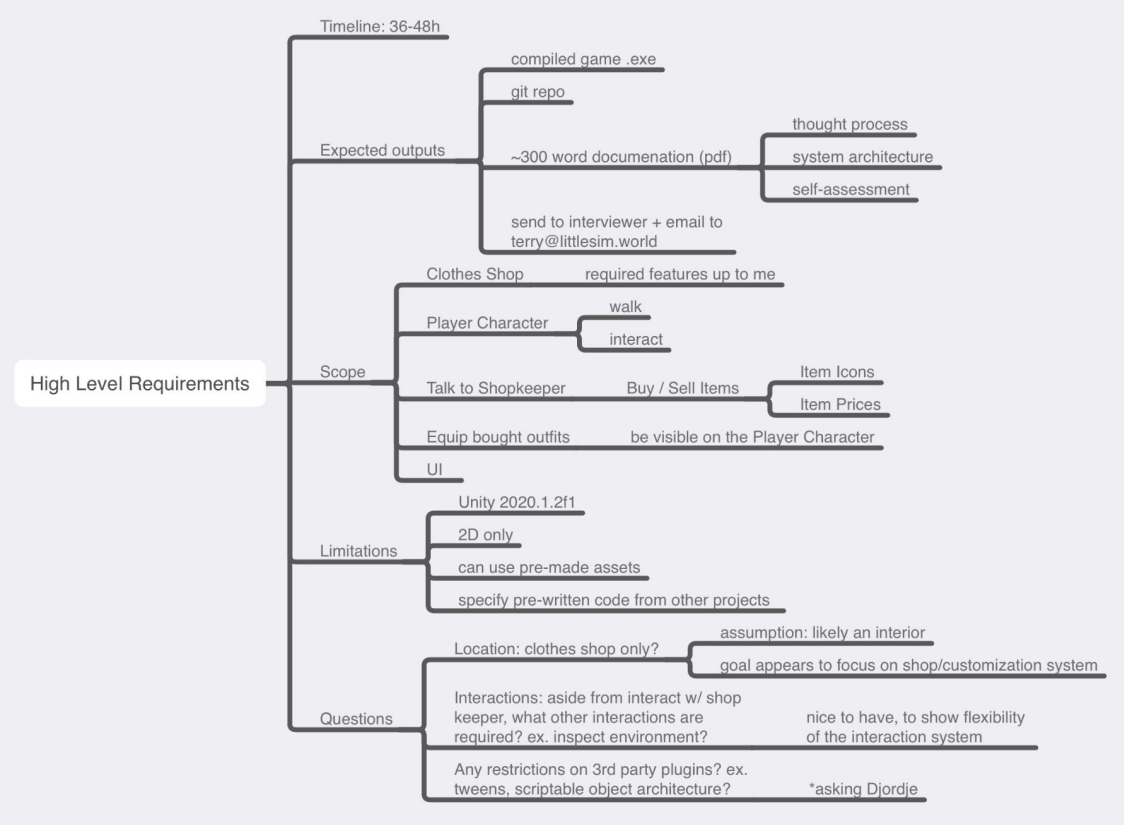
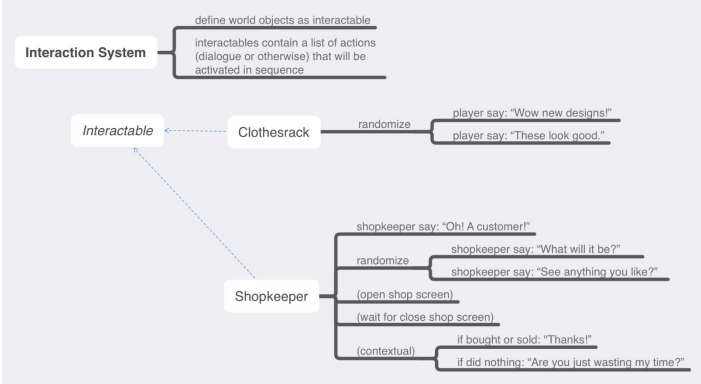
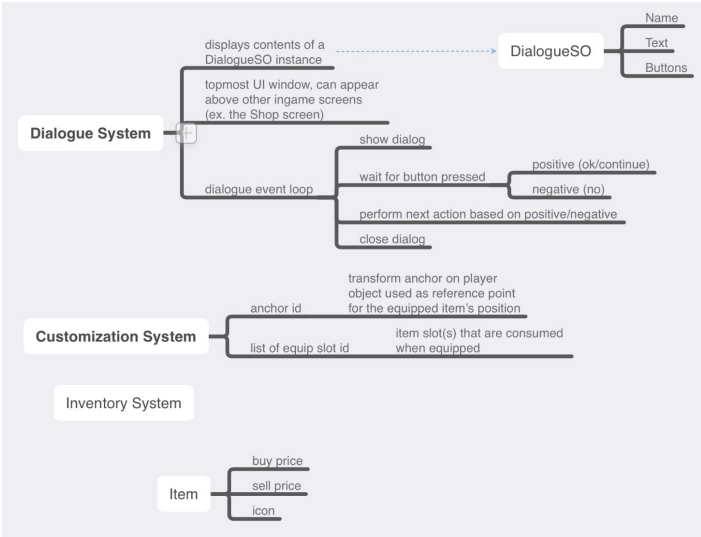
LSW Programming Test - Documentation

Thought Process: Initial Planning

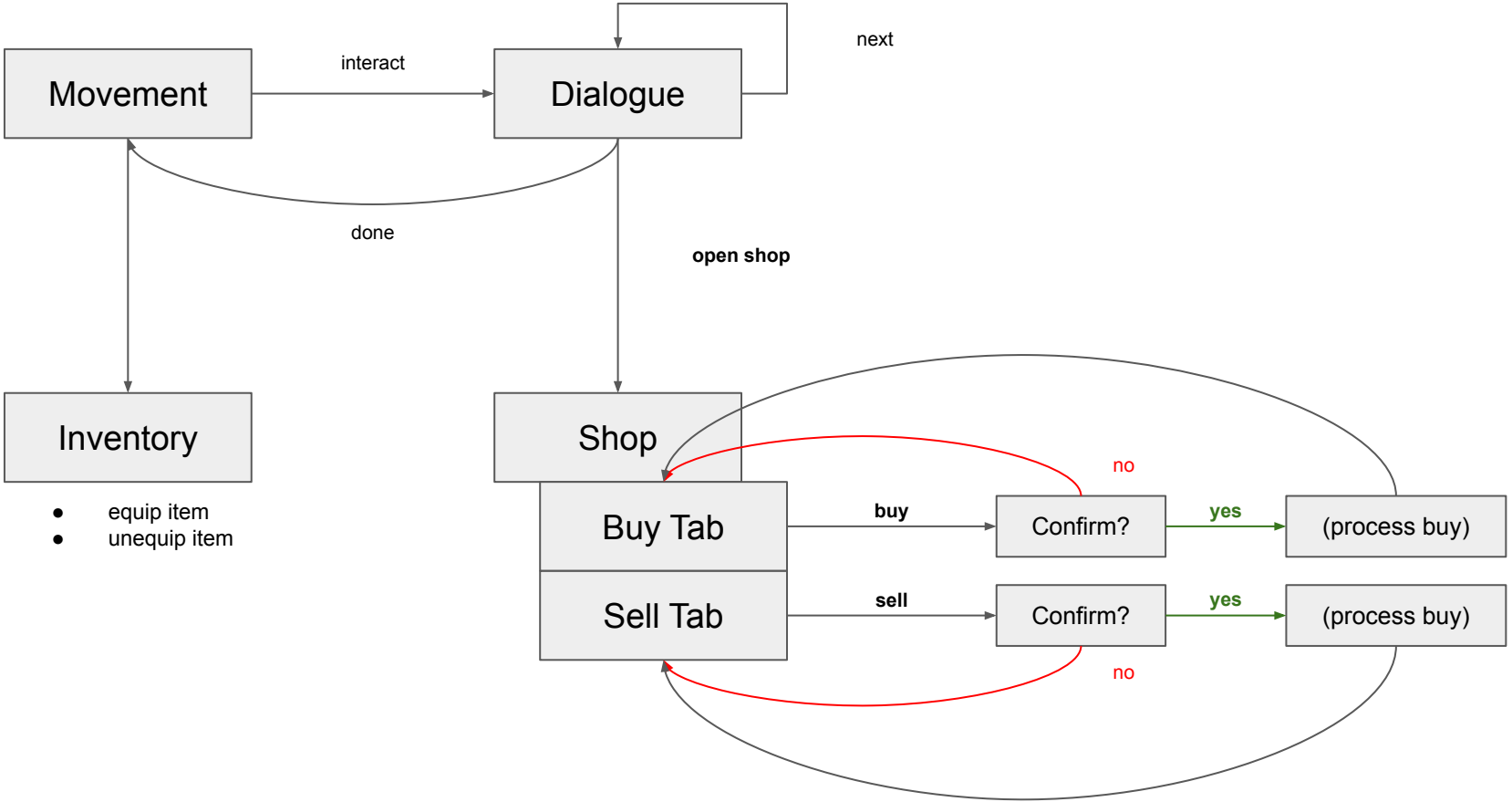
- The test scope was fairly broad, and covered several systems. I originally planned to have some additional features to demonstrate the flexibility of the systems. The primary concern was which systems to prioritize.
- I started by identifying the requirements and identifying the systems and their data/features. I also did some quick research for RPG shop and inventory screens to see the scope of the interactions.
- I also made a state diagram to get a better idea of some of what game events to expect.
- While making the state diagram, I noticed I might need a dialogue system for various things like talking to the shopkeeper and confirming transactions. Since I haven't made a dialogue system before, I made some wireframes to see the common data across different dialogue types.



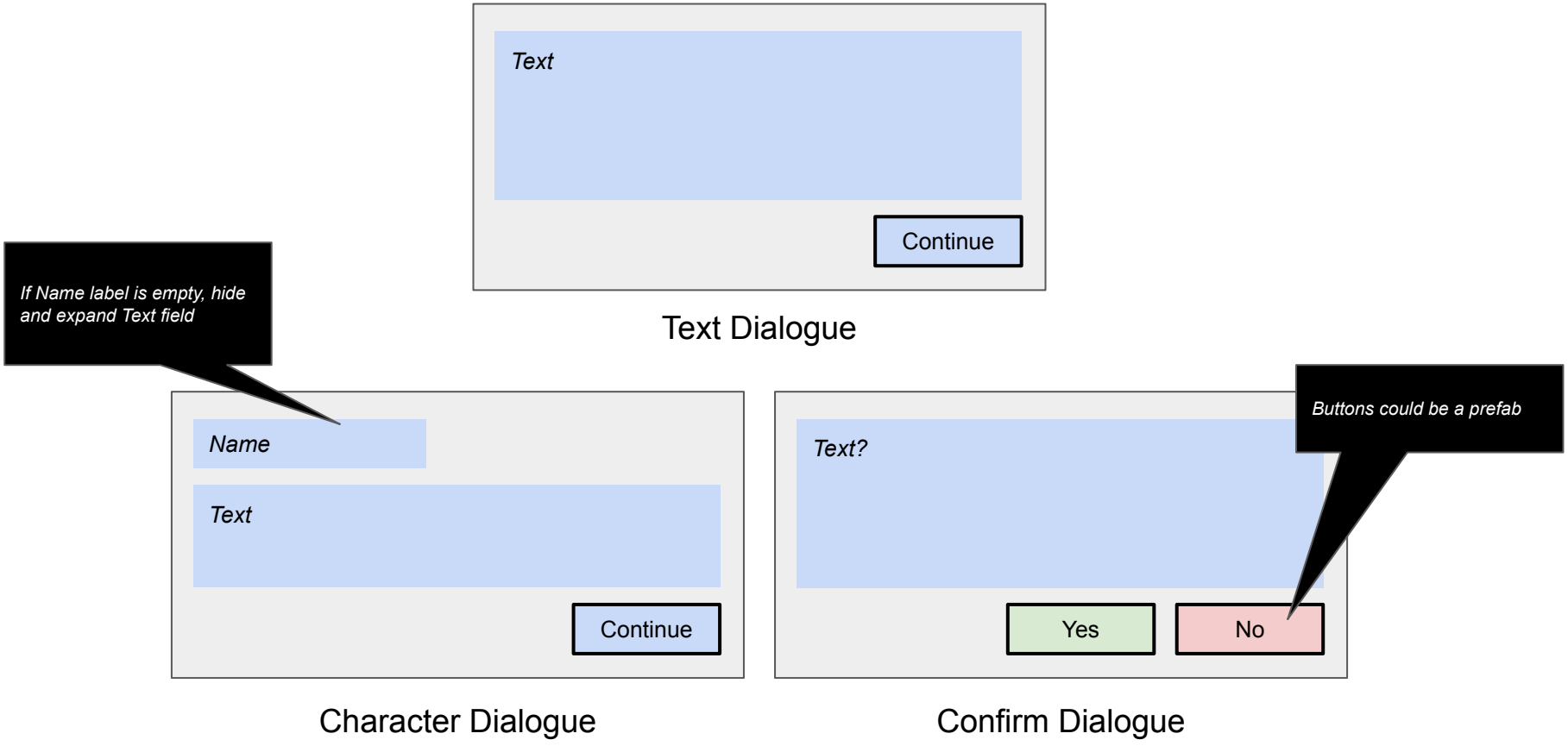




Mindmapping requirements and systems



High level state diagram

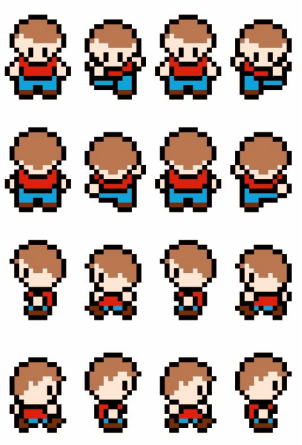


Wireframes of dialogue types

LSW Programming Test - Documentation

Thought Process: Art Assets, Code

- I didn't have a lot of time to create new art assets, so I decided to just look for some character spritesheets and work with it. Unfortunately I couldn't find a suitable one (top-down view with swappable clothes). So I took a low-detail pixel art spritesheet and edited the image to get spritesheets for the clothes.
- After this, I got busy over the next 2 days with some errands and lost about 8-12 hours of time in total. So I had rush into coding and get important features done first.
- My general process was like this:
    - Coding of systems starting with input, then prototype for the clothes system, then a bunch of the UI and inventory code. Almost no art assets at this stage.
    - Lots of additional coding work for the UI and inventory code to fix strange bugs etc.
    - Improved the visuals + fixed UI scaling problems at the last couple of hours.
    - Pushed the build and wrote this documentation



Original

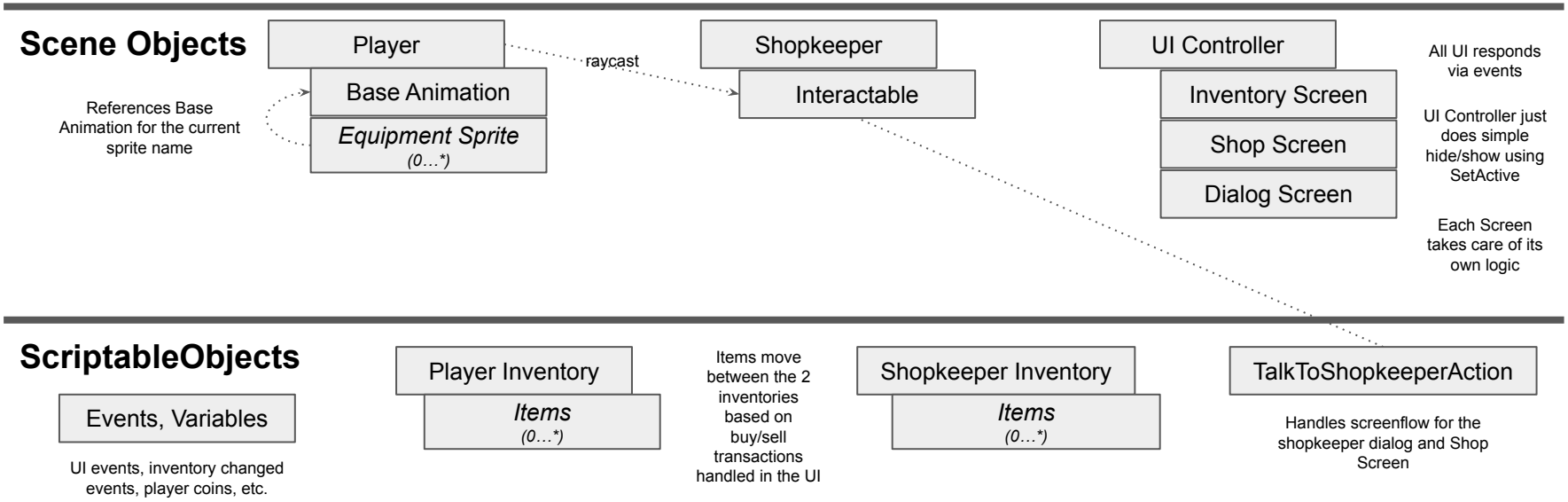
Made color variations for equipped items

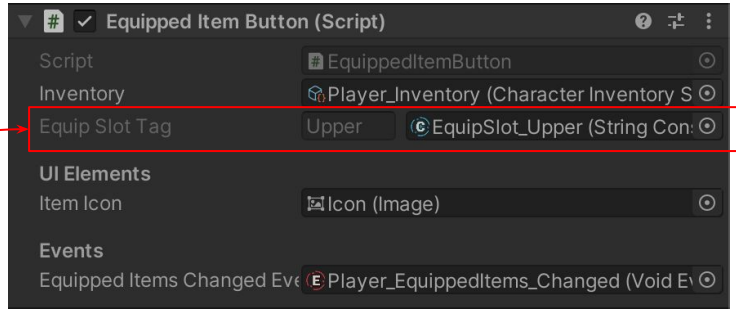
Made simple item icons (this was done towards the end)

Making Art Assets

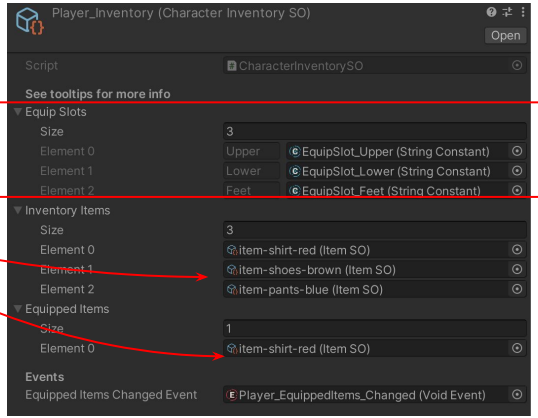
System Architecture

- The game uses a ScriptableObject architecture for events and game variables from the Unity Atoms package.
- Below is a very high level (not 100% accurate) overview of the system

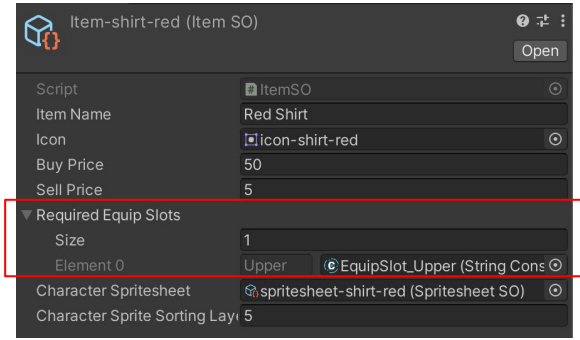




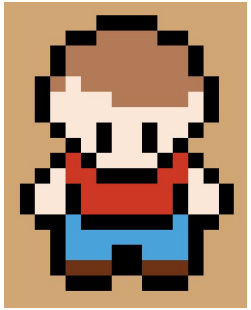
Equip slots are expandable in the future

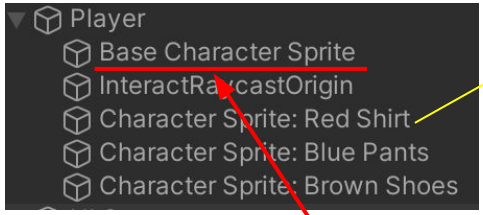


ScriptableObject reflects the inventory state in the scene

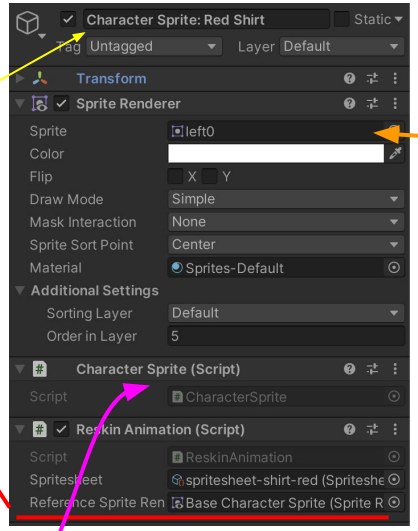


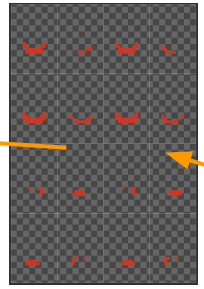
The data model supports items that take up multiple equip slots, but I didn't have time to implement the feature.

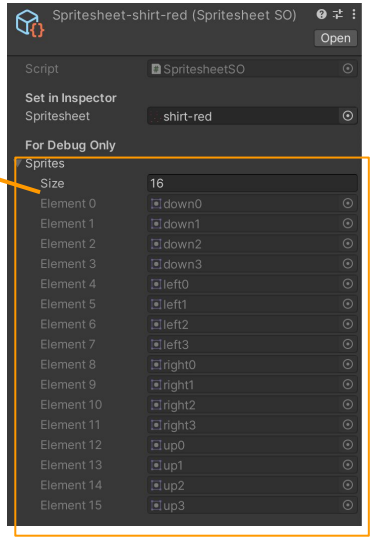




Equipped items are added as layers on top of the animated base sprite





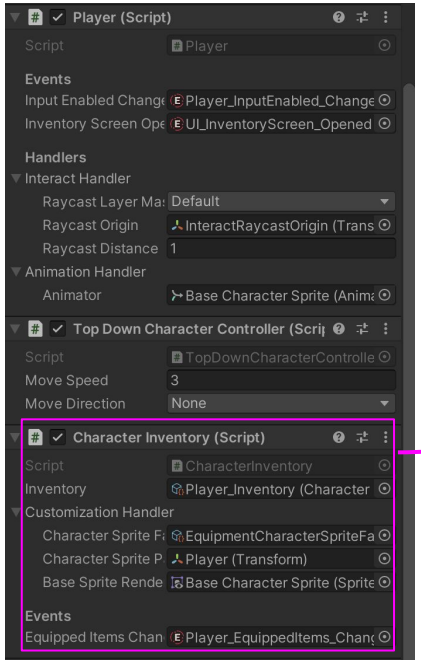


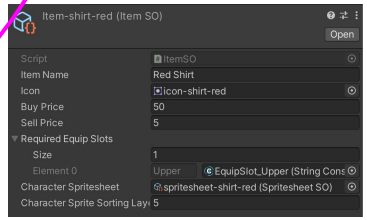
Spritesheet just stores all Sprites found in a Texture2d

Reference Sprite Renderer is in Base Character Sprite (the one with the Animator)

This is used to determine the current animation frame's sprite, so we know what sprite to use in our SpriteRenderer

ReskinAnimation can also be used to reskin the base character sprite, if Reference Sprite Renderer is null





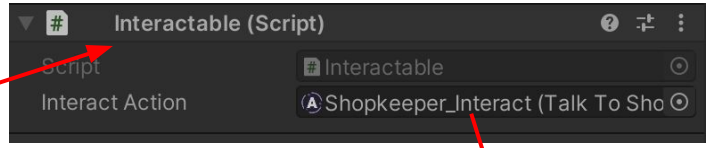
The spritesheet is defined in the item

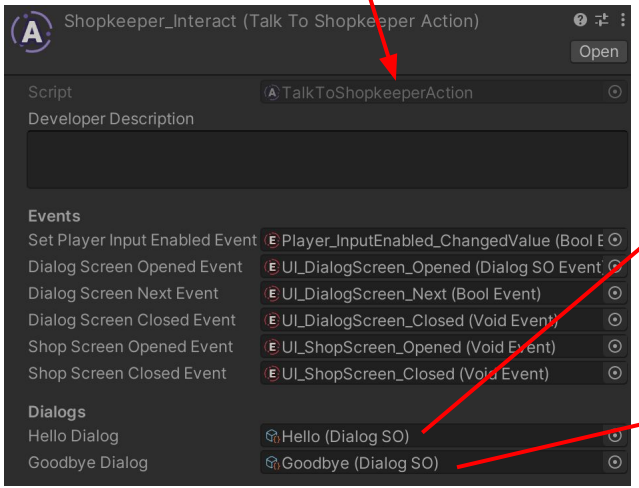
CharacterInventory handles "equipping" of clothes on the player when the Player_EquippedItems_Changed event is raised

It passes the equipped item to the CharacterSprite facade to simplify the setup of ReskinAnimation

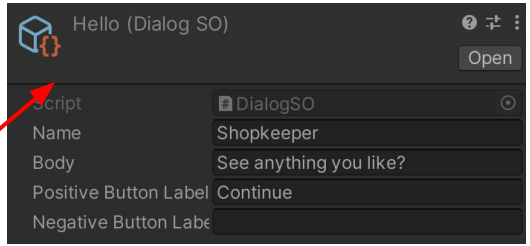
Shopkeeper has an Interactable component. When player presses the interact button, a raycast looks for Interactable objects that it collides with. Then its Interact Action is executed.

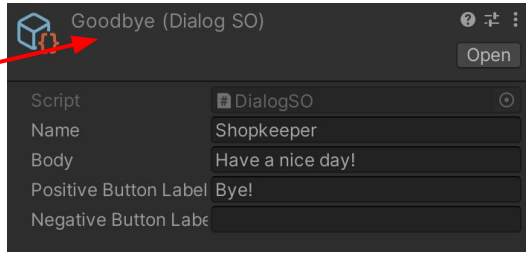
Actions can be coded to facilitate simple or complex dialogs. This one internally works like a simple state machine.



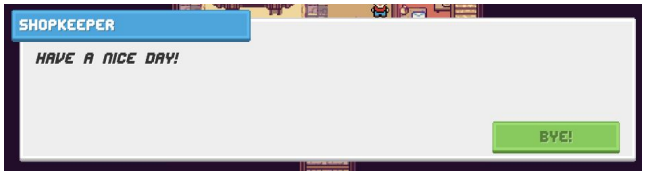


Dialogs are defined in separate ScriptableObjects

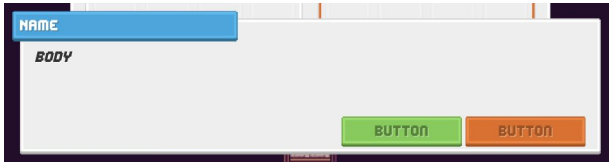


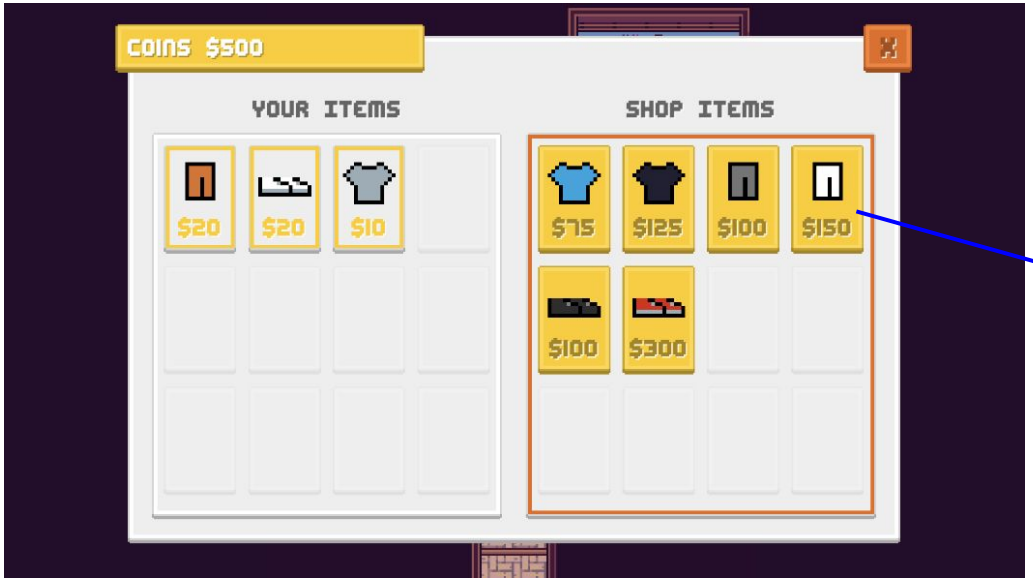


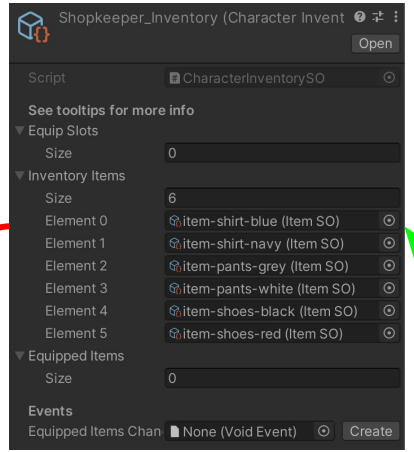
Dialog screen updates the view when the UI_DialogScreen_Opened is raised



System actually supports positive/negative response through the UI_DialogScreen_Next(bool) event but I didn't have time to put a confirm dialog in the shop action

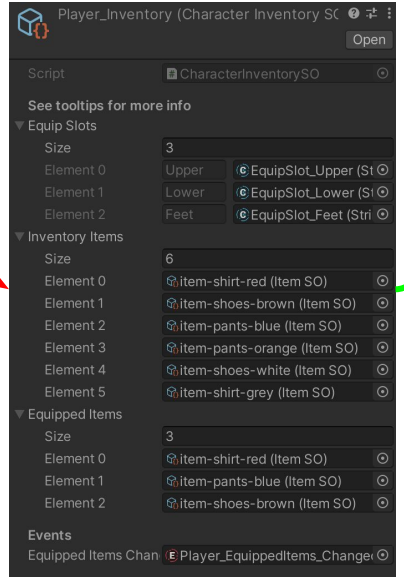






buy

sell



Shop buy/sell logic is contained in the shop buttons

Currently, each button has access to the inventory data models, which is not ideal

Better if the buy/sell transactions could be handled by a dedicated shop service object, but I didn't have time to refactor

So during buy/sell, each button facilitates the transfer of items between shopkeeper and player inventory, and updates the Player_Coins variable