SENG201 - Software Engineering Project I

Farm Simulator Game

Megan Steenkamp (Student ID: 23459587)
Lewis Marshall (Student ID: 71552853)

Application Structure

Our application is structured into three different packages for the game logic, the GUI and the unit tests. The logic of the game is controlled by the Game Environment. The GUI package only communicates with the Game Environment class, achieving low coupling between the two packages. As previously noted, the Game Environment class controls the game logic, instantiation of the Farm, Farmer and General Store, and placing objects where they belong. A Farm has a Farmer as a property and manages the Crops and Animals. A Farmer controls the Items. The General Store class is responsible for instantiating the Animals, Items and Crops that are 'sold' in the game.

Our Animal, Item and Crop classes all have one subclass for each type of item they can take on. For example, the Animal class is extended by Pig, Chicken and Horse subclasses which allows an instance of a Horse to be created without having to pass in parameters each time. The Animal, Item and Crop classes also all implement the 'FarmItem' interface. The implementation of this interface allowed the General Store to be much more succinct. Without an interface, we would have needed separate methods for selling an Item, selling an Animal and selling a Crop. With the FarmItem interface, the sellItem() function can sell an item belonging to any of those classes.

The various classes in this package are all managed by the 'ApplicationManager' class. This class controls the flow of the screens that are displayed. As mentioned above, all GUI classes will only communicate with the Application Manager class and the Game Environment.

JUnit Testing

The unit testing has been agreed to be split equally in half. Megan was responsible for testing the 'Farm', 'GeneralStore' and 'Animal' classes. Lewis was responsible for the testing of the 'Farmer', 'Item' and 'Crop' classes.

Megan: To identify test cases for my classes, basic equivalence partitioning has been used. Where feasible, all methods have been tested with a valid input and with an invalid input. The JUnit tests for the 'Animal' class have 100% code coverage. The tests for the 'Farm' class have 95.9% code coverage. This is due the removeCrop() function having a nested conditional statement within a for loop that is not being marked as covered. There is, however, a test that does enter this statement, successfully removing a crop from the farm. The 'GeneralStore' class tests have 70.3% code coverage. This is a bit lower as the sellItem() function has a large switch/ case block as it is

responsible for selling an instance of every type of Animal/ Crop/ Item that is sold in the store. I have written just one valid test for this method and so it does not cover every possible scenario, significantly decreasing the code coverage in this class. I think in this case this code coverage is sufficient as it tests both valid and invalid inputs. Overall I am happy with the code coverage of my unit tests.

Lewis: The JUnit tests for the 'Crop' and 'Item' classes have a wide code coverage of between 80% and 95% for these classes, testing the set and return functions for the various attributes. Additionally, the validity of processes were checked when necessary. Perfect test coverage has not yet been implemented due to the time constraints of the project but would be an imperative step to complete before a formal release. The Farmer class has not been fully covered by testing, the coverage level is sitting significantly below an accepted level. Testing has been implemented for the rudimentary getting and setting capabilities of the Farmer class. Due to time constraints, further comprehensive testing has not yet been implemented but would be a necessary task to complete before an official public release.

Project Retrospective, Additional Thoughts and Agreed Contribution

The project has been finished to a very high standard. The game works as intended and incorporates all of the requirements set out in the initial specification. A key factor to the success of the project was the thoughtful planning and design of the game at the beginning of the project. Careful thought was put into structuring the classes and detailing the exact attributes and methods that are to be used in the project. Throughout the project we have both improved our knowledge and understanding of Java and Object-orientated programming.

Developing a project of this scale doesn't come without challenges. Coming into the project, there was a significant variation in the skill level and development competency within the team. This made it difficult to coordinate the steps in the development process and allocate tasks that accommodated the level of competency the individual team members were at while still ensuring a high quality product was produced. At times, different communication styles and ideas regarding time management clashed resulting in confusion and misunderstanding. These are all learning experiences and the team has grown from the challenges that had to be overcome.

There was a notable discrepancy in the levels of contribution made by individual team members to the project. It was agreed that Megan Steenkamp had contributed 70% of the work completed in designing and developing this project. Lewis Marshall contributed 30%. Megan has contributed 58 hours of time to this project. Lewis Marshall contributed approximately 25 hours to the project.

To ensure the success of a future project, improvements could be made in the disclosure of skill levels, expectations, and work styles in the introductory stage of the project. This would ensure the entire team is in agreement and understands each individual's strengths and weaknesses. The project can then be divided to play to each team member's strengths.