Seng201 Report

Reed Earl 73244439

Swapnil Bhagat 91415037

Application Structure

The structure of our application had the main classes Game Environment Class, which ran the GUI element and played the game, Farm Class, which kept track of everything related to the farm, and Crop, Tool, and Animal Market classes, each of which were instantiated on user command.

The Farm Class was the parent class of the four farm types, the plan was that if there was time, we would add unique methods for different farm types as an extra. For that reason, we left them as child classes even though we could have kept it all in Farm Class. So, while Farm Class had inheritance, it was not necessary (although it did somewhat improve modularity). It also kept track of the farm's crops and animals through two array lists, which was essential in many of the Game Environment's methods.

The next main Class was Animal Class; this had child classes of different types of animals - four to be exact. Sheep and cow classes had additional methods for milking and shearing, added due to our utility items 'Milk Master' and 'Shear Master' which automated milking and shearing at the end of the day. So, this was a good use of inheritance by us as it would have been easier to add more methods for each child class should the need have risen.

Crop, Animal, and Tool Market Classes were initially just one market class. But we found it easier to break it up into three markets with clear objectives for each one. We believe this also helped with the user experience as all available actions were on one screen, making the gameplay more direct and user controlled.

Finally, there is Farmer Class which was just a simple class that kept track of the farmers name and age.

All of these major classes had the parameter for object 'GameEnvironment'. As such, the core logic of the source code was 'passing around' this one object and altering its attributes as need be. Originally, for the command line game, there was an additional 'UI' Class, which consisted of many methods that enabled communication with the player through the command line. This was a good design choice as it made it easier to merge the game logic and graphics interface windows by simple replacing all instances of where 'UI' was instantiated to a graphics window, with only some parameter changes taking place.

Test Coverage

The unit test coverage for the main game package is 88.6%, which are our non-GUI classes with the biggest loss being from non-essential getters and setters being tested or small methods that were just converting smaller objects to strings or adding to array lists. Which we were happy with not testing. Our GUI test case coverage was comparatively low at 58.5%. This was due to a lack of knowledge on how to effectively test a user input scenario. Much of this percentage was a by-product of testing the GameEnvironment class, which was a surprise.

Project Feedback

Should have started much earlier on the GUI. The labs for it were a bit disappointing in that it lacked quite a few of the basics. Even recommending a YouTube channel would have been more help, there are some good ones online but there are lots of terrible ones too. The GitHub lecture would have also been useful earlier/before the project.

Retrospective

The main classes were done quite well so when we got to the GUI it just was our ignorance of what we could do in the design environment than anything else that slowed us down.

One of the biggest hurdles was GitHub. Forgetting to pull before pushing or working in the same class and pushing similar changes made a bit of a mess. Which is bad communication (and a lack of knowledge) on our part too. Just had to be clear where each person was working, bit of a time waste fixing preventable errors. But then it was our first time working on a group project and we did learn by the end what worked and did not work.

• The effort spent (in hours) in the project per student.

Both 40+

• A statement of agreed % contribution from both partners.

50/50