Oliver Neville: Obn11 84940154

Fergus Ross: Fgr28 74512447

Application Structure:

Reference UML diagrams included in zip file

The basic structure of our game separates out classes into display (UI/GUI) and functionality. The display classes query the GameEnvironement class, which keeps track of how the game variables update overtime, maintaining the state. The GameEnvironement then queries our other classes for structure, for the most part all variables that change over time are stored in GameEnvironement not in the individual classes. Some exceptions to this are variables such as a ship's stats, which do change over time with upgrades. Here it made more sense to track the variables in the ship object itself. Inheritance was used for Random Encounters to differentiate between the different types, as well as have unique methods for each one, such as adding a stealAmount for the pirate encounter, and having unique logic for fleeing or fighting the pirates. The guis were also opened by calling game environment each time.

Test coverage:

We ran out of time to do Junit tests. In hindsight we should have been doing them as we were implementing the code.

<u>Thoughts and Feedback on the project:</u>

Obn11: Overall quite satisfied with the final product, as can be expected could definitely use a few weeks to polish it though. Quite an intensive programing experience but the collaborative nature and focus on big scale structure minimised the time getting stuck on small things, and the premise was very fun to make. I had an excellent partner and we worked well together

Fgr28: Happy that we managed to finish and get a working product. We had many ideas on how to make it more fun and engaging but the time wasn't there for us. A kraken, for example, has

been something I wanted to implement from the start but was unable to. Partner was very helpful especially in areas where I am weak such as hashmaps. Gg would create again.

Retrospective:

The communication between partners went very well for this project, the planning that we did, and regular updates between us ensured that we were never on wildly different pages when it came to what work needed to be done by each of us week by week, how often we were available, the structure of the app and how the classes communicate, exactly how we want each method to function, what was working and what wasn't, and many more aspects of the project were very smooth because of this. Regular communication and proper planning from the beginning with things such as uml diagrams and weekly assessment of what's done/ needs to be done were the primary reasons for this.
Most failings occurred due to the amount of time each of us had to commit to the project, In particular due to other assignments we had to work on. This was especially detrimental because we ended up doing the bulk of our work asynchronously, and so could not collaborate as efficiently. General program skills were another issue, at times being due to our understanding of java and swing etc, other times general problem solving issues. These could come up unexpectedly and take many unanticipated hours to fix.
Next project some improvements to make is to get a clear roadmap of how much time each member can spend, and in particular better assessment during the overlaps. Using more industry standard practices to make our work more readable to each other would have gone a long way as well.

Obn11 hours spent: (Very roughly) 40-45hours
Fgr28 hours spent: 40-45 hours

Percentage contribution: Obn11: ~45-50%, Fgr28: ~55-50% Tick