### Component 1

The first component in this package is the first person player controller from CW1, including player movement, jumping, and camera control. The log for this element will as such be the same.

The first thing to tackle is movement. To do this I create a function that detects directional input, and clamps the input so it can then be scaled by a speed which can be inputted/edited in the Unity editor. This is all contained in an if statement that checks if the bool “canMove” is true (which is set to be the case on Start) so that we can easily add elements that restrict movement in the future.

The next part of this script will create the camera control. For this, I have made the camera a child of the player so they move together. Then for the look direction the script reads the cursor location and moves the camera in the corresponding location. This function is tied to a sensitivity value which like the movement speed can be edited in Unity. The Y value of the camera's movement is clamped into a certain range to avoid unruly camera behaviours such as spinning or turning upside down. This is also all contained in an if statement checking that the bool “canLook” is true in the same way and for the same reason as the “canMove” if statement. Outside of this statement, on Start the cursor mode is set to Locked so the player can’t move the cursor outside of the game window, creating undesired camera behaviour.

The final part of the script enables the character to jump. The first part of this section checks if the player is on the ground. To do this a raycast is used to check if there is an object with the “groudLayer” tag close under the player. The exact distance that it checks is governed by “groundCheckLength”, another value which can be edited in Unity. Then, if the script returns that the player is grounded and the space key is pressed, the script adds velocity to the Y axis of the player using the “jumpPower” value, which again can be changed in the Unity editor. If the player is registered as not grounded then pressing space will do nothing, which avoids unintended double jumps or similar issues. Once again this is all contained in an if statement that checks for the value of the “canJump” bool to be true.

This script is added to a capsule to act as the player, with the camera set inside the capsule for a first person view. The capsule needs a rigidbody component attached to it to function correctly. Finally, a plane is added to the scene with the layer set to “groundLayer” so it can be checked by the jumping portion of the script.

Log:

Researching this component took about 1 day

Creating and debugging this component took about 2 hours

The next component is the “True Gate” - a doorway which when passed through will instantiate a prefab at a set location.

This is a simple script, using the “Instantiate” function along with a GameObject that is assigned the desired prefab. This function is checked using “OnTriggerEnter” so that a box collider that is set to be a trigger in the doorway will be the trigger for this function. However, to make sure only one instance of the object is spawned, a bool called “alreadySpawned” is created and set by default to false. This bool is set to true immediately after the prefab is instantiated, and both of these functions are contained in an if statement that checks if “alreadySpawned” is true. Therefore, next time the player enters the collider it will not duplicate the prefab as “alreadySpawned” has been set to true.

Log:

Researching this component took about half a day

Creating and debugging this component took about 30mins

The third component is a “False Gate” - a doorway which instead of instantiating a prefab plays an audio clip to let the player know they chose the wrong gate.

Since this function will be triggered in the same way as the “True Gate” it also uses “OnTriggerEnter” as the check for running this. This function also have a measure to stop duplication, this time using the bool “alreadyPlayed” which is set to false by default. Entering the trigger for the first time runs through an if statement that checks that “alreadyPlayed” is false, and if so it uses the “audio.PlayOneShot” function to play an audio clip defined by in the Unity Editor using the “public AudioClip” value which is listed as “SoundToPlay”. The function also makes use of a public float for volume so this can be changed in the editor too. After the audio play function, the “alreadyPlayed” bool is set to true so the function cannot be triggered again.

Log:

Researching this component took about half a day

Creating and debugging this component took about an hour

The final component of this package is the prefab which is instantiated by the “True Gate”. This prefab is a sphere that when the player is close enough to it and presses an interact button, will toggle a looping sound on or off.

This script uses “Update”, “OnTriggerEnter”, and “OnTriggerExit” to handle the audio toggle. First, in “OnTriggerEnter” a simple function turns the bool “canPlay” to true.

Similarly, in “OnTriggerExit” a similar function exists to turn the bool “canPlay” to false. This means that “canPlay” is only ever true while the player is in the trigger.

Then, in “Update” there is an if statement checking the value of “canPlay”. If it is true, it sets another bool, “canChange” to true. Then, if the player presses the E key down while “canChange” is true, it sets the audio attached to the prefab to switch its muted status, and sets “canChange” to false. This is to avoid the audio mute to be switched on every frame that the key is pressed down. Another if statement exists to check if the E key is up again (no longer being pressed) and if this is the case it sets “canChange” back to true, so the function can be used again as normal.

Log:

Researching this component took about a day

Creating and debugging this component took about 2 hours