SENG201 Space Explorer Project Report

Mayuko Williams 41163527
Azeeza Mohammed Arif 54897803

Structure

In this project, we have made classes of normal class, abstract class. enum class, and sets of parent and child classes.
Game class is the main class where it controls the game flow. Other classes are connected to this class. Spaceship
Abstract classes are made to provides their child class their variables to have. For example, Item abstract class was made so that different types of items in this game can be in the same class which is the item class.
Enum classes are created to make switch statements where needed. It provides selections to switch between. Some classes extends to their parent class because they need to inherit properties of the parents class.
Inheritances are made where valuables from parent class are inherited to the child class. The CrewMember abstract class is the parent class of all the crew members in the game with different occupations. The child classes has all the variables that are inherited from the CrewMember class. It is the same case with the Planet abstract class and all the planet child classes. Item Class inherits to every items in the game. From Item class, the items are divided into Medicine, Food, Part and Money which inherits variables from the Item class. They are divided as their methods in their class differs. Lastly, from the Medicine and Food class, different types of food items and medical items that inherits their parents.

Collections

In this project, we have used collection of arrays. It is used when we needed to have certain objects grouped together and iterate through each object. We have chosen arrayList type because we do not need special methods like Stack class where I can pop from the top of the list, or LinkedList where Deque and Enque methods are crucial.
In the Spaceship class, there are list of crews, list of pilots, list of inventory items and list of spaceship parts. Spaceship parts list and inventory list are separate lists, so it is easier to keep track of the number of parts collected. List of crews are made to iterate through it to update crew status and checking the number of crews. The arrays are used in the outpost class to add number of items into the inventory list. List of items are made and added to the inventory when they are purchased. Lastly, arrays are used in the planet classes to store items that can be picked up when search action occurs. All the items in this game are in Item class, therefore parts, money, food and medicine objects can be in one list.

JUnit testing

The Unit testing coverage is 16.3% for our game. This is because we did not test every methods we wrote. For example, not all getters and setters are tested. We have tested those methods for CrewMember class and the tests for those were successful. Therefore, we decided not to test every single one of them. Some other tests like buying each item is not tested because they are all in the same form. If we are allowed more time, we would like to check all of them so we did not put water item in the buy apple methods using JUnit testing, but it is visible by reading the code.
We wrote the code for this project with one method having to do multiple functions, which made the testing difficult to create. However those big methods have small methods inside, that have been tested.

Feedback on the project

This project was very challenging for us because we are beginners in java. Therefore, from the planning phase, we found this project challenging. It took a while and it was not easy (with Eclipse crashing every time we tried to create a GUI) but we did our best, it was hard to understand each others' code/work as we had different knowledge and ideas on java and how to control game flow. Putting out code and GUI to work together was very time consuming(very had to debug as we did not know much about GUI). Tutors and google were very helpful (tutors more than google) , it would have been nice if we had more tutorial sessions (at least during the project weeks). At the end, we managed to do this project by dividing our work so that one who is good at certain part do that part of the project. However, this method costed us more time as we needed to understand each others work after coding is done. And if our game flow ideas ended being different we had to start from the scratch. But we learned a lot. As frustrating as it can get when our game is not work as it should and we do not know why, it was still fun putting it all together.

We experienced how a small game creation takes place and understand how games are made. Implementation of the GUI and class codes for the actual game took so much longer than we anticipated. We clearly underestimated that.
We learnt how planning and time management in Software Engineering is very important as we plan, code and test our project. If we could have written the code more tidier, testing could have been simpler. If we could have planned out better, we could have saved some time working as a team. We will definitely want to improve in those areas next time we do a project like this one.

Hours spent, about approximately 180+ (about 100 hours each). Agreed Contribution 50 – 50.