SENG201 (Software Engineering I) Project: Farming Simulator

Student Names	Sarah Bealing	Inga Tokarenko
Student ID Numbers	11299330	75980575

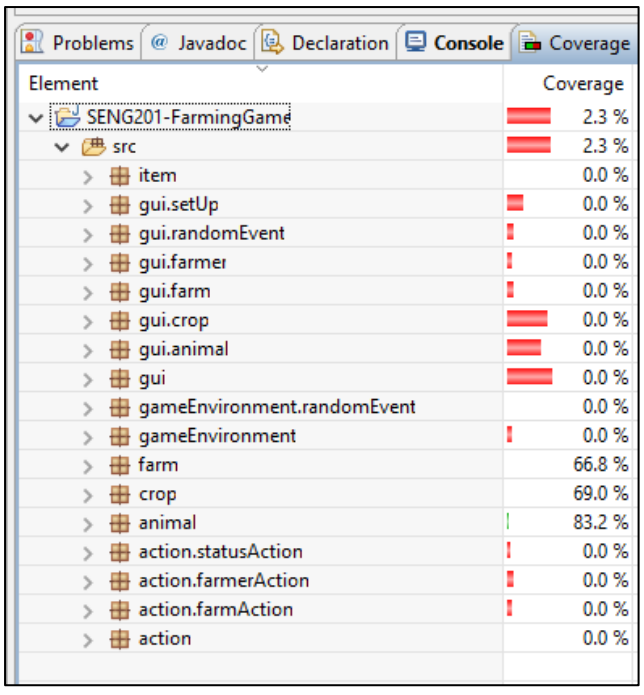
Design Choices

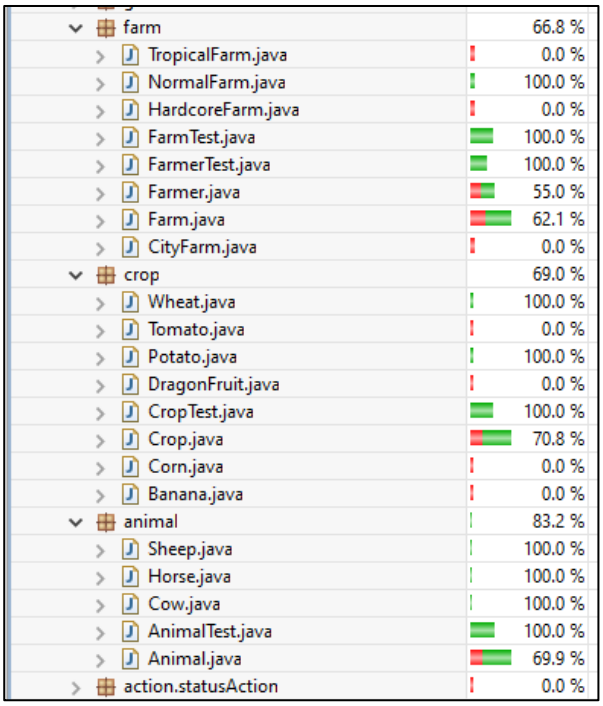
At the start we broke down all the main classes that the project required. We decided to have a design that consisted mainly of main classes, subclasses and classes which would be used as communicators between classes. An example of such a class is GameState. GameState contained the state of the game and all the required subclasses which would be used in running the game. All of the action classes would use GameState to activate certain operations in the game that require the instances of the objects inside GameState.

We also used a main control class which would run the game and activate the actions of the game such as NextDay or going to the Shop. An example of such a class is GameEnvironment which would handle the start and end of the game, as well as everything in between. In the command line application and GUI, GameEnvironment acts as the control panel of the game.

It is clearly shown in the UML Class diagram that inheritance was our main way to create classes. Where we used the subclasses would either be a specific animal, crop, or action of the game. Our design led us to low coupling and high cohesion which we consider a great outcome.

JUnit Testing





The above images show the coverage of our JUnit testing. We wrote unit tests for smaller, basic classes and achieved 66.8 - 83.2% coverage for the farm, crop and animal packages. These small, basic classes only make up a small part of the src folder so overall, we had a low percentage coverage of 2.3%.

Reflections

Overall, we are happy with the project and found it a great learning experience. Through completing the project and learning from each other's mistakes, we have improved our understanding of Java, UML diagrams and Git repositories. We gained experience in planning which will be helpful in upcoming projects.
The project allowed us to experience working in a positive and understanding environment for a group project as we were both helpful and supportive towards each other.

What went well?

Starting the project, a week early gave us plenty of time, meaning that we didn't require any all-nighters and kept a constant weekly improvement without the pressure of other courses.
We managed our communication with regular calls, taking time to check up on each other, organising when we were going to call each week and being ready for the call.
At the end of each call we would set out what we wanted to achieve next and try our best to complete those tasks by an agreed deadline. We broke the project down into small stories to complete weekly.
Using Discord for communication, Creately for our UML diagrams, a Git repository, and shared Google Docs was a great way to keep all the project work together in a shared space that we both had access to at any time or place.

What didn't go well?

We didn't fully understand how GIT works and what kind of git command lines to use, meaning that we often had to seek help.

Improvements for next time

We would have liked to use a wider variety of class types, for example interfaces and more abstract classes. Creating more subclasses for actions could also be helpful.
Next time we should update our UML diagrams constantly as our code is developed instead of updating it all at the end. We should also build our Javadoc as the code developed because writing all the Javadoc comments at the end was time consuming.
We'd like to learn how to implement JUnit testing over a larger part of the project.

Effort Breakdown

	Hours spent on the project (from 14th April to 21st May)	**Agreed percentage contribution**
Sarah	60 hours	43%
Inga	80 hours	57%
Total	**140 hours**	