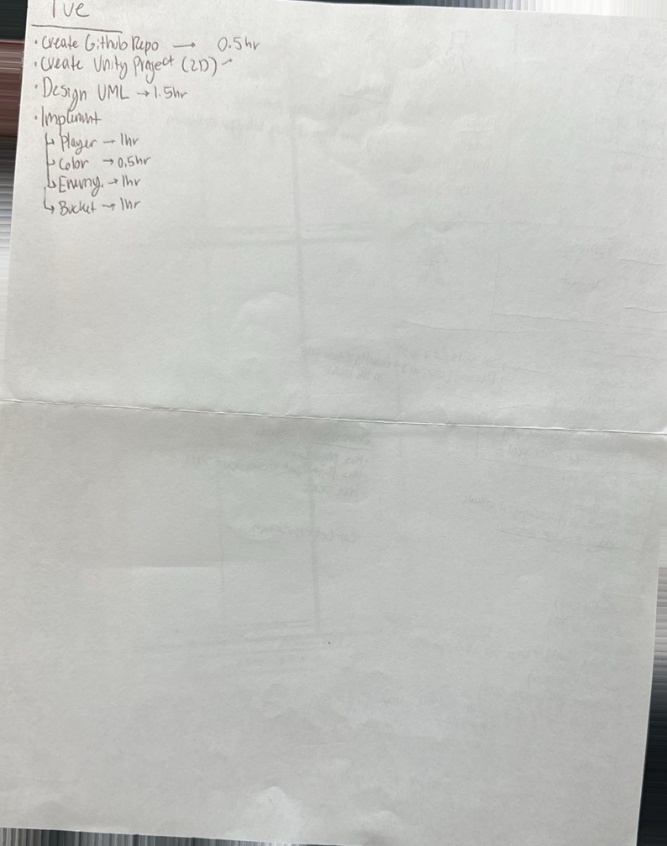
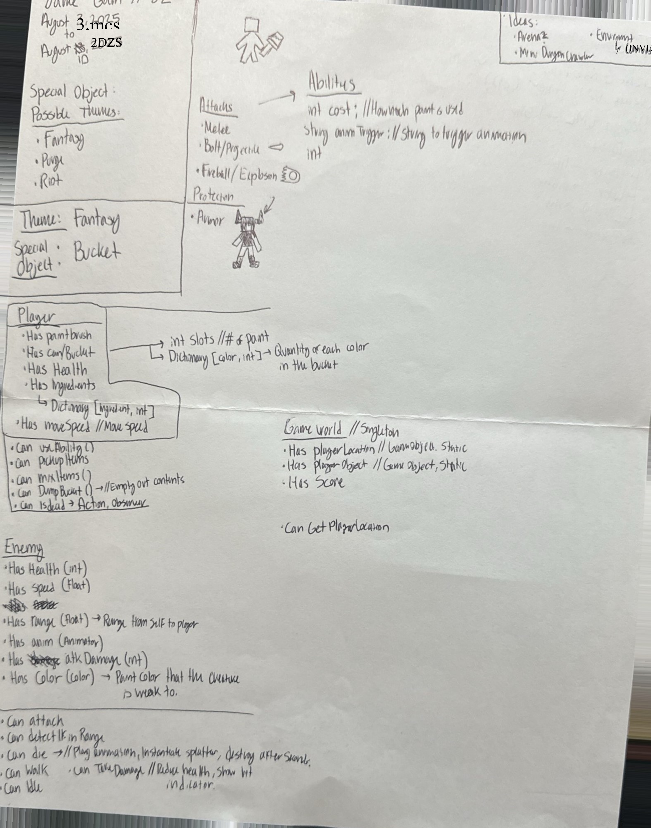
Aug 3:

Drafting some concepts for classes and designs for the character

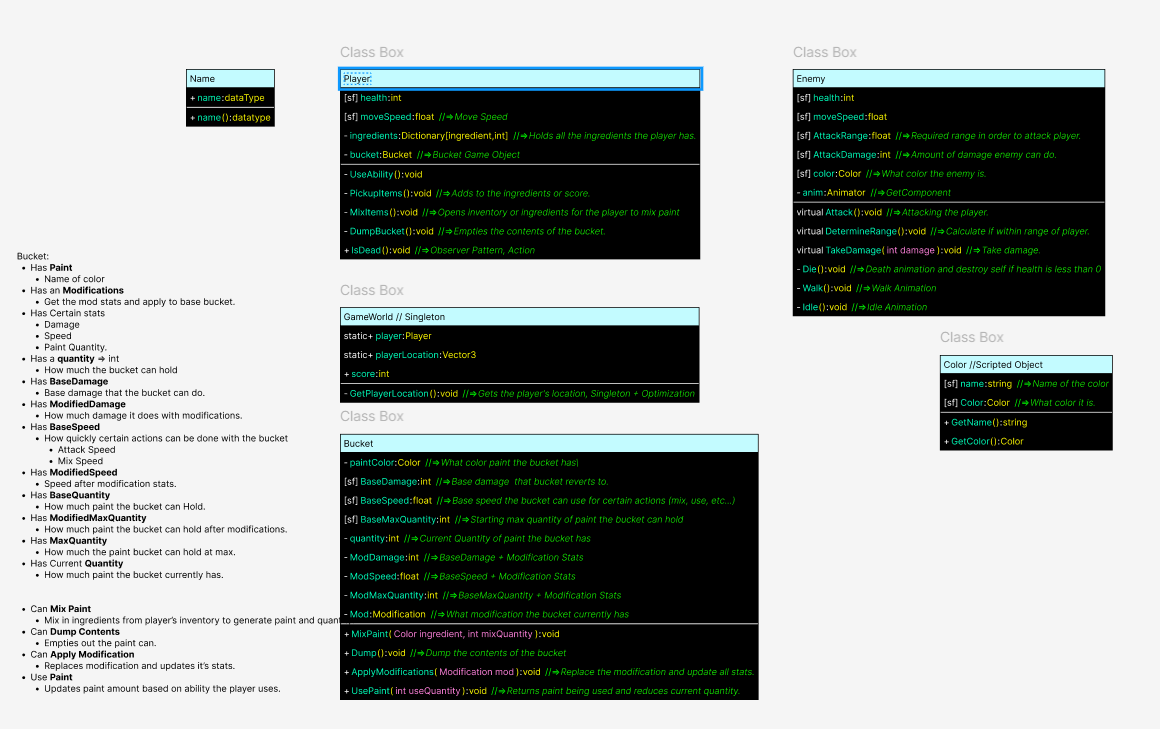
Drafting some abilities that the player may have.



August 5,

14:07

Designed a few class UMLs for starters.

****

16:00

Player is now able to move.

The player prefab has 3 scripts

The Player, which holds all the stats, and code for the player to function.

The PlayerAnimController, which deals only with the player animator. This is to ensure each class has a single purpose.

The PlayerInputController, which solely handles the player’s inputs and sets variables that can be called upon.



23:28

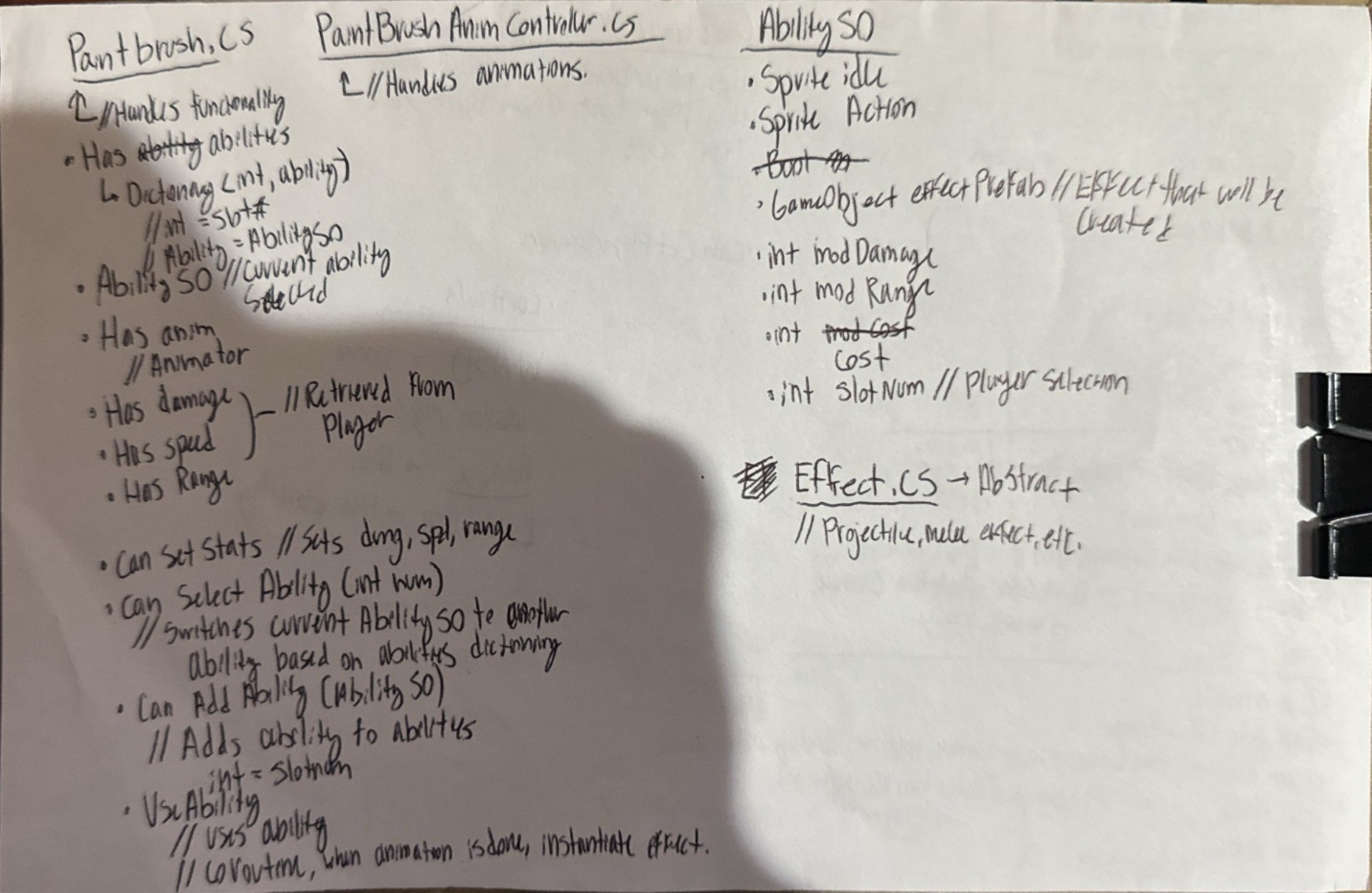
Came up with a UML for PaintBrush, AbilitySO, Effect.cs and possibly a PaintBrushAnimController.

Originally, it was a challenge to figure out how to design each ability to be modular, as well as having the paint brush activate each ability, while making sure to maintain the Single Responsibility Principle.

A solution was to make the AbilitySO a scriptable Object, allowing for easier modular objects for different abilities that the PaintBrush class can call upon.

Within the AbilitySO, it will hold a gameObject prefab, which will be a gameObject with the Effect.cs, or a derived class of Effect.cs.

The Effect.cs will be abstract, ensuring that derived classes will be created from it, and because of this, it will allow different effects to be responsible only for themselves, such as projectiles moving on their own, or special effects working on their own, rather than another class or object operating them.



Aug 6:

1:57

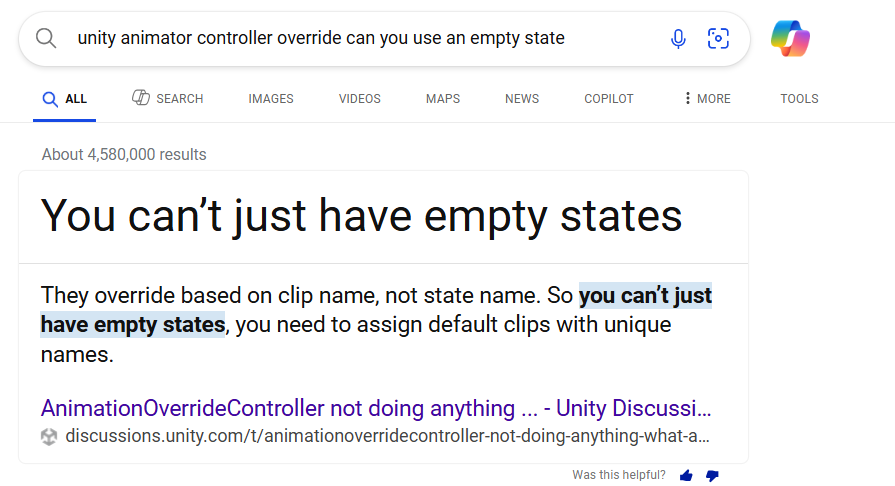
Paint Brush gameObject now utilizes an Animator Override Controller.

Had to research documentation and tutorials for using Animation.

[Animator Override Controllers Explained (Unity Tutorial)](https://www.youtube.com/watch?v=fsImNxsEUto)

[Unity - Scripting API: AnimatorOverrideController](https://docs.unity3d.com/6000.0/Documentation/ScriptReference/AnimatorOverrideController.html)

Apparently, you can’t have empty states in an animator.



For the Animator Override Controller to work properly, there has to be an animation clip that it can reference, not the actual animation state.

This has caused quite some headaches for a while, but once the problem was figured out, the paint brush can now set animation clips to a single animator based on the AbilitySO.

2:42

The player can now use the ability of the paint brush.

After the paint brush animation is completed, then the effect prefab is instantiated.



3:07

PaintBrush can now rotate and the effect is initiated based on the paint brush transform.

This process was easier due to having previous code from other projects.



3:54

Implemented the bucket.cs.

Additionally, created the ColorSO.cs.

The bucket can now get the Color from the ColorSO, and effectively make the paint on the bucket appear to be the same color as well.

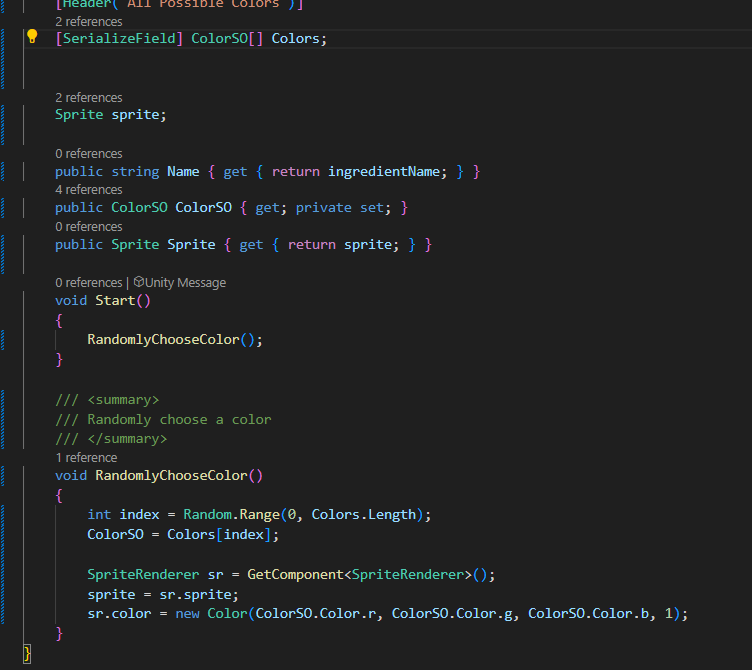


13:32

Adds Ingredients and a Dye Prefab to the game.

The Dye prefab is coded so that it has all the possible colors it can be, and randomly chooses a color upon creation.

This is to allow a more modular prefab, rather than having several branching prefabs with the only difference being the color.

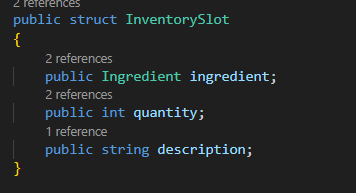
A screenshot of a computer

AI-generated content may be incorrect.

15:31

Inventory System is now implemented, player can now collect dyes.

After several attempts and research of using a Dictionary, I instead opted to use a struct instead, allowing for easier updating via utilizing an array.

A screen shot of a computer program

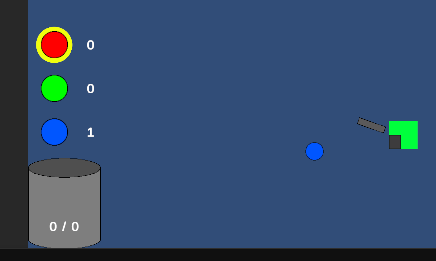
AI-generated content may be incorrect.

Aug 7:

21:11

Player can now only use an ability if there is enough paint in the paint bucket.

The bucket can now also mix paint and show the player that it has paint in it.

A screenshot of a video game

AI-generated content may be incorrect.



22:25

Player can now damage things with the effects created.

Projectile and Melee can now be switched.