that they can be either assigned as a GameObject to public variables on a script or to later destroy them completely.

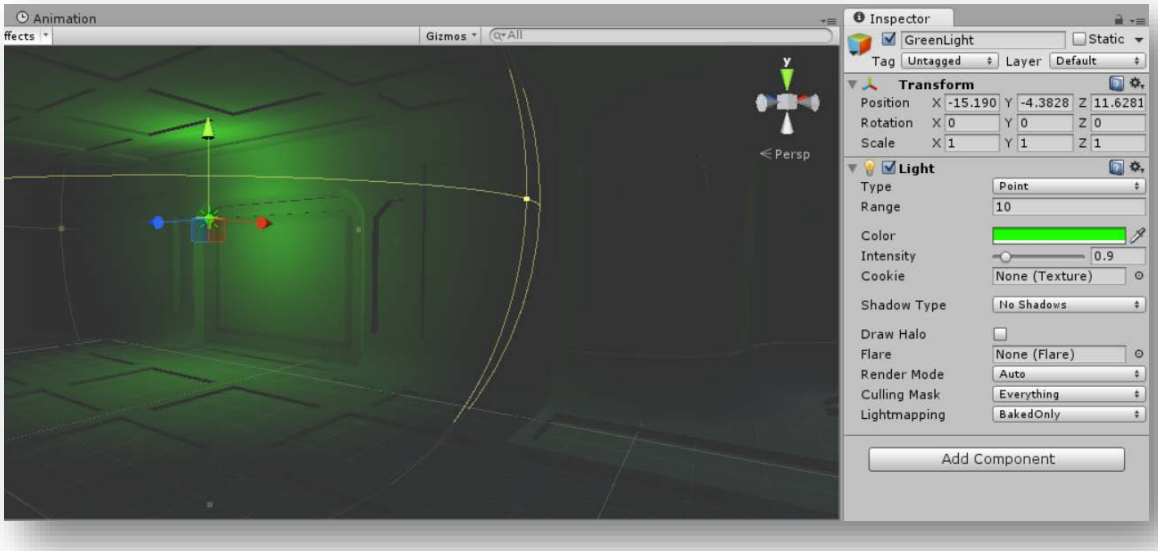
### *Intro Scene*

Firstly, since the intro room has to be during a time of alarm, two red directional lights were brought into the room so that they can light up almost the entirety of the room. This was achieved by rotating each light so they faced opposite walls. They were appropriately named “AlarmLight01” and “AlarmLight02” for organisation and the light colour was changed to a very dark red. The intensity was also changed for testing purposes since the intensity would be handled by a script later on which would create a strobe-like effect.




Afterwards, the red lights were temporarily turned off, and a new green point light was introduced to the scene which would represent the airlock door opening during this scene. The light was brought in and positioned in front of the door. This light did not need to be overly bright since the red lights were scripted to deactivate completely once the timer runs out, sending the room into complete darkness. The player’s lower text on their HUD should display “NO ATMOSPHERE DETECTED” Shortly afterwards, the green light will turn on in the dark room. As a result, the green light sufficiently lights up the door and part of its surroundings, drawing the player towards it as the airlock door slowly opens. At this point, the alarm sound object is destroyed and the breathing sound is stopped. The heartbeat sound continues, and the volume is increased to 1.

The correct properties and effect of this light are shown in the image below:




When the airlock door is open, the player may enter the airlock where the door behind them will close quickly behind them thus beginning the airlock transition. (Although the objective is trivial, if the player fails to enter the airlock after 30 seconds then the level should restart from the beginning.) The player's oxygen HUD text will be updated to display to them "AIRLOCK TRANSITIONING…" and their lower text should now display "ATMOSPHERE DETECTED" during the transition.

Inside the airlock room will be a set of crates with an oxygen container on top of one of them. A small blue light should be attached to this object to draw the player towards it. The object will be destroyed along with the light when the player has entered its trigger zone and the oxygen part of the HUD should be updated to show that the player now has 82% of reserve oxygen remaining. The heartbeat sound will also be destroyed.

The second door will slide open, allowing the player to enter the main hall where they can enter the next open airlock on their right. This airlock will simply close the door behind them, turn on the main light in the world (which is part of the intro room script) and then open the door, allowing the player to exit out into the world. The HUD top text will also be changed to display "ENTERING ISLAND SECTOR A1"

### *Intro Room Script*

The entirety of the intro script is given below:

```
using UnityEngine;
using System.Collections;

public class Intro_Room : MonoBehaviour {

        public GameObject redLight1;
        public GameObject redLight2;
        public GameObject greenLight;
        public GameObject alarmEmitter;
        public GameObject playerHeartbeat;
        public GameObject playerBreathing;
        public GameObject airlockDoor;
        public float introTimer = 45;
```

```
	bool doorOpen = false;
	public bool playerAwake = false;

	// Use this for initialization
	void Start () {

	}

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

		// For the first 35 seconds after the player is awake, countdown based on introTimer - 10.
		if (introTimer <= 45 && introTimer >= 10)
		{
			GameObject userPlayer = GameObject.Find("First Person Controller");
			HUD hudScript = userPlayer.GetComponent<HUD>();

			hudScript.textBottom.GetComponent<TextMesh>().text = "O2 TIME REMAINING: " +
Mathf.Floor (introTimer - 10) + " SECS";

			playerAwake = true;

		}

		// When the player's countdown has ran out, adjust HUD text and turn off red lights.
		if (introTimer <= 10 && introTimer >= 9)
		{
			GameObject userPlayer = GameObject.Find("First Person Controller");
			HUD hudScript = userPlayer.GetComponent<HUD>();

			hudScript.textTop.GetComponent<TextMesh>().text = "...";
			hudScript.textOxygenTop.GetComponent<TextMesh>().text = "[OXYGEN: DEPLETED]";
			hudScript.textBottom.GetComponent<TextMesh>().text = "NO ATMOSPHERE DETECTED";

			redLight1.SetActive (false);
			redLight2.SetActive (false);

			/// Remove alarm, silence breathing and increase heartbeat volume.
			Destroy(alarmEmitter.gameObject);
			playerBreathing.audio.Stop();
			playerHeartbeat.audio.volume = 1.0f;

		}

		// Five seconds later, adjust HUD text and turn on green light.
		if (introTimer <= 5 && introTimer >= 4)
		{
			GameObject userPlayer = GameObject.Find("First Person Controller");
			HUD hudScript = userPlayer.GetComponent<HUD>();

			hudScript.textTop.GetComponent<TextMesh>().text = "OVERRIDING DOOR";

			greenLight.SetActive (true);

		}

		// When the timer is done, begin opening the door.
		if (introTimer <= 0)
		{
			if (doorOpen == false)
			{
				if (airlockDoor.transform.position.y >= -3.2)
				{
```

```
                                        airlockDoor.transform.Translate (Vector3.down * 0.02f, Space.World);
                                }
                                else
                                {
                                        doorOpen = true;
                                }
                        }
                }
                else
                {
                        // Deduct the time from the intro timer based on delta time.
                        introTimer -= Time.deltaTime;

                        // Adjust the intensity of the red lights based on time for a strobe effect.
                        redLight1.gameObject.light.intensity = 8 * (Time.time - Mathf.Floor(Time.time));
                        redLight2.gameObject.light.intensity = 8 * (Time.time - Mathf.Floor(Time.time));
                }

        }
}
```

### *HUD Change Script*

Although this particular script is used for separate trigger zones where the player's HUD should be changed, essentially the HUD change is achieved overall through the following:

```
using UnityEngine;
using System.Collections;

public class HUDTextChange : MonoBehaviour {

        public string topText;
        public string oxygenText;
        public string bottomText;
        public bool shouldChangeTop;
        public bool shouldChangeOxygen;
        public bool shouldChangeBottom;
        bool playerEntered = false;

        // Use this for initialization
        void Start () {

        }

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

        }

        void OnTriggerEnter(Collider other)
        {
                if (playerEntered == false)
                {
                        if (other.gameObject.tag == "Player")
                        {
                                GameObject userPlayer = GameObject.Find ("First Person Controller");
                                HUD hudScript = userPlayer.GetComponent<HUD> ();

                                if (shouldChangeTop == true)
                                {
                                        hudScript.textTop.GetComponent<TextMesh> ().text = topText;
                                }
```

```
                                        if (shouldChangeOxygen == true)
                                        {
                                                hudScript.textOxygenTop.GetComponent<TextMesh> ().text =
oxygenText;

                                        }

                                        if (shouldChangeBottom == true)
                                        {
                                                hudScript.textBottom.GetComponent<TextMesh> ().text = bottomText;
                                        }

                                        playerEntered = true;

                                }
                        }

                }
}
```

## Acknowledgements

The following is a list of resources made by others which were used for this project. Their name / alias and source URL is given where possible:

Models:

- "Radar Tower" – by Unity Technologies
  - https://www.assetstore.unity3d.com/en/#!/content/821
- "Metal Crates Pack" – by Eldanu Games Studio
  - https://www.assetstore.unity3d.com/en/#!/content/8836
- "Ground Textures Pack" – by Nobiax / Yughues
  - https://www.assetstore.unity3d.com/en/#!/content/13001
- "Free Rocks" – by TripleBrick
  - https://www.assetstore.unity3d.com/en/#!/content/19288

Audio:

- "01586 air swoosh.wav" – by Robinhood76
  - http://www.freesound.org/people/Robinhood76/sounds/94841/
- "Single Heartbeat Clean HQ_BeatSmith.wav" – by Lunardrive
  - http://www.freesound.org/people/Lunardrive/sounds/22440/
- "alarm_fatal.wav" – by sirplus
  - http://www.freesound.org/people/sirplus/sounds/25031/

Applications:

- Unity Version 4.5.5f1
- Adobe Photoshop CC (2014)
- Blender 2.72
- Substance Painter 1.0.2
- Substance Designer 4.5