LSW Clothes Shop Prototype Documentation

Managers

All main systems in the prototype work through the use of Singleton managers. By being singletons there's no need to predefine their reference on external scripts.

Audio Manager

AudioManager.cs

The audio manager stores audio that will be used in game, spawns, keep track of, and despawns audio sources.
Uses the Pool Manager for instancing.

Dialogue Manager

DialogueManager.cs

The dialogue manager handles the dialogue UI, player interactions with the dialogue UI, and related features such as the Typewriter method used to make letters from dialogue appear one by one.

Input Manager

InputManager.cs

The input manager handles player inputs and calls on delegated events for easier access to input information. These events are used throughout a number of scripts including the player controller script, the character sprite controller script, and the ui manager.

Inventory Manager

InventoryManager.cs

The inventory manager handles the storage of owned clothing items, clothing items in the shopping cart, equipped clothing items, and money. All this data, when modified, such as when a new clothing item is purchased and added to owned items, sends out delegated events to make communication with UI and other external elements easier.

Pool Manager
PoolManager.cs

The pool manager handles the enabling, disabling and resetting of pool-able prefabs. This manager is used most by the UI manager to reuse repeatable UI elements and reduce unnecessary instantiating and destroying of gameObjects.

Scenes Manager
ScenesManager.cs

Only function at the moment is quitting the application.

UI Manager
UIManager.cs

The UI manager is by far the most used manager. It handles the references for most UI elements, interactions, pooling, UI animations, and tracking of related elements. It contains methods to Open, Close, Setup, and Update most UI windows. This manager also contains all Inventory Manager events to dynamically adjust displayed information.

Mechanics

With the managers handling most links and interactions between different systems. I'll now explain the basics of how the following mechanics work, as well as describe some of their key scripts.

Shopping

SO_ClothingItem.cs
InventoryManager.cs
UIManager.cs
ClothingPreviewBehaviour.cs
NPCBehaviour.cs

SO_ClothingItem is a scriptable object script that contains information about a particular clothing item. This information ranges from the name of the item, its cost, and its icon, to all necessary sprites to render it on the character skeleton. This script and its information is used by most scripts in the game.

The ClothingPreviewBehaviour script handles a specific UI prefab that is pooled and set up with the information from the SO through the UI manager.

The NPCBehaviour script is a generic behaviour script for NPCs, it is not used on its own but rather through a child script called NPC_Shopkeeper.cs, which inherits from the generic NPCBehaviour. The npc behaviour script is necessary in this mechanics since a purchase can only be completed by talking to the shopkeeper and paying for the items.

The mechanic then follows the path:
Player interacts with the clothes' rack. This calls on UIManager to display the clothes' rack items, setup through clothingPreviewBehaviour. On screen the player can (amongst other actions) click a button to add the selected item to their shopping cart. The player now exits the clothes' rack and goes to talk to the shopkeeper. Through dialogue options the player can choose to buy the items in their cart and if they have enough money, the transaction is completed. The items in the cart then are cleared out and instead added to the player's owned items.

Selling

InventoryManager.cs
UIManager.cs
NPCBehaviour.cs

Selling is a similar mechanic to purchasing, the only difference being that instead of entering dialogue to purchase items, the player can choose dialogue options to sell items they own to the shopkeeper. An important detail to note is that NPCs have a buyback rate, which means items will be less valuable when sold than when purchased. This buyback rate can be set up for each NPC individually.

The mechanic follows the path:
Assuming the player already has items owned. The player talks to the shopkeeper, through dialogue chooses to sell items, a UI window with owned items and disabled equipped items pops up with the sell value displayed on each of them, the player can then click on any unequipped item to sell it.

Customization

InventoryManager.cs
UIManager.cs
CharacterSpriteController.cs

The player character contains a skeleton hierarchy with each bone object having a spriteRenderer as a child. The character can be animated and each of the sprites can be changed without one thing affecting the other.

Customization, or more specifically, changing the characters' displayed clothes, can be summed up to these 3 scripts. The inventory manager stores the owned (and as such available) clothing items and the currently equipped items. The UI manager displays this information on a window when the player enters a customization area. And the CharacterSpriteController handles the storing of specific sprites for each direction and each body section. Going into more detail about the CharacterSpriteController script, it has a reference to the currently equipped sprites, falling back on a set of default sprites which are always available in case one clothing item may be missing any. The sprite references for each sprite include 3 different versions of the sprite for 3 different directions that the character can be facing at any given time. These being forward, to the sides, and backward. The script has access to all 3 sprite references so that it can dynamically switch them based on the movement of the character. This link is done through a delegated input event that passes direction information. A different method handles the update of the currently equipped sprites, replacing specific sets for new ones (ie the torso and arms' sprites get replaced when a different top is equipped). Both of these methods can then be used in combination to change the sprites of the character, and update them according to movement.

Dialogue

SO_Dialogue.cs
NPCBehaviour.cs
NPC_Shopkeeper.cs
DialogueManager.cs

SO_Dialogue is a scriptable object script that stores dialogue options for npc's, these options have a dialogueType attribute to differentiate them.

Dialogue is handled mainly through the SO script and the NPCBehaviour child script. The NPC_Shopkeeper script contains different responses to player chosen options. These options are also presented by the shopkeeper and displayed through the dialogue manager. (Options such as talk, sell, and buy). The npc behaviour script dictates what the npc will say and has linear reactions to the player's choices. (If the player intends to buy, the npc asks them to pay or if they have no items in the cart, to go get items first).

Disclaimers

- The Singleton.cs script was imported from a previously developed project.
- The base for the PoolManager.cs script was imported from a previously developed project and then modified.

Final Thoughts

I enjoyed working on this interview task. The requirements were simple enough that I could plan out the mechanics I wanted to be included broadly and then polish them as I made progress on each of them. Because of time constraints I wasn’t able to polish every one of them as much as I would have liked to however. This is especially true when it comes to the UI visuals as I had planned to include shaders to make the experience more appealing. On the other hand I believe I was able to complete most of the required features with decent scalability and flexibility. Overall I believe I did a good job within the amount of time I had for this project.