Felipe Gonçalves Brigagão
Unity Developer

LSW Test

Thought Process

The first stage of the test presented was the preparation of an initial scope, where the tasks that would be necessary to reach the final project were divided over the days available for testing, with possible delays due to research and bugs that would come to appear.
The main idea for the development of the project was to make the code as generic as possible, while meeting the needs of the test.

First day: The tasks done on the first day were the construction of the scenario using tilemaps, and the basic construction of the player, such as movement, animation and input control.
The TileMaps were separated into 4 categories, floor, walls, shadows and boundaries, where each of these are in their own sorting layers, with the pivots and colliders located in positions where they will give a feeling of depth, with the application of the classification axis configuration of transparency for only the y.
The scripts were made to be able to work distinctly, where most of them only receive their input through public methods, being possible to apply them to any character, and only provide the input.

Second day: Inventory system development started, this was set up with the intention that it could be reused in any inventory that existed in the game, both the player and the store. With that said it took the basic mechanics of an inventory, such as storing items, selecting and using them in relation to their type, and then creating base classes, which could be inherited by future inventory classes. Scriptable objects were created for the items, these were stored in lists within each base class of the inventory, these lists are updated by methods within the base class, which insert and remove items from it when the logic of the current inventory requires such action.
The Inventory UI consists of slots, which are stored inside a slot holder, which has a unique script to update the icons of each slot after the inventory is opened, an item addition or removal, in the case of inventory undergo a change while the UI is disabled, this update method is called in OnEnable. Each slot has its own script, which makes it possible to change the icon and the item present in it, in addition each slot has a use method, which is linked to the item saved in it, where when the UI slot button is activated the use method, or another method related to the type of inventory will be triggered. For UI update all slots are saved in an array and then the list of items present in the base inventory is scrolled, where each item belonging to the base inventory is inserted into a different slot in the UI, and then with that each slot when pressed, the action of the specific item is triggered.

Third day: With the entire inventory base ready, the inventory construction for the store began, with some logic for opening UI's, in a way that they are not visually unpleasant. It started by inheriting from the inventory base classes the methods needed to build a common inventory, but as class inheritance allows the use of methods and events from the parent classes separately, it was possible to build the store menu, which consists of two panels, one for the purchase of products found in the store, and the other for the sale of products from the player's inventory, for the sale and purchase the slot script was updated on top of the base script, which now also has the methods sale and buy, where the sale evaluates the amount of empty slots in the store, and then the item is passed to the store, and the item value is added to a manager class that is responsible for the currency transaction, the item is taken straight from the inventory manager of the player, the sale of products is done in a similar way but with opposite sides, and the amount of coins the player has is evaluated for the purchase based on the value of the item. Each inventory has its own slot holder, where each slot uses a buy-to-store and sell-to-player action. A logic was implemented so that when an inventory is open the player cannot move, and no other inventory can be opened until the present inventory is closed. All purchased items are added to the player's main inventory.

Fourth day: A system of interactive objects was created, with which the player pressing the interaction key generates some action, opening the inventory for the store uses this method. For the interaction to occur an interactable abstract class was created, this class will be inherited by classes that will be placed in interactive objects, this class observes the player's entry in its interaction range, which is determined by the collider in trigger mode of the object, and a virtual method to implement the interact method will define in the inheriting classes what the interaction will be. When the player enters this range and the interaction icon appears, the interaction key can be pressed which will take an interactable instance through a box casting from the player, but the polymorph will trigger the method of the child interactive class, such as ShopCatInteractable, which will evaluate if any inventory is open, if not the shop will open.

Fifth day: With all the necessary mechanics for inventories elaborated, a panel of equipped clothes in the character was created, for that a new specific class was created for the UI and inventory of equipped parts, where each piece has its specific slot.
A logic was developed to equip wearable items from the inventory, where a scriptable object heir to the scriptable object item has an identification provided by an enum, where in addition to the separate identification in weapons and clothing, the scriptable object clothing is separated into head and chest, with this separation, it is possible to observe in which inventory slot a certain item will be equipped, so when another item of the same type is equipped, and one is already in the position, the already equipped one returns to the inventory and the activated one is equipped in its place.
The system for the UI of the clothes that will be equipped consists of placing 3 game objects inside the player's UI, one for the character going down, one for the character going up and

one for the sides, each game object has two sprite renderers, one for the head and the other for the body, where when an item in the player's menu is triggered, and that item is a clothing the sprites referring to that clothing are saved in game objects, these game objects are triggered by the character's animation.

Inputs:

Movement: A S W D or directional arrows.
Open inventory: Button in upper right corner or i key.
Interaction button: And when the icon appears over the character's head.

Final thoughts

I believe that I completed all the necessary steps to consider the test complete, some points such as modularization of some scripts and a better way of organizing the animations are things to improve, but several things were learned during this test, and I am very grateful for the opportunity given to me by the LSW team, I will try harder and harder to improve myself.