SEG201 Project Report

Authored by:
Harry Collard : 71662706
Kenzie Tandun: 34427482

Structure
We implemented a structure where all classes are connected to one Game Engine class, this streamlined our workflow and allowed us to run both the command line program and the GUI with the same code, which meant all updates we made when changing the program logic were transferred to both clients. Additionally it would be trivial to add more game clients in the future as they only need to communicate with the API provided by the Game Engine.

Inheritance
We noticed that both the classes Ship and Crew Members had some common attributes. Therefore we constructed a superclass called Unit with health and the name methods being common to both. However Crew Members also have fatigue level, luck and actions which were then applied in its class. This was useful because it allowed methods to be shared between all subclasses meaning less duplicate code. Additionally it was useful for maintainability, for example when running tests at the end we found that health could go below 0 when reduced. This was easily fixed by editing the reduce health method in the super class. If inheritance was not used each of the classes would have to be edited individually, resulting in many lines of altered code and a lot of wasted time. A similar approach was used for food and medical supplies as they both have a price, a name and a healing amount, by using a superclass called Item our code became less congested and therefore easier to read while also becoming significantly shorter.
We then decided to set these super classes to abstract classes this was because there was never a situation where we had to create a new instance of Item or Unit, only create instances of their subclasses.

Collections
Collections were used significantly in the project, in particular ArrayLists were utilised to store multitudes of different variables, such as lists of crew members where each element is a crew member object, a method could then be performed on a crew member using their specific index i.e putting to sleep. ArrayLists were also used to store button objects, these were created using for loops which allowed us to easily refer to the button using the index. To store crew's consumables, TreeMap was used with the Consumable object as the key and the quantity as the value, making it easy to keep track of crew's inventory when the player buys items at the Outpost and also use items when commiting actions.

Unit Test Coverage
All of the classes except CLI and GUI clients has above 80% coverage. We managed to get a large percentage coverage for our tests because a wide range of tests were created. Overall though we had a 31.4% test coverage. This may seem low however this was due to not being able to test our GUI which is very large class. On top of that some class's methods were located in multiple places. For example we tested the space plague class through its random events interface which we found to be working. So therefore there was no reason to retest the class on its own as it had already been tested. As a whole we have an extensive set of JUnit tests that cover (where applicable) the entire program in effect we believe that we have reduced a large portion of bugs.

Feedback and thoughts

Overall the project went quite smoothly, by implementing a draft UML we were able to plan the structure of the project which really helped to get an idea of the project layout. The project as a whole was a useful experience in collaborating with other developers and managing source code as a group. Git was our choice for version control and it proved to be very valuable as the code grows. Although we never reverted back to a commit, the fact that we have a safety net to fall back on whenever we implemented something is a nice thought to have. By using external 3rd party tools, we could also generate the data stored by git to see other interesting data such as contributions by each member, who wrote this particular line, what is the most common time someone commits a change, and so on. The project was also a good practice to get familiar with concepts that are the strength of Java language such as inheritance. Additionally, we also implemented the save and load feature which introduced us to using 3rd party module to read and write JSON files.

Issues we came across were that once we finished the game we found that there were many unforeseen bugs. This led to a large amount of time wasted as we subsequently had to backtrack and locate these bugs. An improvement for next time is to write our tests at the start, this would allow us to catch bugs as we created them because we would be testing it at the same time. This therefore would make our workflow much more efficient as we could get the code right in the first place rather than racing to finish the game only to find its not quite right.

Although it was very hard to judge we estimated each of us spent around 100 hours on the project. Around 4 hours were spent on working on the UML class and use case diagram, 6 hours to write JUnit tests, another 6 hours to write the Javadocs and the rest was spent working on the code with GUI consuming a big chunk of our time.

Contribution agreement: 50:50