# SENG201 Assignment Report

Josh Bernasconi – 68613585 Jack Boulton 71037478

Structure and Design Choices

Our virtual pet application was first implemented as a command line application with a few, large, classes, to get a feel for how difficult implementing the required functionality would be. Once most of the functionality was implemented we decided to use the Model View Controller architecture to structure our code properly for GUI implementation. This allowed the model class *TamaModel* to become almost a data structure, it holds the state of the game as it is being played. The Model keeps track of all the player objects, the current day the game is in etc. and allows access to these through getters and setters. The Controller contains the logic for running the game and updating the View and Model based on user input. The View utilises a few GUI classes and provides methods for updating to the Controller.

We decided on using parent classes for Pets, Toys and Foods. Each of their children classes contain the stats specific to that Toy, Pet or Food. This cut down on code repetition and allows for easier inventory control because each item or Pet is an instance of its child class, so can be added and removed easily. This also meant we used *ArrayLists* to store and keep track of Players, Pets, Toys and Foods, as the amount of each of these is variable and changes often throughout the game.

We also decided in out GUI to use *JOptionPanes* to display messages to the player such as a pet dying or getting sick as they are modal. This means the main game is disabled while the pop up is open, this ensures the player handles the event at the time. We also found that once a pet’s actions have been used up, the easiest way to stop the user doing more is to disable the buttons.

We implemented the Toy class as an abstract class in order to have the toys all have a different break action, but decided not to use it, so changed it back.

Unit Tests

Three unit tests are used to test nontrivial methods for Player, Toy, and Pet classes. Methods in Food class were too self-explanatory for unit tests.

Thoughts and Feedback

This assignment was interesting and challenging. The time frame given was enough to complete the project without being too strenuous so long as constant work was done. It definitely increased our knowledge of the Java language and Object Oriented Programming, and was a good way to practice implementing a MVC. Teaching GUI a little earlier and to a greater extent would have helped complete this project faster.

Retrospective

We used GitHub to manage our code and it went well, for the most part, except for merge conflicts. Next time we could try a different source management solution to see what works best for small projects where both contributors are coding at the same time. Learning Swing took a while but once we decided to not use WindowBuilder and code the GUIs ourselves, it went a lot better. Implementing random events and dying went surprisingly well. Next time we start with a GUI instead of command line and build from there, if GUI was the target.

Contributions

Josh – 50%

One of my larger contributions was the initial switch from command line to MVC, which included a lot of refactoring of code. Another thing was teaching Jack GUI so that he could help build the remainder of the GUI. I decided on the structure of the Pet parent child classes and implemented them. We both contributed a lot to the MVC.

Jack – 50%

Code and GUI for the viewing and interacting with the Shop View. Junit testing and Toy/Food refactoring. I did the MS Paint designs for our lovely animal icons. Implementation of Random events and pet states. Both worked simultaneously contributing and editing most classes.