Euan Widjaja ewi45, Timothy O’Neill teo31

MODULARITY:
We began our project by discussing how we wanted to implement some of our classes. We had five core classes at the start, as stated from the projects specs: GameEnvironment. Crew, CrewMembers, FoodItem, and MedicalItem. Creating these main classes made it easier to visualise the flow of functions between the classes and how the game would possibly look like. Our Crew class was almost as large as the GameEnvironment class, as seen from the UML diagram. This is because every call from GameEnvironment is mostly through Crew because this class connects GameEnvironment to some methods in classes such as CrewMember, FoodItem, MedicalItem, SpaceOutpost, and Planets. This helped preserve modularity in the functions so that there was not huge lumps of code in the methods of GameEnvironment. Connecting the classes to Crew also made it easier for some actions like buying items from the outpost and items bought from the SpaceOutpost can be added directly to an ArrayList within Crew.

COLLECTIONS:
Crew holds ArrayList collections of food items and medical items a crew has, a main crew members ArrayList, containing the crew members set at the start of the game, and variations of that ArrayList such as: dead members, unhealthy members, hungry members, tired members, and infected members. These variables are set when clicking the next day button in the spaceship window and the planet window which affects health, hunger, tiredness, and the infected status of a crew member. We are aware that having this much variables that are almost alike may seem inefficient, but it makes it much easier for the warning screen to obtain these particular members by ‘getting’ it through the non-GUI functions rather than having an oddly placed for loop to set these values inside the spaceship and planet windows.

INHERITANCE:
We did had some minor setbacks with the FoodItem and MedicalItem classes. Initially, they were filled with switch statements, with FoodItem almost being 250 lines long. The switch statements were setting up values of food or medical items. The items themselves were referenced as a String, not an object of itself. If a player wanted a banana, the program would call feedCrewMember(banana) and sequentially traverse through the switch statements in the function to find the attributes of the banana. This method proved very inefficient and that we realised that we needed to consider better ways of implementing the methods, i.e. Inheritance (refer to UML diagram). It is was more convenient to create subclasses for six food items and three medical items since each have common variables.; name, price, health, hunger for food items, and cure for medical items. A subclass Banana would construct attributes through the FoodItem constructor, creating an object Banana of type FoodItem with its attributes simply obtained by the get methods implemented in the FoodItem class. We used inheritance for Planet since all 11 planets consisted of the same attributes; name, transporter pieces, money, a boolean ‘cleansed’ to check if a planet’s items have been obtained, and two ArrayLists of type FoodItem and MedicalItem. We created an ArrayList directly in the constructor of the planet subclasses using Arrays.asList.

COVERAGE:
We tested coverage for most of our main classes except for GameEnvironment, since this class merely derives methods through other classes. Our coverage score was 11.6% and this includes Crew, CrewMember, and FoodItem tests. We didn’t include classes like Spaceship and Planet, Outpost, and MedicalItem since their methods are mostly similar and linked with the three covered classes. We covered some getters and setters but left most of them out. Additionally. we didn’t include event classes such as AsteroidBelt, AlienPirates, SpacePlague as they are random events.

Euan
After hours of spending my nights peeling my eyes for this project, I have developed a form of respect to it. I appreciate what it was doing, as it is almost a glimpse of what is to come in the workplace; not in making a *Space Explorer* game, but in figuring out how to structure a program using the tools given to us from the lectures. Also, I discovered how important it is to manage and compartmentalise my time and workload, as I do have three other courses alongside SENG201. I believe that this was one of our setbacks with the project, since we did not complete any significant work during the holidays. I was in Wellington and my partner was in Invercargill, so the distance somewhat discouraged us to talk about the project. We only started doing serious work the week after the holidays and already I felt the pressure from my peers, who were already setting up a GUI whereas I was trying to figure out how Inheritance worked. All that time spent in boredom during the holidays I could have learned a thing or two about inheritance. Another fault of ours, was in taking too much time developing a command-line application. I spent hours trying to catch errors for all of the functions, i.e. If there were only two options, 0 and 1, how would I catch an error if a player clicked 2. I regret not spending more time discussing the structure of the classes and some of the style choices I have made, like having most actions from many classes go through the Crew class (as seen on the UML Diagram), even though it would have been simpler to call it directly from the GameEnvironment class. This particular setback made me realise and appreciate how GUI coding works. I was particularly surprised at how interesting it was implementing the application, as I had learned that html attributes (like </div>) worked very well on Java. I discovered that Swing appears differently on different operating systems, and after all the hard work spent on the 'look' and 'feel' of the game on the Macbook, seeing it look horrendous on the lab machines dispirited me a little bit. Fortunately, I figured out how to import fonts in Java through the likes of the internet and some friendly mates in the labs, and now the program looks somewhat nice and tidy. Overall, I spent rough 210 hours on this project, and for next projects, I would devote more time in exploring and considering the barebones of the application, before doing any coding.

Tim
This project was an interesting way to gain an insight into game design and the effort and behind the scenes work that goes into applications. At times it was difficult to comprehend the tasks and understand how inheritance would be used between classes. Re-watching lectures helped with some of this understanding in order to overcome these challenges. As this was our first time using Java, the project provided a great introduction to using this language along with creating a GUI. Creating our game screens was an exciting part of the project as our ideas came to fruition and we were able to see all our planning and work come together. It was an interesting and fun task as we were able to learn the different components and select the ones that best suited our plan and idea for our application. I am happy with our final product and believe it is enjoyable to play.

In retrospective, looking back at our project, having set goals for each person would have been more beneficial to ensure equal workload was distributed between the two of us. Having a large workload from my other courses, with tests, along with essays and an assignment due each week meant my partner ended up completing more work than me. This was my fault and should not have happened. For future projects I will ensure to plan out my weeks and set out times to work on the code, ensuring I do my equal share of the workload, as it is not fair on my partner.

We both went home for the term break, so using that time to complete research on functions within our code and how inheritance would work within our classes would have been helpful for when we arrived back at university. Instead we had to spend time figuring it out when we came back, while others were finishing their command line application. Limiting the time spent on the command line application attempting to fix errors and find problems would have also been beneficial. This task took up the first few weeks, putting us behind schedule. The positive from this was, spending a large amount of time on it allowed us to create our GUI without having to go back and fix or implement code into our command line application.

A big contributor to completing our project efficiently was our ability to plan. Each sunday we would meet up and discuss when we would be free the following week to work on the project. We also set our goals we wanted to achieve for that week. This allowed us to ensure we were not falling behind on tasks and had a clear idea about what was required for each member. Overall his project was a challenging and interesting task which broadened our knowledge of Java and creating game code. I believe our final product is aesthetically pleasing and is an enjoyable, fun game for the user. Overall, I spent roughly 130 hours on this project (Approximately 16 hours a week since the project was released).

Agreed Contribution
We agree that Euan completed 65% of the project, while Tim completed 35%.