Context

Blue Gravity Task was to implement a clothes shop assuming to be part of a simulation game similar to “The Sims‘ and “Stardew Valley”. And Also should be a top down game.
To do so I created a small game where you go to the store and depending on the player's actions he gets an ending.

Systems Breakdown

Game Scenes

The game has 3 scenes:

OpenScene: the game opens on this scene. Clicking play button goes to GameScene.

GameScene: Here the player controls the main character, being able to buy/sell clothes, interact with the scenery, earn and lose money, equip his clothes and trigger multiple endings;

EndGame: Here player learns his ending and can go back to OpenScene

Tiles

The tiles on GameScene were implemented using Unity’s Tilemap Editor.

Camera

The camera was implemented using Unity’s Cinemachine.

Player

Player is controlled by the class PlayerController. This class controls character movement using Rigidbody2D, his animations are controlled by a regular animator that interacts with the movement input and system. Since the game is short and will not grow I opted to add to PlayerController all player related responsibilities, usually it would be best to have each class a single responsibility, however to save time PlayerController acquired an inventory, wallet, character’s monolog system, the clothes displayed on character and player interaction triggers.

Inventory

The inventory controls 3 types of items:

Product Items: These are the clothes that a player can have on its inventory, they have statuses that tell if they are owned by the player, the store or are on the shopping cart.
Equiped Items: These are the items to be considered when asking for player status.
Named Items: These are items that are not clothes that the player can acquire.

Actions can be registered to inventory to be called when an equip is changed, this exists to allow me to update Inventory UI and Character Displayed Clothes.

Wallet

Since the game is based on the act of buying and selling clothes, I've implemented a wallet that can be accessed and used anywhere and can notify it's changes to anyone to subscribe to it. This is relevant mainly to update UI.

Character's Monolog System

The monolog system is the creation of a dialog box with lifespan that is triggered by interactables on map. It is attached to the character sprite.

Clothes Display System

Initially I was planning to create a system that used a modular character that would change its parts depending on the clothes used, however I could not find adequate assets, so I settled to 3 sprites attached to the character that would be changed based on the clothes and animated them.

Player Trigger System

One of my goals was to allow my player to interact with scenery, but only if the player is facing the object. To do that the character has 4 triggers that store interactable objects facing each of the four cardinal directions.

UI

All menus are game objects on the GameScene, but I made them to be self-contained. My goal was to extract each one to a scene and load them additively to better manage memory use, but alas I did not have the time to do so. Those menus are called by an UIManager. One detail was that the StoreStand Menu and the Inventory Menu are very similar and probably could both inherit from the same class, but due to the nature of the project being small I decided not to spend time doing it.

Interactables

There are two types of interactables, those that open a menu and those that call an action. The shopkeeper and the store stands open menus where you interact with the clothes in some way. All others call an InteractableAction. All interactables implement an interface called IInteractable, and have the tag "Interactable".

InteractableAction

To do the interactable action I used an architecture that I learned while working on CardGuardians, the idea is you create an abstract class that is inherited by other classes. These classes are capable of a behavior. To support these I've used a Service architecture to allow the actions to interact with any part of

the system they desire. The greatest advantage is that once you implement a few behaviors they work as lego pieces allowing you to create new interactions without having to script them. Actions created:

- CallEndGameAction: Call an end game, allows bg and text setup;
- Give a named Item to Player: add a named item to players inventory
- ConsumeNamedItemAndDoSomething: check if player has a named item and do something depending on the answer
- GiveMoneyAction: add money to wallet
- TakeMoneyAction: remove money from wallet
- YesOrNoInteraction: asks the player a yes or no question and does something depending on an answer.
- PlayerMonolog: Create a monolog
- PlayerMonologActionAndCallOtherAction: Create a monolog and do something

Interactable Actor

To use the InteractableActions I created 2 kinds of InteractableActors, SequentialInteractable, ConditionalSequentialInteractable, one aways progresses its interactions and the other only does so if a condition is met. With these I was able to implement all interaction on game