Project BGS Documentation

This is the documentation of the interview task for BGS. Below is the explanation of how the clothing store feature code works as well as the developers thought process behind it.

1) Thoughts about the task

First and foremost, the task clearly mentions that the feature should be created having in mind a Sims / Stardew Valley type of game similar to LSW, prompting to a modern setting and an ordinary clothing store. Given the fact that the time for the task was limited and that creating art is not my strongest suit, I decided not to spend time on drawing the assets used and used some premade assets for the project that I own from Unity Asset store. The assets mentioned are in the Fantasy theme which not perfectly suit the needs of the project but I feel they are fine in order for me to be able to create a functional prototype for the designers to test the feature and if that works to change the art assets later on in the process.

With having the functionality behind the system in mind, how I perceived the task was to create the mechanics behind a clothing shop system, with the player being able to buy/sell clothes, equip them in fixed equip slots he would have and also create a system to make those clothes changes visible to the player. That being said I focused mostly on delivering the functionality behind the system rather than the overall aesthetics of the prototype, since I know that the designers looking at it will probably have their “dev googles” on and that aesthetics is something that could be worked upon later if the system works as intended.

Due to the time limitations, I decided not to create a shopkeeper for the prototype and all the interactions to be made via UI buttons. Adding a shopkeeper and some text conversation between him and the player while a really easy task, it is something that could easily be added in the future if the system passes the inspection from the designers. Another thing I did not implement is prices of the items showing in the Shop Panel UI. This makes the player unable to know how much each item costs, even if the price / buy / sell system work fine in the background. It is something that the functionality for it is already implemented and all there is left to be done is to create a UI text in the item entry prefab and link it to the items price. I decided to skip that because I was running out of time and I wanted to wrap up the project soon.

Overall it was a very fun weekend! I was initially surprised by the fact that no art assets where provided and I spent a lot of time thinking how to proceed with the project and if I should try to find some to use or draw something myself. A lot time was spent on that part of the project. Having finished it, I know there are things that I could have done better but I prefer to actually do stuff than think over them forever. In a real work scenario there must be a balance between quality and time consumed for a feature and I feel that the final product is fine for the time spent on it. I would want to have made it a bit more expandable and maybe a bit more optimised if I had the time to reiterate on it, but I feel it is a good sample of my abilities and it covers the mechanics asked of me to create.

2) Code Documentation

Every script is commented and has a description on top of it describing its basic functionality. The idea behind the system was to make it easy for the designers to tweak things as they see fit. That being said I decided to make the pillar of the system being Scriptable Object items that could easily be created without the need to write extra code. Both the player and the shop would need to have an inventory to store their items in and the player should also have some equipment slots to place in the items he is wearing. When the player wears an item, the appropriate Sprite on the player GFX would swap to the sprite stored in the data of the item he equipped. As the game's view is top down 2D, this would mean 4 different variations of the player GFX (Front, Back, Left, Right views) and also 4 variations of the clothes worn.

ItemData: This is the base for all the items. Derives from Scriptable Object and is used to hold data for each item so that the designer can create them easily via the inspector.

Item: A class into which ItemData are loaded in. Each item is an instance of this class, getting its data from an ItemData SO.

Inventory: This script is responsible for creating inventory instances. Can be instantiated via a constructor and holds a list of Items.

InventoryManager: Handles the display of an inventory under a specified Transform, creating entries as children of that transform, by instantiating an entry prefab.

ItemSlot: Is a script attached to the entry prefabs the InventoryManager creates. It handles the functionality of buying / selling / equipping / unequipping the items. It holds the ItemData of the item it represents so that the programm knows which items to transfer when the button attached to the same GameObject is pressed. The script is a bit messy, with a lot of repeating code and some parts that could be written better. I am aware of this fact.

NOTE: The combination of InventoryManager and ItemSlot, instantiate / destroy GameObjects very often. In a real work scenario I would definatelly try to use some sort of Object Pooling to do that, but due to time limitations I decided to leave it as is as it creates no problems to a small project like that.

PlayerManager: Is responsible for holding player data, like his inventory and gold total, as well as references to the Sprites that need to change when the player equips an item and also some Item type variables where the equipped Items are stored.

ShopManager: The same functionality as the PlayerManager but for the store entity.

DatabaseManager: A static instance that holds data like the UI Transforms that are being accessed from many scripts. They are stored there for ease of access.

UIManager: The script that handles anything UI related.

PlayerControls: Handles player movement, sprite swapping for front, back and side views and handling the player animations.

CameraFollow: A simple camera script to make it follow the player’s movement.

Singleton: A utility script holding 3 classes (StaticInstance, Singleton, PersistentSingleton) that other classes can inherit from. I use it to most of my projects so that I can easily create those 3 type of classes saving me a lot of time in the long run.