Change Report

Team 21
Zain Alshaikh
Corin Bertrand
Damian Heaton
Mandy Li
Brandon Oliver
Toms Tafijs

Summary: Changes to management

The results of the previous assessment revealed areas where we needed to improve. Overall, we were satisfied with how we organized our documentation, so we continued with the same approach. Each section of the documentation was assigned to one person who was responsible for creating a draft. Furthermore, to prevent inconsistencies between the documentation and coding, we reviewed these as a group.

For the coding, we assigned two people to work on implementing the new features. We were conscious about the communication between the documentation and coding teams as this was a weak point in the first assessment. The coding and testing teams worked closely together as well as maintaining frequent communication with the requirements person to ensure everything was consistent.

We also considered coding style and inconsistencies in that line of thought. As there were only two people focusing on the code, we thought it would be easy to make modifications as we went along to maintain a consistent style. At the end, we as a group, reviewed all the code, ensured it was documented correctly, and commented appropriately with the new modifications.

Change report:

I. **Requirements:** https://lyrenhex.github.io/ENG1-Project-Part2/pdfs/Req2.pdf

Firstly, the existing introduction had all the necessary explanations for the requirements elicitation, but we modified it to be more concise and used bullet points to make it more readable. Furthermore, we added references and clarified the standard we used to define requirements as this was not made clear.

We agreed that our requirements elicitation during the first assessment was effective so we continued with the same approach. Firstly, as a group, we looked at the existing requirements and removed those that we thought were not needed, and relocated others that were in the incorrect table. Then, we went through the explicit requirements listed in the brief and noted the new features that had to be implemented. We also looked at the structure of the existing code base and to see how we could incorporate our decisions into it.

The second modification we made to the requirements was the priority system which now uses H, M, L (High, Medium, Low) rather than the (Shall, Should, May) system, as we thought this would be more intuitive.

II. **Abstract and concrete architecture:** https://lyrenhex.github.io/ENG1-Project-Part2/pdfs/Arch2.pdf

Our approach here was focused on assimilating new elements into the existing system whilst avoiding any major changes to the structure. We will describe the modifications made and justify our design choice.

The architecture document includes the new UML diagrams that contain the updates we made. Most notably, there are two new packages for implementing saves and obstacles.

Abstract Architecture

The changes to the abstract architecture are reflected in the updated package diagram, which is seen in the first picture, and is also linked on the website. When sketching this design, we were able to easily identify which abstract classes the new features would come under. For example, we added a package for Obstacles. Since PhysicsObject has a method for handling collision, it made sense for classes in Obstacles to inherit from it. Classes in this package implement features such as floating ducks, the Longboi duck, and weather.

Overall, we identified with the object oriented approach which worked well for continuing this project so we only added those new classes and kept everything else the same.

Concrete Architecture

For the concrete architecture, we made some changes to add new attributes and modify existing methods. Overall, the class inheritance structure was very efficient for adding new classes so we did not make any big changes.

The second picture shows a section of the concrete architecture, specifically the inheritance of Boat from PhysicsObject. We modified this image to include the new classes in the Boat package, which were CollegeBoat and Blessing.

The following section was extended to include the new classes and a brief explanation of the attributes and methods. These were CollegeBoat, Blessing, Longboi, and the classes in the Saves and Obstacle package.

In the last section, we added justifications for the new requirements relevant to Assessment 2. Finally, we updated the website with new images for the updated architecture and this is linked appropriately within the document.

Justification for changes:

Save functionality (UR_SAVE_GAME)
The Saves package is used to capture the game state. It consists of classes such as BlessingState, PlayerBoatState, etc which saves information about various aspects of the game. These are used in the class SaveState, which when instantiated, creates an overall save state of the whole game.

The saving is triggered when Esc is pressed whilst the menu is open. It is implemented this way so that the game saves when the game is exited. These changes are reflected in the Menu class.

Load functionality (UR_LOAD_GAME)
We modified the render() method in eng1game to search for a local save file and load it if existed. The only weakness of this implementation as opposed to a manual load button is if the player wanted to start a new game and a save file existed, it is automatically loaded. Thus, they would have to delete or relocate the file. However, we decided that this was not a priority to change.

Enemy combat (UR_SHIP_COMBAT)
We added a new class CollegeBoat, which has a EnemyCollege attribute to show which college it belongs to. We updated the onShoot() method to define behaviour for attacking the player.

Longboi event (UR_MAGICAL_POWERUPS)
The Longboi class was added to implement the Longboi event. This is an obstacle, which inherits from College, as this class behaves similarly to how we wanted Longboi to (it has high health and methods for attacking the player). When destroyed, it modifies the ProjectileData (to ducks) and increases the value of the projectileDamageMultiplier attribute.

Obstacle (UR_OBSTACLES)
We added a new Obstacle class for implementing objects such as rocks and shipwrecks. They inherit from PhysicsObject - however, the collisions are handled by the playerBoat object since it affects its HP attribute.

Blessing obstacle (UR_MAGICAL_POWERUPS)

We added a new Blessing class which has similar functionality to a Boat and grants an Immune powerup when the player collides with it. This works by calling the setImmune() method which modifies the isImmune and timeImmune attributes of the player boat. We then modified the onCollision() method in playerBoat so that it would not take damage whilst immune.

Weather (UR_WEATHER)
We created a new abstract class for Weather, which has a position and texture defined. Furthermore, we have a Storm class which inherits from Weather. When the playerBoat is within the grid on the map that represents bad weather, it decreases the HP attribute of the playerBoat.

We also added a new class ChoppyWaves which has a position defined. When collided with, this inverts the player controls e.g. A normally turns left, so it would now turn right. Upon collision, the invertedControl attribute of playerBoat is set to True.

New shop menu (UR_SHOP, UR_SHOP_POWERUPS)
To implement the shop upgrades, we added a shop menu with its functionality defined in HUD. When triggered (via keypress), it calls the new method InitialiseMenu(). We chose this as opposed to a physical shop to prevent the game from taking too long (UR_WIN_TIME). After players spend gold on a powerup, this updates the relevant attribute in playerBoat, e.g. a speed boost modifies the speed attribute. There is also a new method RandomiseUpgrades() so that the shop would contain two random powerup choices from the Upgrade enum. Furthermore, the HUD class was modified to include new textures for the shop menu and upgrades.

In total, there are 5 possible powerups that can be obtained throughout the game, which is inline with the requirements.

Added difficulty feature (UR_LEVELS)
There is a Difficulty enum containing Easy, Normal, Hard, and the player is given the option to select the level when the game starts by an appropriate keypress button. The setDifficulty() method in eng1game was updated to modify the difficulty by using the method defined in GameController. Specifically, it modifies the maxHP, projectileDamageMultiplier and defense attributes in playerBoat to reflect this change.

Changes to map and GameController
The original map was small with enemy colleges spawning on each corner of the screen. To improve the aesthetic, we updated the GameController so that it was possible for movement outside of the screen, using a moving camera. Furthermore, we modified it to spawn the new obstacles described above such as ChoppyWaves, Blessing, LongBoi, rocks and shipwrecks, whilst taking into account the larger map size. We kept the position of the playerCollege the same - it would always spawn on the bottom left corner, which was intuitive for the player.

III. Methods and plans: https://lyrenhex.github.io/ENG1-Project-Part2/pdfs/Plan2.pdf

https://lyrenhex.github.io/ENG1-Project-Part2/pdfs/Plan1.pdf

Our team used the scrum agile method to approach this part of the project which is the approach the first team used. Therefore, there aren't many changes when it comes to the method selection. However, a few things that changed, such as:
-The number of key stages.
- Continuous integration as well as a proper testing stage were added.
The approach we took to each stage is very similar as well, just like the first team we implemented the documentation stage while the implementation was being written. One major difference was the approach to testing which we did alongside the implementation as well. In order to make catching the errors easier as well as to make sure every stage of the game was working as it should.
Similar applications were used in the project as we also used the LibGDX library and GitHub as the main collaboration tool.
The team approach we took was very different. The other team's strategy was to split their team into two sub teams. We, however, did not do this, we split different sections between the six of us and appointed a head and shadow head to each one of the sections. This approach ensured that:

- Everyone had an equal amount of work.
- Every part of the project had at least two people who could focus their attention on it.
- The team could supervise each other and make sure that the others are also completing their tasks.

The approach taken to this project is very different from the first teams. This was mostly due to the fact that the two assessments were very different and had different stages of planning and implementation that had to be completed. Some parts of the plan were similar though, as we also used a Gantt chart to track our progress weekly.

Gantt charts: https://lyrenhex.github.io/ENG1-Project-Part2/v2/snapshots

IV. Risk assessment and mitigation:

https://lyrenhex.github.io/ENG1-Project-Part2/pdfs/Risk2.pdf

Their risk document was the bit that we made moderate changes to as it was overall pretty solid. The main change made was rewriting and structuring the introduction as we felt that it was rather lacking in its explanations of how did we manage the risks, how did we identify risk, how did we analyse risk and how did we track risk as well as structuring it out into sections to make it easier to read and digest. Other than that the document was kept the same as the table itself didn't need to be changed due to the risks all still being applicable to the second half of the project. This is a document that no matter who is making expansions of the game, there shouldn't be any changes unless something is changed majorly in the management of the project. Since we aren't meant to consider things like bugs or the product itself not functioning as risks, and the conditions of the project are different, then nothing had to be changed.