SENG201 - Project Space Explorer

Saahil Hari - 58536578
Isaac Walton - 33627090

Application Design:
Our project is built up of a number of classes that are passed into and used by one main game class. We quickly identified that crew members, crew, ship, planets and items were all worth building independent classes for, and that these all made sense as far as thinking about how the different parts of the game needed to interact. These classes were mainly identified by way of a UML class diagram. With this diagram we figured out which of our classes would be able to best make use of inheritance, and how our classes would need to interact with one another.

When it came to the crew members, it was apparent that all of the crew member types would share a lot of similar functionality, all while having specific functionality of their own. As such, we built the each of the crew member types to use the main crew member class as a parent to inherit these common functions from. Additionally, as each crew member needed to have a specific type, an object solely of the crew member class would never need to be instantiated, so it was decided that it made sense to make the crew member an abstract class.

The development of the item classes followed a similar process to that of the crew member classes. The two classes of items (Food and Medical), had clearly different purposes in the game, but would still share a significant portion of their functionality, so we decided to have these attributes be written into an abstract item class, from which both of these classes inherited their functionality.

Use of Collections:
Collections were used in a number of parts of the program, specifically ArrayLists were used to store our crew members in the crew class, along with another ArrayList for the items owned by the crew. ArrayLists were also used by our main Game class to store a list of the planets available to be visited by the player and another ArrayList of items it was possible for the player to find or buy.

Test Coverage:
We have written JUnit tests for the majority of our smaller classes, which has resulted in roughly a 20% coverage of the entire project's code (including the GUI application). The coverage for those classes tested ranges from 50-100%, with those on the lower end of code coverage being there due to not having a number of their getter and setter methods tested. The classes with the most code coverage are often the subclasses that inherited the majority of their getter and setter methods. The more important class functions were tested using a range of values that were possible for them to receive, resulting in all parts of that particular function's code being passed over by the test.

Project Feedback:

Isaac:
The project was a good opportunity to put into practice some of the design tactics we'd learnt in class. Additionally, the experience of building a program on a significantly larger scale than what we've done so far at University has helped put into perspective how important planning and time management are in these kinds of projects.

Saahil:
I felt that the while the premise of the assignment was good, the limitations provided in that we were required to use Swing was not. I feel that if this assignment were to be run in the future, it would be more beneficial for the students' knowledge to instead encourage the usage of JavaFX and its FXML-based UI, as it is a more modern approach to UI design. Doing so would not only teach students Java, but also XML, which is widely used in the industry. In addition to this, JavaFX and FXML work well with MVC.

Project Retrospective:

Isaac:
On the whole, the project went reasonably smoothly. However, we could have made more use of our time during the 3 week break to get well ahead of the project, instead of leaving it until after the break, to be done alongside assignments from all our other classes.

Saahil:
Over the course of the project, progress went more or less without too many issues. As Isaac has already stated, we could have done more work on the project earlier in the semester, which would have lessened the workload later in the semester. Due to us leaving certain aspects late, certain features that we had planned via our UML were not implemented (such as saving and loading). This is one of the reasons our UML does not completely reflect the system. Another reason for the UML not being completely accurate is that due to the nature of the project, there were times when we found ourselves needing access to data from other classes. As a result of this, we needed to create new getters and setters for variables in classes, as well as create entirely new variables in certain cases. Something that we did that saved us a lot of time while creating our GUI was create a generic list item selector screen. It was initially a CrewMember selector, however we realised that due to how we were planning on implementing some of our actions, we would need to be able to select more than just crew members. In order to save time, rather than creating a new GUI screen for each of these selectors, we stripped out all of the CrewMember references in the crew member selector screen, and instead made it take a Generic when initialised. This allowed us to reuse the screen for all of the times we needed to select an instance from a list.
The music that was used in the game was created by myself using a software called OpenMPT, and samples that have either been drawn or generated by myself.

Effort and Contribution:
Saahil Hari - ~45 hours, 60%
Isaac Walton - ~30 hours, 40%