# Documentation

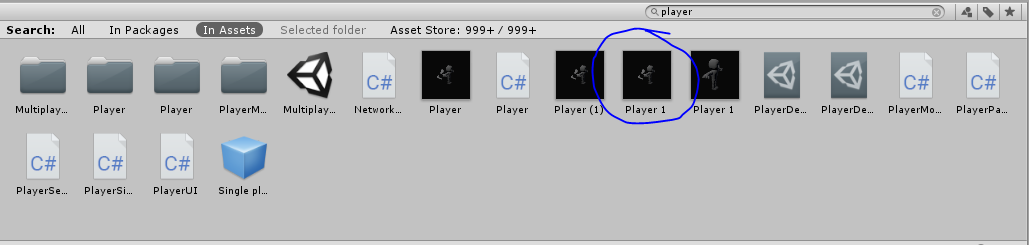
## High Definition Render Pipeline (HDRP)

This is an important part of the game. This game now runs on the High Definition Render Pipeline (HDRP), so before implementing anything into the game, just keep this in mind. I won’t go into too much detail about what HDRP is, other than it is just a render pipeline designed for users who want better graphics. If you want to know more about it, read the Unity documentation on it. It’s really just a way to get some great looking lighting and high quality assets.

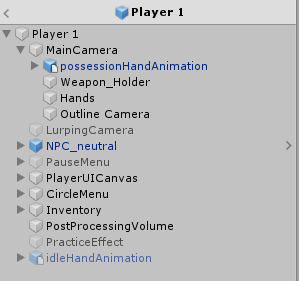
## Quick heads up about Player Prefabs

Unfortunately, we did not have enough time to clean-up some of the prefab names from the previous group of students, so instead I will show you which player prefab is the important one for development, and then you can figure out what to do with the rest.

If you search “player” in Unity’s project search area, you will be displayed with 4 player prefabs (see image below). The circled prefab is the prefab you should use for development, as it is the local player that is spawned when a player connects.



Opening this prefab will look like the image below. This is the layout for how we handle our players. The main camera is the camera that each player sees out of. The important stuff is the pauseMenu, PlayerUICanvas, CircleMenu, Inventory and PostProcesssingVolume. I will be explaining these features below.



As a last note, the player 1 prefab is stored in the prefabs folder in the assets folder.

## UI / HUD

For the game to have a user interface (UI) and a heads up display (HUD) for each individual player, we have setup these elements on the player prefab, rather than in the Unity Editor. This is due to the game being multiplayer. If we store our UI elements in the “Player 1” prefab, then every time a new instance of a player is created (when a new player logs into the sever), then it creates a new UI environment for that specific player. As you can see in the image above from inside the player 1 prefab, I have each of the UI / HUD items in their own drop down section. These different sections are normally their canvas for the elements to display on. A canvas, for anyone who doesn’t know, is what you link the camera to that it will display on. This essentially controls what player sees what information in regard to the multiplayer.

PauseMenu controls the pause menu. This needs to be reworked as the quit button stopped working right before we finished development. This should be an easy fix. It also needs a settings menu at some point. This pause menu is controlled by an animation when it opens. This animation is stored in the Animations folder in the assets folder.

CircleMenu controls the menu canvas that displays when a user opens up their inventory. This inventory has 4 slots, but if you play around with the code and the image, youll be able to have it have as many slots as you wish. This can be achieved by doing some radial math, and having the image cut to the right degree amount. It was originally designed as just a placeholder inventory circle menu, however it seems to be staying in the game for now. Feel free to scrap it for a better menu, I never intended for it to actually be the final product.

The inventory just connects to the CircleMenu in the end. It controls what is in each inventory slot.

PostProcessingVolume is the important values that control the post processing that each player sees. In this game, each player handles their own post processing, that way they are able to control what they want on and off once a settings menu is created. I will elaborate on how this work below in the Post Processing Section.

## Day / Night Cycle

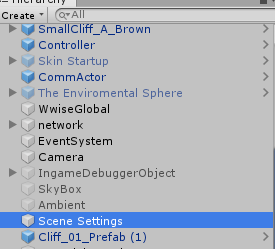
Currently in the game, there are 2 day / night cycles. There is one that I built, and one that Adon built. They differ by quite a bit.

The original day / night cycle consisted of 2 spheres with directional lights attached to them. They were on a 24 hour clock cycle, and had a proper yearly cycle. The yearly cycle controlled the sky colours too. In this version, the sky is running as a gradient sky. We controlled the colour of the sky, based on the time of day in a script called “Time Clock”. It was a pretty hard coded way of doing things, but it worked well. However, near the end of development, we scrapped that day/night cycle for a newer one that has more potential.

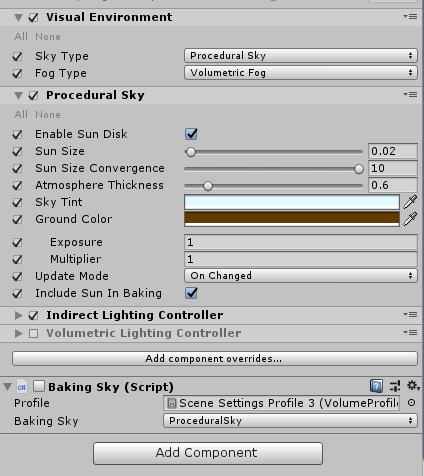
The newer day / night cycle just has 2 directional lights that rotate oppositely depending on the time of day. The sky is a procedural sky, which meant that a sun disk is automatically produced in the sky box, rather than needing a 3D object. This day / night cycle is controlled by the “Day Night Cycle” script.

## Scene Settings

To control the different sky boxes, and other environment features, find the “Scene Settings” object in the Unity Editor (see image below).



This Scene Settings object controls a lot of the different environment stuff, including the different types of sky box types, fog, volumetric lighting etc. To create something new in here, click the “Add component overrides…” button, and select what you want.

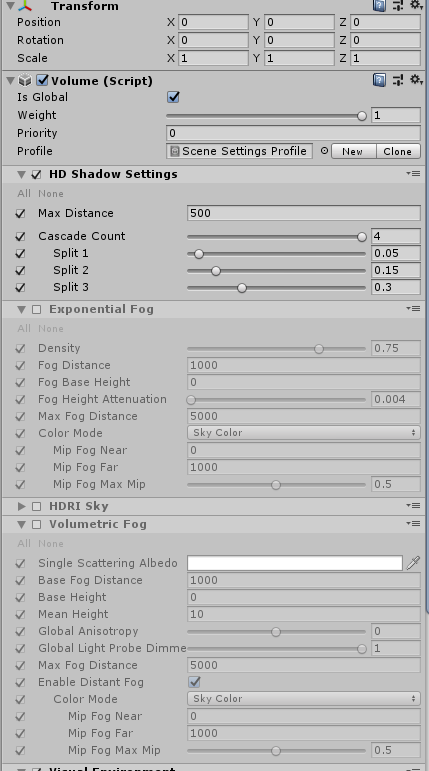


Visual Environment controls the type of sky we wish to use, and the type of fog that we want in our scene. If you set these variables to something you have not set up, it will just take the default values instead. This just controls what Unity will render in the scene.

Procedural Sky is the sky that we are currently using. This controls the gradient colours of the sky, as well as a few other things including the sun disk that shows in the sky. It is pretty self-explanatory, just play around to figure it out properly.

Baking Sky is a script that controls what sky is actually baked into the scene. You can setup and turn on other types of sky types, and then set a different one in this script, but the results can be messy. Normally just set this value to whatever sky you have active above.

Volumetric fog controls the fog in the scene, however this type of fog differs from the other type of fog as it reacts to the lighting differently. This fog will give off volumetric lighting (also known as God Rays). This is noticeable when light shines through the trees. This effect is very pleasing, so keep it.



As you will notice, there are a lot of unused / unchecked values in here. This stuff is stuff we have used in the past, for different builds of the game. We are still trying to figure out what feel we what in this game, so we have kept these values so we have references to past work. Turn them on and off to find out how they work. They all work very similarly to how the previous things work.

## Book of the Dead

Book of the Dead is a demo showcase which Unity created to show off the power of Unity. It is a visually beautiful scene that runs amazingly well. Adon was inspired by this. If you want to test this scene out, either open up the unity project labelled something like “New Book of Dead”, or re-import the Book of the Dead package into a new scene and play around with it from there. The “New Book of Dead” scene has been broken down, so it is not the full scene, and will probably look quite broken. This is due to me breaking down the entire system they use to try and replicate it.

In a nutshell, they use a combination of high resolution textures, well-made post processing, custom built wind controllers / shaders and custom built lighting and occlusion probes. I had replicated their post processing in our game, however that has changed now with the attempt to create the dynamic post processing between the different regions on the map. The thing that makes Book of the Dead better visually than our scene, is the use of their Occlusion probes. This is something that Unity custom built for that demo, and is not technically available to the public. You could try and write your own, but it was far too advanced for us to try and do in the short time period we had.

In our game, we have also “borrowed” some of the assets they have used in their demo. Since we have set our game up to be an HDRP scene, it actually ends up being rather simple to bring these assets in now.

## Animations

I have created a basic animation setup for the player. This is completely a placeholder as it was a last-minute addition. However, it is set up in a way that it shouldn’t be too difficult to figure out from here.

I have 2 animations set up in the player prefab. The first is called “isleHandAnimation” and it is turned off and at the bottom of the project. This was originally set up to be the idle animation for the player. I am not sure if it still works, but I have kept it in case it does work.

The second animation is currently in use. It is called “possessionHandAnimation”, and it plays when a player attempts to possess an animal.

Currently, for an animation to play properly, it needs to be in the first slot in the Main Camera. This is due to how previous students have coded up the hierarchy of the code. This can be fixed easily by having a more dynamic code structure, but we didn’t know about this properly until the last week of development.

## Particle Systems

I have a couple of particle systems built into this game. These are used for the clouds, and for the possession of the deer.

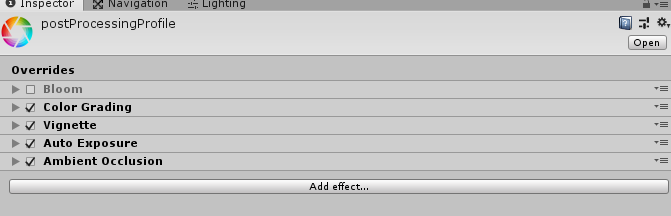
To do the cloud particles, I am using the prebuilt Unity prefab for the dust particles in their particle showcase. I have customised this prefab a lot, and it is named “clouds” in the project. It needs a bit of tweaking from where I have left it, but the clouds work pretty realistically.

The other particle system in the game is the particle system designed for the possession of animals. I made this in the last week of the semester, due to a last minute request, so it is currently named “practiceEffect”, and is on the deer, wolf and player layers. The player layer does not matter for now, unless you wish to customise it later. The layers on the deer and wold need to be tweaked to include animation of the deer, and to include the shape of other animals, including the wolf. This shouldn’t be too hard if you follow the skeleton I have made.

## Post Processing

Post processing in our game is rather simple and has changed a lot over development. The current post processing needs some hard work done to it as it was a placeholder and needs to react with the player moving into different zones on the map. There is a post processing volume attached to the player prefab, and this controls what post processing this player sees. Each player renders their own post processing; thus it can be dynamic based on zones. If you need to know how to lay it out, read the post processing documentation, its rather simple to understand.

We have a post processing profile (image below). This profile controls the post processing effects and is linked to the player prefab’s post processing volume. This is where you add the magical effects of post processing. Each of the different options is self-explanatory, so just play around with them to get the results you want.



## Potential Pink Object Fix

Due to working in HDRP, sometimes objects will import and become pink, or you might just randomly find that assets in the same may already be pink. This is due to the objects not having HDRP compatible shader / materials attached to them.

To fix this problem, go to **Edit** -> **Render Pipeline** -> **Upgrade Project Materials to High Definition Materials**

If this does not fix the problem, try reimporting the asset that has gone pink.