Adonis Carvajal

The way the system currently works is by using multiple systems along with event invocations to alert the other interested systems of changes so they know when and how to react. To start, the player (*Hero.cs*) gets initialized and calls on the inventory manager (*InventoryManager.cs*) to create its inventory. The inventory manager then calls an event once it is done creating the player's inventory which the UI (*MoneyCount.cs*) is listening for in order to initialize itself. The inventory manager also invokes a separate event when the player's balance changes which the UI is also listening for in order to reflect the changes back to the player. Along with inventory creation, the player sets its separate parts as variables to allow for the sprite to be changed later on. The inventory (*PlayerInventory.cs*) is a Unity scriptable object which contains a list of Inventory Items (*InventoryItem.cs*) as well as a float for the player's balance. An Inventory Item is also a Unity scriptable object, these however hold data for: Item Name, Item Description, Item Image, Item Buy Price, Item Sell Price, Item Type, and a boolean which states if the item is currently equipped or not. I use the Inventory Item scriptable object heavily in order to obtain a reference to price and image in order to populate both the shop and the player's inventory (*LoadInventory.cs*) as well as to assist with swapping equipped clothing.

The shop works fairly simple, and follows a basic button based flow (*ShopFlow.cs*). When the player enters the shopkeeper's interact zone (*CollisionCheck.cs*), a symbol will appear above the player's head, signaling that they are able to enter the shop menu. Upon pressing the E key, the shopkeeper will display three options: "Buy", "Sell", and "Chat". Clicking on one of these buttons will continue the shop flow. Firstly, clicking on "Buy" will load the shop's inventory (*LoadInventory.cs*) using the Inventory Items located within the shop's PlayerInventory scriptable object. You can then click and drag these items to the zone labeled "Buy" which will add the item to the player's inventory, remove it from the shop's, and reduce the player's balance by its price (*BuySellDrop.cs*). Secondly, clicking "Chat" will have the shopkeeper greet the player, the player then left clicks to continue the dialog. Once the dialog has been exhausted the player is returned to the "Choose" screen to select an option once again. The dialog system is relatively basic, and uses an IEnumerator function to display every character from a string array over time as opposed to at once (*ShopTalk.cs*). Lastly, clicking "Sell" will load an inventory using the player's inventory (*LoadInventory.cs*). This inventory will only display non-equipped items and the player can then drag and drop them to the zone labeled "Sell" to sell the item to the shopkeeper. Doing so will remove the item from the player's inventory, add it to the shop's and increase the player's balance by its sell price (*BuySellDrop.cs*).

The way swapping works is relatively simple, as all that really occurs is that the current sprite being used for the respective part is swapped with the new item's image. I make use of Unity's Interfaces to create easy to use drag and drop functionality for buying, selling, and equipping clothing with some minor checks to make sure the player can't equip a body part to his head or buy items if their balance is below its price. Both the player and the shop inventory use the same exact script to build themselves on the UI along with personalized content size fitters and grid layouts to help make them look better. The player then, can open their inventory by

exiting the interact zone and pressing the E key. The player can then drag the item (*UIItem.cs*) they would like to equip and drop it on its respectively slot (*EquipmentSlot.cs*), this changes the sprite image being used for that equipment type on the UI Player (*UIPlayer.cs*) which then invokes an event that the player's playable character is listening for to alert him to change its sprite for that body part as well.

My thought process during this was result based. By this, I mean that I looked at what my end goal was and worked by way backwards from there. For example, I thought to myself that the best and easiest way to swap clothing is to have a reference to the sprite, and then just change the sprite for that body part. This would not affect any animations or any other systems in place as I am not changing anything about the object itself and so I felt that this was the most effective way to go about it. I then kept moving forward with this end goal in mind, constantly asking myself questions from relatively easy ones like, "How do I get a reference to the current sprite?" to more difficult ones like, "How can I obtain a reference to the InventoryItem script attached to the item I just dropped on my EquipmentSlot?". Answering some of the questions proved to be challenging, but thanks to both my researching abilities and the Visual Studio debugger, I could find answers on external sites, on the Unity forums, or by looking at what was happening in memory.

I think this prototype is a good reflection of my current abilities with both game development, programming and Unity. I could never argue it to be a perfect prototype, but it definitely shows that I am a game developer with a passion for my craft.

*** I only reused one script, this was the script for the camera called *CameraController.cs*, every other script was written from scratch by me during the making of the prototype with some of them having influence from previous projects I have worked on.