Jamie Barnes 56505830
Seth Gunn 42032785

All of our program interacts with the GameEnvironment class because it is where we initialise all of our items, ships, islands and routes so it is the heart of our program. Our game environment is initialised through the Main class which also calls either the GUISetupScreen or the CommandLineUI. If you look at our UML Class diagram it is very easy to see how it all works together. The only inheritance we used with the project was in the custom illegal argument exceptions. Our Route class interacts the Island class as to initialise a Route, you need to pass it an Island. Our Item class interacts with both the Ship and Island classes as to initialise an Island, you need to pass it an Item and to make use of the cargo in Ship, it also needs to be passed an Item. RandomEvent is dependent on Ship because a RandomEvent cannot happen without a ship to act upon.

Our overall testing coverage is 33.9% however this includes the untested GUIcode. Our Main package has a coverage of 91%. We decided not to write tests for class Item, class Island or class Route, because the classes were all getters and setters so it was not worth our time to write tests for as either the getters and setters would work or they wouldn't, which would make itself known as soon as we decided to use the class. These classes are also tested by proxy with the tests we have written. We decided not to test the CommandLineUI, GUIGameOverScreen, GUIMainScreen and GUISetupScreen classes with JUnit as testing them by playing the game while bug testing was far easier and as far as we can tell, there is no real way of testing GUI elements with just code. The project did not need a lot of testing classes as most of the code was very basic getters and setters with only a few complicated parts in code segments that are not in the GUI classes. We needed to add setActiveIsland to GameEnvironment to make it testable, we also needed to change how randomNumber worked in RandomEvent to make it testable. The JUnit tests are all commented but we left them out of the Javadoc because we believe that having them in the documentation is not necessary.

In gameplay testing, there was a bug where the cargo was not reset when being robbed by the pirates, this was caused by the way that we were resetting the available cargo capacity. The available cargo space was being set to 0 when it should have been set to the max cargo capacity of the ship.

Thoughts & Feedback

The provided design architecture class ideas were very useful in setting up the skeleton for our own project and giving some direction on how to start coding the game.

In **2.2 Playing the Main Game** there are some specifications that overlap and do not make sense, an example of this would be Section 3, where it says that you should be able to see where each item was purchased and how much it cost. Doing this would be rather messy and potentially require another class, we also could not think of a way to make the display of a list of potentially 20 items, each with their own value and island, user friendly. The way we decided to implement it in our game where there was a default value, and an island would have a multiplier on its value and the player was only shown the base price when looking at their cargo.

The structure of classes that we were given in project specifications has a hint that mentions; "Functionality can be placed in its own package or class. Modularisation is the key, especially when you begin GUI programming". This was briefly touched on in the project specification, so we glanced over it when we were constructing our program.

Retrospective

Creating the UML class diagram did not go well as creating the UML use case diagram as most of the lecture and lab material on UML was focused on the use case diagrams.

The initial scope for the GUI was too grand to be done feasibly in the time given for the game which led to some inconsistencies in the GUI code which would make it harder to maintain in the future. Something to do in this regard for a future project would be to flesh out the GUI ideas sooner in the project so that there was some idea of what we would be making when the time came to code it. This would also help with keeping more maintainable code.

We could have put all our initialisers into their own classes and then their own packages which would have made our GameEnvironmet class a lot easier to both read and maintain as well as making it a lot shorter.

We probably could have used inheritance for our getter and setter classes like Item, Route, Island. We could have used inheritance for our GUI code.

Hours spent on Project & % Contribution:
Jaime: 70 55%
Seth: 65 45%