**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

**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 Diagram:

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:

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.