Implementation Report

At the beginning of the development phase for Assessment 4, our team carried out rigorous testing to pinpoint any bugs not yet identified during the Selection phase. The software we inherited had been in good condition however there were still a few disparate issues. This coupled with the requested requirement changes from the client, which resulted in two new functional requirements **F19** and **F20 [1]**, prompted for modifications to the existing implementation of the game. Outlined below is a brief overview of the key alterations we have made to the software (code and GUI). How these changes affect the concrete architecture can be found in the Architecture Report **[2]**.

New type of crew member (F19)

The first change to the project specification required a new type of crew member to be added to the player's ship after achieving an objective, granting the ship a special ability. In order to fulfill this, we incorporated a system which gives random rewards to the player upon the defeat of an enemy college's boss. Though this can result in the player receiving a large quantity of gold, there is now a possibility for the arrival of a new crew member who increases different statistics (e.g. Speed, health, etc.) of the ship depending on their type.

To implement this, background code was added to the combat system which would upgrade the ship's statistics,followed by a message letting the player know which reward they had received and what effect it had on their ship. An advantage of this approach is that it minimises the impact on the existing software as we already had a system in place that would reward the player every time they defeat a boss.

New type of natural obstacle (F20)

The other part of the new requirements involves the addition of a new type of natural obstacle, which should appear randomly whilst sailing and can only be avoided or endured. Our interpretation of this requirement resulted in the implementation of a system which randomly spawns a variety of sea monsters that chase the player and damage their ship on collision. The player can either run away from the monster or stand still and endure it, perfectly aligning with the demand.

This change required implementation work deeper into the code, including the creation of multiple new classes (LabelTimer, GameUtils, AnimatedActor and SeaMonster) as well as some modification to the SailingScreen code. The LabelTimer class was created to allow the generation of damage labels which would then disappear after a defined time, and can also be further used to display temporary text on any screen whenever necessary. We used this feature to generate small texts indicating the player have lost their health whenever they collide with the monsters. The AnimatedActor class is a new base class created to simplify the generation and modification of any actor with animation as it gives us complete control over the animation currently active as well as storing a set of animations for every actors to be used when needed. This addition also introduced changes to some of our existing base classes as AnimatedActor now extends BaseActor and PhysicsActor now extends AnimatedActor instead of BaseActor.

To further support the creation of actors with animation, we also made the parseSpriteSheet method and put it inside a class called GameUtils. This method allows us to create animation from a single sprite sheet by treating it as a matrix, where we can then decide whether we want to use parts or all of it to make the animation by passing the elements we want to use as an array to the function. With the help of the above classes, we continued to make the SeaMonster class which extends PhysicsActor and uses parseSpriteSheet for its animations. It also contains the code for controlling movement of monsters as well as a timer system similar to that of the LabelTimer class to give control to the lifetime of each sea monster. Finally, adjustments were made to the SailingScreen code to support the generation and collision detection of the monsters. A new situation where the player run out of health just from collision with monsters was also account for through these adjustments.

Other Changes and Improvements

In addition to adding features to meet the new requirements, we also made some other miscellaneous changes and improvements to the game that we decided were necessary:

- We felt that the bright green and white buttons the UI used looked fairly out of place with the design of the rest of the game, so we changed the base colour to more of a brown/beige colour to better fit the pirate theme of the game.

- Further modifications to the GUI were mainly fixing alignment of actors on various screens to improve the look and not have ugly offsets, this was particularly noticeable in the department screen where the upgrades available were strangely laid out.

- To meet Requirement **F15**, we implemented a Save/Load feature into the game. This meant making some objects implement the Serializable interface, allowing for the serialisation of game objects into byte code using a package from apache commons called SerializationUtils. We then encoded this bytecode into a string to be stored in a file using libGDX's Preferences class, which works similarly to a HashMap.

- We found that in the version of the game inherited from Limewire there were some issues with the collision detection system in the minigame. In particular, the player sometimes got stopped in the middle of an open path while other times can go through the walls. After a thorough inspection of the existing code, we noticed Limewire had treated the whole minigame as a matrix and implemented everything from collision to enemy AI based on interaction with nodes within the matrix. Ditching this system meant that we would have to rewrite the whole minigame from scratch which was not ideal so we decided to keep the existing implementation but redesign player's movement and collision detection between the player and the walls. A similar approach to collision detection in the SailingScreen was used, boundaries were drawn onto the tiled map and checked for overlaps within MiniGameScreen, keeping everything separated from the matrix approach.

- We also noticed that Limewire had not implemented any game objective even though it was clearly stated in requirements **F2** and **F5**. To accommodate for this, we decided that the game should end after the player has defeated all the enemy college's bosses as this should allow the game to be completed within a reasonable amount of time. A new class WinScreen has been added to achieve this, showing a congratulation message alongside the number of points acquired during the playthrough after the player defeat the last enemy boss.

- Various tweaks and balance changes were also made to support requirement **NF1** (The game should be easy to pick up and playing should feel pleasing and satisfying). These ranges from adjusting movement speed on SailingScreen to strength of enemies and bosses.

References

[1] Rear Admirals Assessment 4 Requirements, 2019. [Online]. Available: https://therandomnessguy.github.io/SEPR/Assessment/4/Updates/Req4.pdf [Accessed 30 - April - 2019]

[2] Rear Admirals Assessment 4 Architecture, 2019. [Online]. Available: https://therandomnessguy.github.io/SEPR/Assessment/4/Updates/Arch4.pdf [Accessed 30 - April - 2019]