SENG201 Project Report

Simon Lorimer 34339189 | Sam Johnson 77631789

Structure & Design Choices:
In the early stages of the project, we planned out how we wanted to approach things. We started by formulating a basic UML design to get an idea of the classes and methods that we may need as we go forward. As we moved into the development and implementation of the program, we found that we needed more properties and methods, and that the UML that we had planned on wasn't being followed (see below 1.).

The command line structure is based around a few main classes: Adventure, Player and Action. Adventure holds all game information, including the Crew Member's details, Ship's status and whether or not a part can be found on the current planet. Player is used to perform actions for the Player that don't cost any Crew Member actions, and Action is used to perform actions on a specified Crew Member. The game also has many small side classes filling in the inner details of the game to be called from the main classes. This can be seen in the UML diagram. We used ArrayLists which worked well for our classes. These are used in the Adventure for Crew Members, Ship for the inventory, and Shop for the shop's inventory.

When designing the CrewMember class, we initially thought of having a normal class and hard coding in the six different types of Crew Members. However to have better coding practices, we made the CrewMember class abstract and created six classes that initialize their different values. Our favourite type of CrewMember was LuckyMan. His health, crafting and treasure hunting skill are all randomly generated.

We also did the same for Item. We changed the Item class to abstract and implemented a Food, Medical and Part subclass that inherits values from its abstract class. For Item we implemented a "Quantity" value, which meant that we could "stack" items, rather than displaying the same information over and over again. This can be seen when you view, buy and sell items at the shop, or view the Ship's inventory. We faced a few bugs with trying to implement this, with changing the information in the inventory also changing quantities in the shop, but this was fixed by creating a brand new Item class when appending it to the Ship's inventory.

Unit Test Coverage:
The unit tests implemented covered approximately 19% of the program. The reason for this being so low, is that a lot of methods output to console, rather than returning a value. We would have been able to increase the coverage by making functions return rather than output to console, however void functions were needed to keep cycling through options. No unit tests were written for getters and setters, as were very confident that this would always perform the given operation of the method.

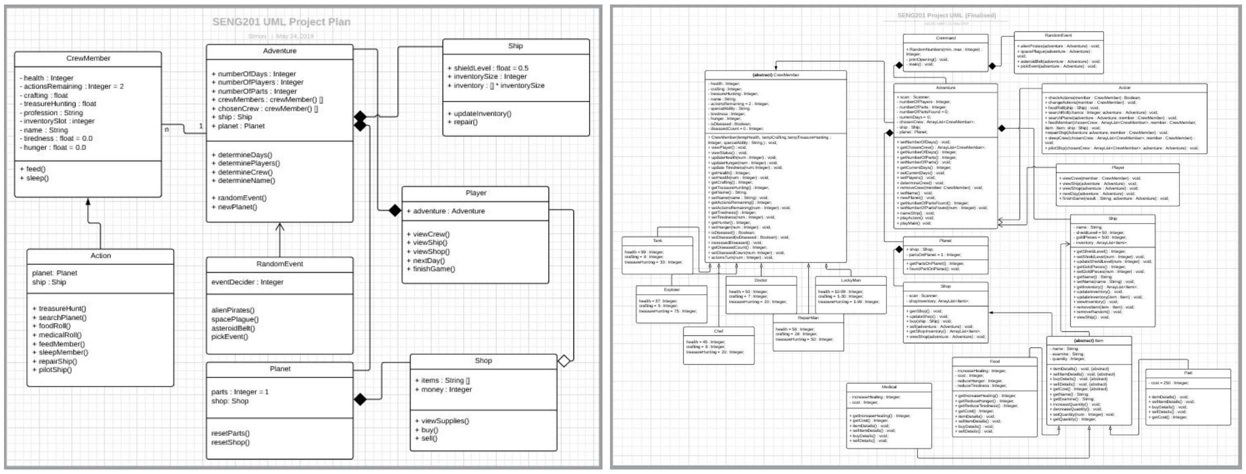
Thoughts & Improvements:
Getting the chance to work with someone towards an end goal made it much easier to put time into the project. From early planning to learning how to implement the GUI offered great opportunities to improve our planning and coding skills. The project took a lot of time and we ended up deciding to put our efforts into completing everything other than the GUI as we may have not have been able to finish it in time. Overall designing and choosing how we wanted the game to run while having good guidelines to follow set up a challenging but completable project with better commitment and planning.

From an early stage, the planning of our project went well, we had a UML diagram to follow making the early writing of classes go smoothly and gave us a good idea of where we wanted to go with the game. We initially planned that Simon planned on implementation of Command Line and Sam would look at/start working on the GUI. We knew we would essentially need slightly different code for each. Sam did manage to implement the starting screen, character selection and the basics of the GUI screen layout (see below 2.). Due to Sam being unable to commit a lot of time to the project in the later weeks, we decided it would be unlikely that we would complete the GUI in time. However, with a fully functioning game through the command line code, utilizing a few main classes each calling subclasses to fill in the game information, the command line was finished. With randomly generated special events, and crew members we managed to make a game that seems quite balanced. The biggest improvement for the next project would be looking in assigning more time for the GUI.

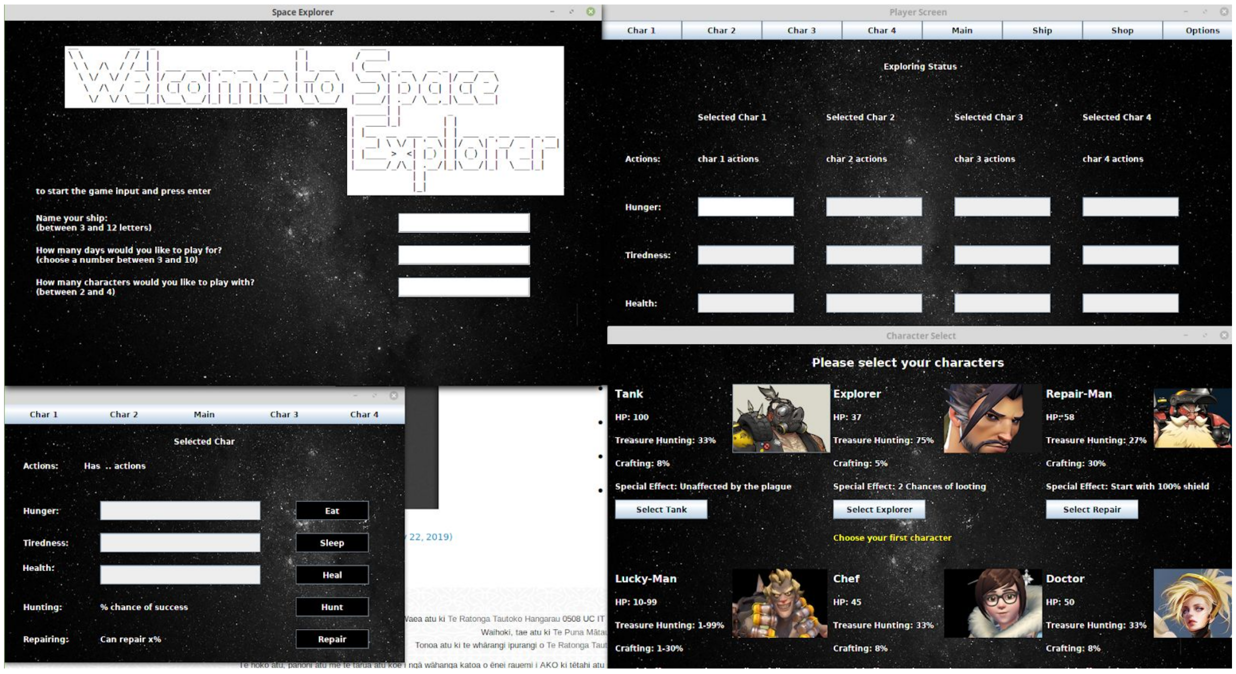
Effort:

Task	**Simon**	**Sam**
Writing UML & Planning	50%	50%
Implementing a Command Line Application	85%	15%
Writing Javadoc	30%	70%
Writing Unit Tests	90%	10%
Report	50%	50%
Agreed Contribution Split	**65%**	**35%**

The total amount of time spent on the project collectively was approximately 100 hours. We agree that the agreed contribution split should be Simon 65%/Sam 35%.



1. *Pre-implementation UML vs. post-implementation UML*



2. *Implementation of the GUI by Sam*