MDU112.2 ‘Simple Farming Game’ – Jordon Dodds

My process with this game was fairly similar to the first, the main difference being that now we were to implement two classes, instead of just 1.

Starting with the Crop class, the first step was to implement the variables without any data, then the properties were added and given the required data. Both constructors were added, but because there was only enough data for one crop, the data was repeated for all five crop slots, making the game unplayable once the class was implemented. These bugs were left until later, as I was under the impression I didn’t have the knowledge to fix them, just yet. The final step of implementing the Crop class was to add the Update function. As the game was unplayable at this point, I couldn’t see if it was working correctly.

Moving on to the GameState class, I started by adding AvailableCrops list and Money integer. I did not implement the dictionary at this time as I did not know what a “MonoBehaviour object”, so I moved on to part 6, which was the GameState constructor that takes the parameter that is the amount of crops. The constructor adds crop names to the AvailableCrop list, and gives the Money variable a value, which is the player starting currency.

Before moving on to part 7, I did some research on MonoBehaviour. My wonderful lecturer told me that ‘MonoBehaviour’ was to be used in the same was I would use an int or a string. With this knowledge I added the PlantedCrops dictionary, and decided to move on to bug squashing in the Crop class, as part 7 required more knowledge I did not know, and decided it would be more time efficient to leave that until everything else was working properly, as I intended to not finish GameState until Crop was working properly anyway, in order to avoid having bugs from both classes implemented and not knowing which ones came from where.

The first step was to make the game the game playable, which was done by removing a while loop, that was causing the game to crash when the player tried to plant a crop. The next bug was then the growth of the crops. Not only were the crops maturing way too quickly, but they were also not reaching full maturity. This was easily fixed by implementing an if statement, that gradually increases by the time since the crop was planted divided by the amount of time it takes for the crop to reach maturity, plus the former maturity of the crop, in order to make sure the crop I always at least as mature as it was before the function was called. The next bug that became apparent was the fact that crops were not dying. This was fixed by repositioning the if statements, so that instead of two separate pieces of code it was one big statement and changing the death chance to make it more reasonable. With this done, all that was needed was to comment the code, and differentiate the different values between crops. I decided to leave both of these until later, to make sure every function was being implemented properly.

Next I moved through GameState.cs implementing the different functions that were yet to be implemented. I actually had a fairly hard time getting through this part, and it took me a week or

two just to have everything implemented, but once it was, it was pretty easy to fix the bugs (if crop didn't plant, the problem would be with PlantCrop).

The game is played by the player picking a crop to plant and a tile to plant it on. The crop will then grow to maturity, unless it dies. The player can choose to harvest any crop at any time, regardless of if it's alive or dead, and will be given more money based on the maturity and whether or not it is alive.

Setting up Crop without any bugs was fairly simple, as it was fairly easy to see where a bug was coming from, based on what the bug is doing. In regards to time management, even with my problems, I still had plenty of time to finish the assessment so I would definitely say that I improved in that reguard. As for what went wrong, as previously mentioned, I definitely had troubles with the lower half of GameState.cs. I am positive that the best way to combat this is by actually planning out what my code needs to be doing before I start programming, as my wonderful lecturer keeps suggesting, and so for the next assignment, I plan on having all my planning for a class done, before I begin to implement it.