TASK DOCUMENTATION

INTRODUCTION
The assigned task involved creating a clothing shop within a virtual environment and developing a character capable of interacting with the world and performing movement actions. This document will provide an overview of the approach taken to accomplish this task.

IMPLEMENTATION

Movement:
The character movement was implemented using Unity's new Input System, which provides a streamlined and flexible way to handle player input across various devices and platforms. By connecting character movement to the OnMove function provided by Unity's Input System, we can easily take information about Vector2 input changes and use them to make character movments.

Character movement is managed within the *CharacterController* script, where its primary responsibility lies. Additionally, this script handles animation triggering. However, due to the animator component residing on a child GameObject for accommodating various character skins, accessing the animator requires first accessing the active child GameObject.

Skins:
The player has the ability to change their character's appearance, clothing. Each skin is represented as a separate prefab within the player object in Unity.

Player skins

These prefabs depict different clothing combinations or distinctive styles for the character. Implementation for changing these skins is done via *OutfitManager* script.

Each skin is purchasable based on the amount of in-game currency the player possesses. The purchasing mechanism is implemented within the *Store* script, which manages the acquisition of items as well as the corresponding UI interactions. This ensures that players can browse available skins, make purchases according to their budget, and receive real-time updates on their currency balance.

Store UI

As depicted in the image above, the top section of this panel showcases the store interface, presenting various items available for purchase. Below, the player's inventory is displayed, housing their collection of skins. Through this inventory panel, players can effortlessly alter their character's outfit by selecting the "Equip" button. Upon doing so, immediate visual feedback is provided in the left corner, reflecting the changes made to the player's skin. In this panel we can also see amount of money that player have.

Store script is responsible for everything representing each element on this panel, except coin element which is connected to the player *MoneyManager* script.

The *MoneyManage*r script serves as the guardian of the player's financial resources within the game. Its duties include managing the addition and expenditure of money, while also providing alerts when funds are insufficient for a particular transaction. Additionally, the script takes charge of refreshing UI elements that display the current balance of money available to the player.

The Store and Player inventory panels are composed of inventory slots, which have been instantiated as prefabs. Each slot comprises UI elements such as price, name, image, and buttons. Associated with the game object are two scripts: the InventorySlots script, responsible for managing slot information by updating and clearing slot data, and another script that handles button interactions.

The InventorySlots script serves as the backbone, managing slot information and ensuring it's updated and cleared appropriately. Meanwhile, the second script is tasked with handling button events. It dynamically adjusts its behavior based on the specific button clicked – whether it's for equipping an item or making a purchase.

All the scripts that are binded to the itemSlots are stored in Scripts->UI.

Upon clicking the Equip button, the associated script utilizes its reference to the player to trigger the *Outfit Manager.* This action prompts the player's skin to be changed accordingly. Additionally, the script invokes a function within the UIManager, triggering an event that notifies all items to update their visuals to reflect the player's new icon.

if the Purchase button is clicked and the player possesses sufficient funds, the button delegates its responsibilities to the *Store* script. This script takes charge of managing the purchase process, ensuring that the transaction is completed successfully and that any associated actions, such as updating the player's inventory or deducting funds, are executed accurately.

Upon successfully purchasing an item, it instantaneously appears in the panel below within the player's inventory.

Store:
The Store script manages two key lists: the list of items and the list of inventory slots. Each item in the item list corresponds to a ScriptableObject that encapsulates essential information such as price, name, image, and ID.
During initialization, developers populate the item list with ScriptableObjects via the Unity Inspector. Subsequently, the Store script dynamically initializes inventory slots based on the count of items in the item list.

Store script is positioned on the StoreManager game object.

Inventory
InventoryWarderobe in the code, is a script that is positioned on the player and holds info about player skins (purchased skins), as we purchase skin, sell function within store class call function on the player class. This function assumes responsibility for adding the newly purchased item to the player's inventory and deducting the appropriate amount of money from the player's funds.

Dialogue:
Dialogue implementation involves three main classes: Dialogue, DialogueTrigger, and BaseInteract, along with the TalkableObject script.

BaseInteract Abstract Class: This abstract class serves as a foundation for interactable objects. It contains an abstract interaction function that inheriting classes, such as TalkableObject.

The *TalkableObject* script, associated with objects capable of initiating dialogue, handles interaction events. When the interaction function is called, it triggers the *DialogueTrigger* script, which activates the dialogue system. This script holds information about the character we are talking to, stored in a ScriptableObject called NPC.

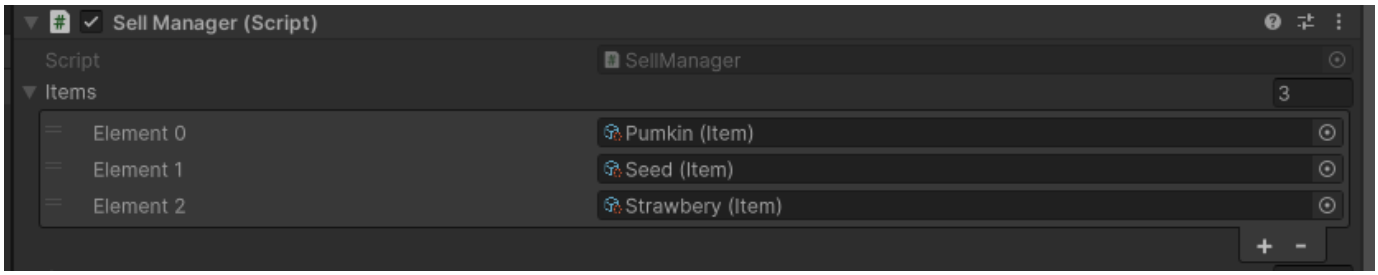
Initially, it may display instructions for interaction, followed by the opening of the dialogue system itself.

Dialogue Class: This class defines the structure of a dialogue.
DialogueManager script, responsible for managing the presentation of dialogue messages and additional buttons.

TalkableObejct script is responsible for tracking player disatnce, It manages a radius where it checks if the player is still within proximity. If the player moves out of this radius, the script disables the dialogue system. We can see example in the image below.

Talkable object radius

Sell Items:
Additionally, there is functionality for selling items, during which the player earns money. The script responsible for this resides on the player itself and is named *SellManager*. This script accesses the inventory, which stores items for sale. These items are also created as ScriptableObjects, and before the start of the game, we can populate them into the inventory list. Subsequently, we can sell these items when entering the shop.



Populating inventory

Sell Panel

I would like to note that this object on image is store because it is not so clear on the scene. When we enter it sell panel pop-ups.

Store where we can sell items