**Developer Log**

**Updates:**

* Fixed bug where player can open the monster menu without other menus closing first
* Added boundaries along all sides of the map so player cannot move off camera
* Fixed bug where the enemy battler’s HP was not reset after a battle
* Added new enemy monsters that are buffed and represent the monster the player will get for beating the battle
* Added a screen for when players lose (all player monsters are at 0 HP)
* Added new move sets for each type of monster (each monster now has 4 move of its type with different base powers)

**Design Decisions:**

To fix the issue where the player could open the monster menu while still having other menus open, I decided to make the opening of the monster menu close all other menus. I did this by adding more methods to the UIManager class and having that also manage the monster menu, which I should have done in the first place anyways. I did not have to make any other design changes with it came to the game’s programming compared to how it was before.

**Technical Debt:**

One thing that I do not like about my scripts is that I probably made way too many properties of things public. If I had more time to work on polishing the game’s code, I would alter the architecture to get around any problems created by making these previously public properties private. I also know that I could have separated some classes out more, like how the player inventory class has a bit too much in it. To handle some of the things that class handles, I could have created something like a player inventory manager that was separate from the class actually holding the inventory data. Finally, I wish I had something that generated UI elements like buttons dynamically to generate when the player has more monsters they could buy.

I also have a big bug where all the monster stats are shared by monsters of the same type, which is due to the way I structured it with the factory pattern. I tried making it so new instances were created every time a new monster was grown, and it copied the data from the factory over to the new instance, but for some reason this did not help.

**Standard Documentation**

**Game Name: Monster World**

**Game Description Short:**

Become a monster farmer that grows monsters and raises them up to fight other monsters!

**Game Description Long:**

Monster World is a combination of a turn-based RPG and a farming simulator, where the crops the player grows are the monsters themselves and food to raise their stats. When they feel like they have trained their monsters enough, they can challenge the monster tamer to earn the right to buy different kinds of monster seeds. They will have to train up and consider the type of the enemy monster to knock them up with an effective type matchup!

**Credits:**

* Ryan Ruocco: Tomatoad Concept
* Ricky Bakersmith: Raccorn Concept
* Duncan McDonald: Giraffodil Concept

**Genre:**

Casual, Simulation, RPG

**Postmortem**

**UML Diagrams:**

Battle Classes:

**A picture containing text, electronics, several

Description automatically generated**

**Plant Classes:**

**A picture containing text, iPod, electronics, bunch

Description automatically generated**

**Other Farming Classes: A picture containing text, computer, bunch, different

Description automatically generated**

**Game Backend and Monster Classes:**

**Text

Description automatically generated**

Patterns and Dependencies:

The most prevalent pattern I used in the game’s programming was the factory pattern, which I used for countless aspects of the game. This includes monsters, their stats, their movesets, the plants they grow from, the food they eat, the plants the food come form, etc. By using the factory pattern, it made it really easy to keep on getting more instances of all these things for use in the game. It also let me set up these instances in a script in a way that felt like I had a database of everything in the game. I also used the singleton and flyweight patterns, with the singleton one being the one I used the most out of the two. Singleton was used for things like the player and game manager, since there would only ever be one of each and a lot of scripts needed to refer to instances of them to work.

Challenges:

Most of the challenges I encountered were between the POC and the VS, as I had to implement a lot of new features and actually add functionality that was working beyond a testing level. At first, I had trouble thinking of how to make the classes know about one another without making them monobehaviors and dragging in the game objects in the inspector, but then I remembered the singleton pattern. The singleton pattern was probably the glue that held the architecture together. Some examples of this was how every shop could fill in the appropriate plots since it knew about the farm manager in the game manager, every menu could tell the player singleton when it could not move, and the shops knew when to update their stock when looking at the progress manager. When I hit a problem where something was not working as intended, I would put in debug log statements to check for things like null values and confirm that properties had the intended values at specific times.

Reusability:

At first, I found it kind of overwhelming to try to make most of the scripts not derive from Monobehavior, since I had to figure out what the system would be like underneath all the Unity stuff. I am very glad that I decided to do that though, since it made it a lot easier to implement most of the features in the long run and keep the code separate from Unity. If I were to estimate, perhaps 70% of my code is not specific to Unity, and just specific to C#. However, I do think a lot of the code is specific to the game itself since a lot of it is directly related to the concepts of growing monsters/fighting with those monsters. I think I could reuse the techniques I used when programming this game to other games I make in the future, especially if they are a similar concept.

Maintainability:

I would say that my code is very maintainable, since it is really easy for me to add new moves, stats, monsters, food, etc. To add a new monster, all I have to do is make a sprite, make a new entry in the monster factory, make a new monster plant in that factory, and make it so the monster can be bought in the store. I would not have to make any changes to anything in the Unity engine other than adding the UI element that would buy the monster. If anything, the ease of adding a new monster, food, etc is what I am most proud of in my game architecture.

Did I Finish the Game?:

If I am being honest, I would not say that I have “finished” the game. I realized around the VS that I may have been too ambitious with my proposal, and I probably should have done something simpler. For me to say that I had finished the game, there would probably have to be more connecting the farm and battle systems, like how in Pokemon there is still stuff to do other than battling. There are also smaller game design things that I did not think of until it was too late, like implementing a currency system or more polish to the battle system. I did not do this yet because I was focused on getting the core elements in and programmed well. If I were to continue working on this project over the Summer, I would tackle these things right away, but for now I just ran out of time.