**CW2 Reflective Journal**

Package 1: Movement and Card Map

First, I photoshopped the cards and imported them into Unity. I made some materials out of the card textures using an alpha cut-out. I applied them to two planes facing opposite directions to form a card. I then duplicated them and laid them out across the scene to form the basis for my map. I made a script to attach to each individual card to identify and distinguish between them later.

I used a capsule as a placeholder for a player’s body. I created a script for the player movement, within which I made an Enum listing move directions and a dictionary to hold detected cards using the move directions as a key. To populate the dictionary, I made a function to detect the cards around the player using an overlap sphere, setting their move directions as the key and the cards as the value.

To move the player, in the update function I detected if the directional keys are pressed down. If so, a vector3 variable is set with the next card position, then an if statement checks if the current position of the player is identical to the next card position variable. If not, the player is moved to the next card position.

To add some polish, I animated the cards to flip if they are detected by the player, only allowing the player to move to cards once they have flipped over. Furthermore, I added a spawn animation for the player to drop onto the map at the start of the game. Additionally, I implemented the ability for the card to spawn a prefab set in its properties so that I can spawn inventory items from a separate package.

Package 2: Dialogue System

I added a UI panel and configured a canvas. I then used TextMeshPro for the text. I formatted the text and resized it for the scene. I then created a C# script called dialogue. In the script, I made a using statement for TMP. I made a reference to the UI and made a string for the lines. I then made a float for the text speed and an index. I made a new method and an IEnumerator to type the line.

In the IEnumerator, I made a foreach that will break the string down into a character. Within that, I made a text component and told it to “wait for seconds” (using the text speed). In the method, I set the index to zero and made a coroutine with the typeline method passed in. In the start method I set a text component to an empty string and called the start dialogue method.

In the Dialogue Box’s inspector in Unity, I dragged the TMP into the text component under the script and set a text speed and size.

Back in the script, I created a new method for moving onto the next lines called NextLine. Within which I created an if statement that moves onto the coroutine should the index length is less than -1. Else if we don’t have anything else to say in the dialogue, the game object is set to inactive.

In the update method, I made an if statement so that if the left mouse button is clicked and the text component is equal to the lines at the index, then we want to proceed to the next line. Else if it is not equal, all coroutines stop and the text component is set to be equal to the lines index, getting the current line and instantly filling it out.

Package 3-4: Inventory and Pick-Up System

I created two scriptable objects:

- One to make new item objects

- And another to make new inventories to hold items

I then created an abstract base script which I used to define the basics for an inventory interface. I created a third scriptable object to make a new item database to hold the item objects and later reference for extra item information. I created an item data script to hold basic information of items required by inventory interfaces.

Next, I created an inventory slot script to hold item and interface data. Then I created a derived class of the base inventory interface script to deal with dynamic type interfaces of a variable number of slots. Conversely, I also created a derived class of the base inventory interface script to deal with static type interfaces with pre-set slots.

I then created a world item script to apply to scene pick-up objects to pick up items, along with prefab variants of each item. To make the item images I used two free AI software, they were:

- iamfy

- and artimator

Finally, I created a filter button script to apply to the UI buttons to apply item filters to the inventory interface. However, because I was trying to convert an inventory that took item amounts into account to a single item per slot inventory, I encountered a bug where it would duplicate items.

This happened when dragging items from the “stored” inventory slots into the playable slots for one-on-one encounters, and back into the stored inventory. It appeared to sometimes copy the representing image of the items rather than cut and pasting them, resulting in there being two of the same item, or more if I kept dragging them back and forth.

I fixed this by referencing the items directly, rather than using the index of items in a list.