Instructions:
WASD/Arrow Keys - Move
Space/Enter - Interact
Escape/Tab - Open Inventory

The player can talk to everyone on the island and interact with the pumpkins. Going south, the player finds a shopkeeper and a shopping cart in which they can buy new equipment or sell their old equipment (by clicking on them). The player can then open their inventory and equip their new equipment.

For this test, I started by analyzing the required features and what external assets I was going to need to develop them

.

A top down game with dialogue, inventory, shopping/selling items and character customization requires a lot of art and sound assets, so I spent some time searching for the best ones that could keep the balance between visually appealing and visually cohesive.

After importing the assets to the project, I set up an Input Manager using the new input system, which would be used for the entire project. Setting up a basic 2D top down movement and a Cinemachine 2D Camera was pretty straightforward, so I started working on the dialogue system.

I used scriptable objects to define “Speakers” (characters with names, portraits and voices) and “Dialogues” (arrays of sentences said by speakers). The DialogueSystem class handles the interactions with NPCs and items and initiates the dialogue. The DialogueView handles the UI of the dialogue, such as animations, autotyping, and the textbox behavior in general.

To interact with NPCs in the game world, Three classes were created:
PlayerInteraction.cs, which listens to the interaction input and searches for interactive objects with a Physics2D.OverlapCircle();
IInteractive, which has an Interact() method and allows flexibility to be implemented by every interactive system (such as objects and shops);
and NpcController.cs, which implements IInteractive.cs and stores a queue of dialogues that can be triggered with an interaction.

Then, I started working on the inventory. The inventory can be populated with equipment (EquipmentDatas with names, descriptions, icons, types and prices) and each slot can store an item. The InventoryController.cs class handles the data involving the items the player has, while InventoryView.cs and InventorySlotView.cs focus on the UI of the inventory window and its behaviors, such as animations, hovers, etc. I was inspired by Stardew Valley’s inventory system and tried to replicate their way of interacting with the UI. The player can select items from any slot and position them wherever they want.

Afterwards, I developed an equipment slot (a class that derives from inventory slot) that stores an item and a reference to a Sprite Renderer in the player. If the item changes, the sprite also changes. That way, I was able to customize the player's appearance by equipping different items.

For the shopping system, I started by making an IInteractive shop with a list of items. ShopView.cs handles the shopping interface as a whole, and Purchasable.cs handles the individual items the player can buy in the shop. I used the same inventory in the shopping system (also inspired by Stardew Valley), but selecting an item in a shopping context will sell it.

In the end, I just had fun with the characters, the dialogue and the lighting. I believe the project has a clean code, looks appealing and feels fun.

EDIT: In this new submission, I ensured the items were well organized in the project, removed some unused variables, and improved the inventory architecture, which had some gameplay and UI logic mixed up.