Luka’s Shop - Development Notes

1. Game Mechanics

The main mechanics of the game are:

- 4-directional controlled top-down view;
- World and NPC interaction;
- Shopping system;
- Item inventory;
- Item equipping.

1.1 4-directional controlled top-down view

The player movement is done through the PlayerController script. The method PlayerMovement, that is looped on FixedUpdate to synchronize the RigidBody2D and its collisions, grabs the directional inputs horizontal and vertical (Arrow keys and WASD) through the method Get4WayPlayerMovement, that captures the values and limits them so that only one direction can be set at once. This limited value is then used to move the player's RigidBody2D's position.

The character is able to run faster if the player holds down the Run Button (the X key on the keyboard). If the character is running, the attached Animator's speed is also affected accordingly. Besides changing the running animation speed, three other variables are sent to the character's Animator: the current horizontal and vertical input values and the square magnitude of those values, named as speed, to detect player movement and help in the animation transitions.

Inside the Animator, there's a script that holds the last inputted movement values and updates them into separate parameters, used to determine which position (up, down, left or right) the player stopped moving, to set the idle animation accordingly.

1.2 World and NPC interaction

The world and NPC interactions are managed by a conjunction of events and event managers, that are present both in the scene environment and the project environment in the shape of ScriptableObjects.

The interaction system begins in the Interaction abstract class. This class is used as the parent class for all types of interactions present in the game: TalkInteraction and ShopInteraction. Each inherited class overrides the OnInteraction method according to the function that they need to execute. Since we're working with ScriptableObjects, the reference from both the player character and the interacted items are sent through the method. Another reference that is also sent together is the GameManager reference. This reference is really important to make the bridge between the scene environment and the project environment.

ScriptableObjects can trigger events and for every event, there's an event listener inside the scene space that handles such events. There is a big limitation on the interaction between the scene environment and the project environment, where ScriptableObjects are instantiated. It's impossible to send object references between the two realms, making it very difficult to connect them together. For every ScriptableObject that triggers an event, you need a handler on the other side that listens to this event and continues the desired operation. However, sometimes it's difficult to continue certain operations.

1.3 Shopping system

The shopping system was programmed to be bi-directional. Players - NPCs or not - are able to sell and buy items to and from other players. This is done through the Player ScriptableObject, which holds the respective player inventory, and also through the ItemTransaction and TransactionController classes.

The ShopController is an interface to select which item from the other player's inventory that the transaction should be made on. Through it, an ItemTransaction order is made, in which variables like the players' origin and destination - seller and buyer -, the item and its index on the seller player's inventory are saved and processed by the TransactionController.

The transaction is then processed, where different rules apply for PC and NPC characters. PC characters sell at the resale price (half-price - not necessarily. A different value can be set on the item's ScriptableObject), while NPC characters sell at the item's price (full-price). This differentiation is made through class inheritance. The PC character uses a virtual method that can be overridden by the inherited class.

1.4 Item Inventory

The item inventory system is made on ScriptableObjects. Each player has its own inventory. Be it a PC or an NPC. It has two lists: Items and EquippedItems. Each list holds respectively, the items that the player owns and the items that the player has currently equipped. There isn't a limit on how many items the player can equip on code, however, the Player Inventory UI only allows for 5 items to be equipped, as there are only 5 slots. Like the ShopController, the PlayerInventoryController only handles item selection and display.

1.5 Item Equipping

The equipping and unequipping mechanics are handled by the ItemSlot and InventoryDropHandler, respectively, and set in the Inventory ScriptableObject. Each Item ScriptableObject has their own events related to equipping and unequipping. These events are triggered and answered by the event listeners inside the scene. As mentioned earlier, it's impossible to send references between the two realms, so all changes related to the scene need to be done with objects that belong to it. This means that there's two listeners inside the scene for every item. One that listens to the OnEquip event and another that listens to the OnUnequip event. The wanted action is then executed through the listeners.

Clothing equipping is a bit different however. It suffers from the same limitations discussed above with cross-realm references, however, for all clothing items, only one event of each is needed.

When the player equips an item, the PlayerController will respond to the event, searching all currently equipped items for valid clothing. This is done this way because of the object reference limitation.

When a valid clothing item is found by the PlayerController, it will merge the AnimatorOverrideController that is saved inside the Item ScriptableObject with the current AOC animations.

The player character is separated into three parts: body, hair and head. By separating into these parts, I'm able to mix them by merging the animations.

2. Conclusion

By doing this interview task, I've learned and practiced many things and implemented new scripts that I had never implemented before. It was quite challenging and fun. I finished the main requirements in about 85 hours out of the 96 hour-limit. Using the ScriptableObject / Event combo can become really convoluted to program and understand, however, once you have the programming part of it over, it becomes more modular as you can reuse objects over and over.

3. Third-party Assets

All scripts were written during the interview task, with the exception of:

- Game Architecture with Scriptable Objects by Ryan Hipple;
- CoroutineWithData (not used) by Ted-Bigham;
- ExtensionMethods by JonLord;

All graphical assets were pre-made assets:

- 2D Tiles and Sprites by Hyptosis;
- Sprite Pack - Fantasy Male Mage by WitPOP;
- Top down RPG characters by Arcade-Island;
- Generic 16 * 16 Tileset (not used) by Ojas Sabadra

All sound assets were pre-made assets:

- Typewriter SFX by Mixkit;
- The Hero BGM by Guilherme Ferrari, recorded by me;

Text assets:

- Monty Python & The Holy Grail - Movie Script