System Functionality

The player walks around and interact with the “spacebar”.

The interaction opens the respective menu.

The “Inventory_Manager” handles the communication with the menu and the inventory.

Each menu has it specific functions. In the shop you can buy, in the player inventory inside the shop you can sell, and in the player inventory wardrobe you can equip items.

Thought Process

I started taking notes of the key points that the project needs to fulfill what was asked.

1 - Character Movement
2 - Animation System
3 - Scenery interaction
4 - Shop menus
5 - Inventory System
6 - Equipping System
7 - Visual Upgrades

As listed, the first thing I did was to create character movement functionality utilizing a rigidbody. I chose the rigidbody for the built-in collision functionalities.

After that, I started implementing the animation system. I created animations for each of the cosmetic items used in the game and an animator controller with a blend tree to handle the movement directions. I made the “Cosmetic_Item” scriptable object to store the animations and other relevant variables, so I can have easy access to their animator controller, prices, names and icons.

With the animations and character movement done, I created the “Interactive” class so the “GetComponent” processes could be done easier with multiples interactive objects utilizing the same “Action” function by inheriting it. I did this because I wanted to have the shop, the wardrobe and the money savings to be accessible using the same function and button.

The shop menus were the second most time-consuming part. I created the “UI_Controller” to handle with the calls to open and close menus and to act as a controller for the menu tabs and buttons. During the process of implementing the menus, I created the base for the inventory system to begin setting the references needed for the system's communication.

I made the “Inventory_Manager”, separating the logic functions and the tracking of the inventory tabs from the inventory itself and helping to let the “UI_Controller” only handling the UI display, trying to apply some concepts of the model view controller pattern. I also created the “Inventory_Menu” and the “Item_Slot” classes to help with modularity and to not let the “UI_Controller” messy. Furthermore, I chose tho create the “Inventory” utilizing scriptable objects because it helps with the modularity, and because wanted to have an inventory for the player and another one for the shop.

After the shop and inventory done, the equipping system was a piece of cake, just a matter of having a menu for equipping and a few variable more in the “Player_Inventory_Manager”, which inherits from the manager, to keep track of what items are equipped and equip the items.

With all the systems done, the last part was to make the game more nice looking using a tile set for the background.

Personal assessment

I think I had a slow start and get a little bit lost in the beginning, not knowing for sure what to do first, then I stated with the basics made in my head the key functions that need to be done, after that I think that my work pace was good.

I believe that my solution for the animations was not the better suited for this model of game, probably utilizing a shader to draw all the cosmetics items in a single image and get the UV position of the animation to do the match would do the process of adding new assets to the game much faster. But with the solution I made, I also made the equipping system easier.

I recognize that the visuals are far from perfect because the pixel art of some elements doesn't follow the same size, but I didn't want to spend a lot of time finding perfect matching asset packs for UI, background and character sprites due to the limited time.