Engineering 1

Group Assessment 2

Change Report

Cohort 2

Team 11

Change Management

First of all, we went through documentation provided by Team 12 in order to get familiar with their approaches and methods, as well as to identify both unimplemented and new features that must be added to the game. Then, we divided tasks between all members of the team to let everyone contribute to the project. Each member of the team was primarily responsible for one deliverable which they made changes to.

Every week we had a team meeting where we discussed what had been changed or added and what was yet to be done, as well as assigned new tasks. In our own Discord server, members of the team could ask questions or discuss new changes anytime. It allowed members to have a clear understanding of all changes and decisions, and to have an opportunity to express their opinion about a certain change.

To keep track of all changes and make sure that all of them are necessary and justified, we created a change log in our GitHub repository to record all significant changes along with the date when they were made and their details. It enabled the team to react to new changes quickly, for example, to identify the cause of an error in the code or approve/disapprove a change. Additionally, GitHub allows you to view the commit history of any branch, as well as get detailed information about each commit, which is particularly helpful for keeping track of all changes and actions of contributors.

Furthermore, we continued to use Google Drive and some of the office tools that come with it, namely Google Docs and Google Sheets. One of the main advantages of Google tools is the ability to view edit history of a document and restore older versions of it, which is especially useful for change management.

Changes made to Assessment 1 deliverables

Requirements:

Once the new requirements for Assessment 2 were released, we analyzed them and defined any associated risks and possible alternatives. We added three new user requirements to Requirements document, namely UR_LEVELS, UR_SAVE and UR_POWER_UP. All of them were set by the customer and listed on VLE. Additionally, we defined new Functional requirements and added them to the spreadsheet created by Team 12.

UR_LEVELS refers to the new requirement that the game must allow players to choose between different levels of difficulty: Easy, Normal and Hard. These were the difficulty levels listed as an example on the VLE, and are sufficient for this program.

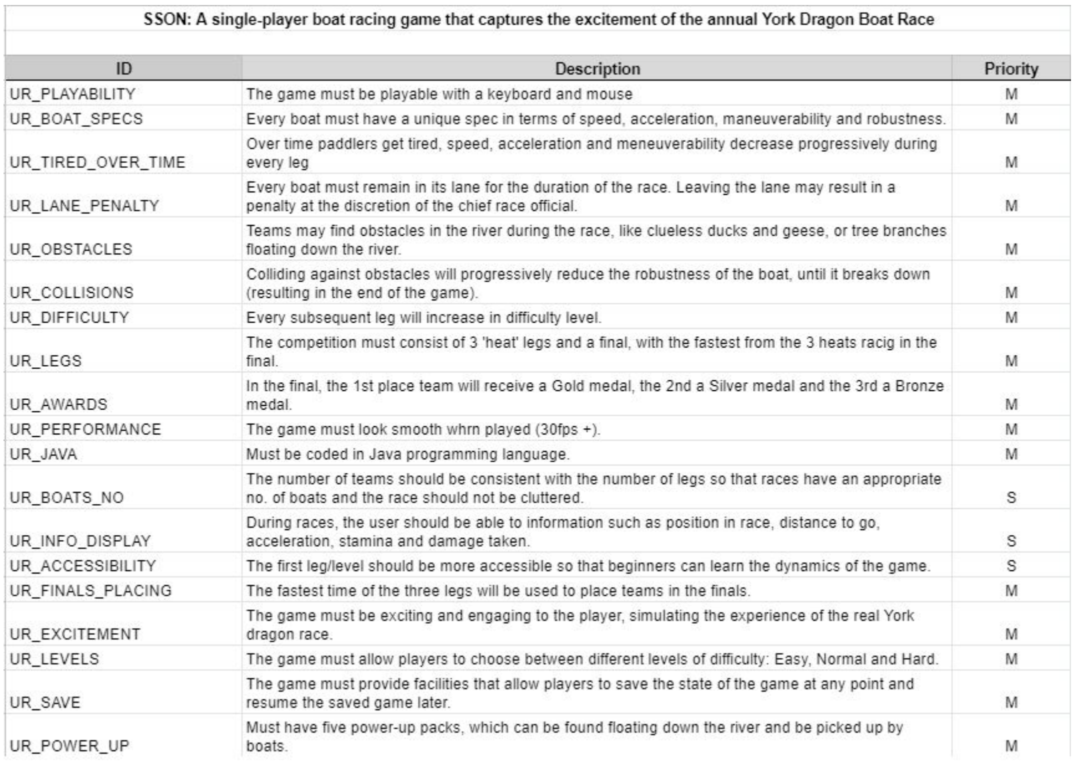
UR_SAVE denotes the requirement to provide facilities that would allow players to save the state of the game at any point and resume the saved game later.

Lastly, UR_POWER_UP is a new requirement to implement five power-up packs, which could be found floating down the river and be picked up by boats to improve some of their characteristics.

The URL of these requirements is:
https://eng1-team-11.github.io/ENG1-Team-12/Documentation2/Req1.pdf, page 4

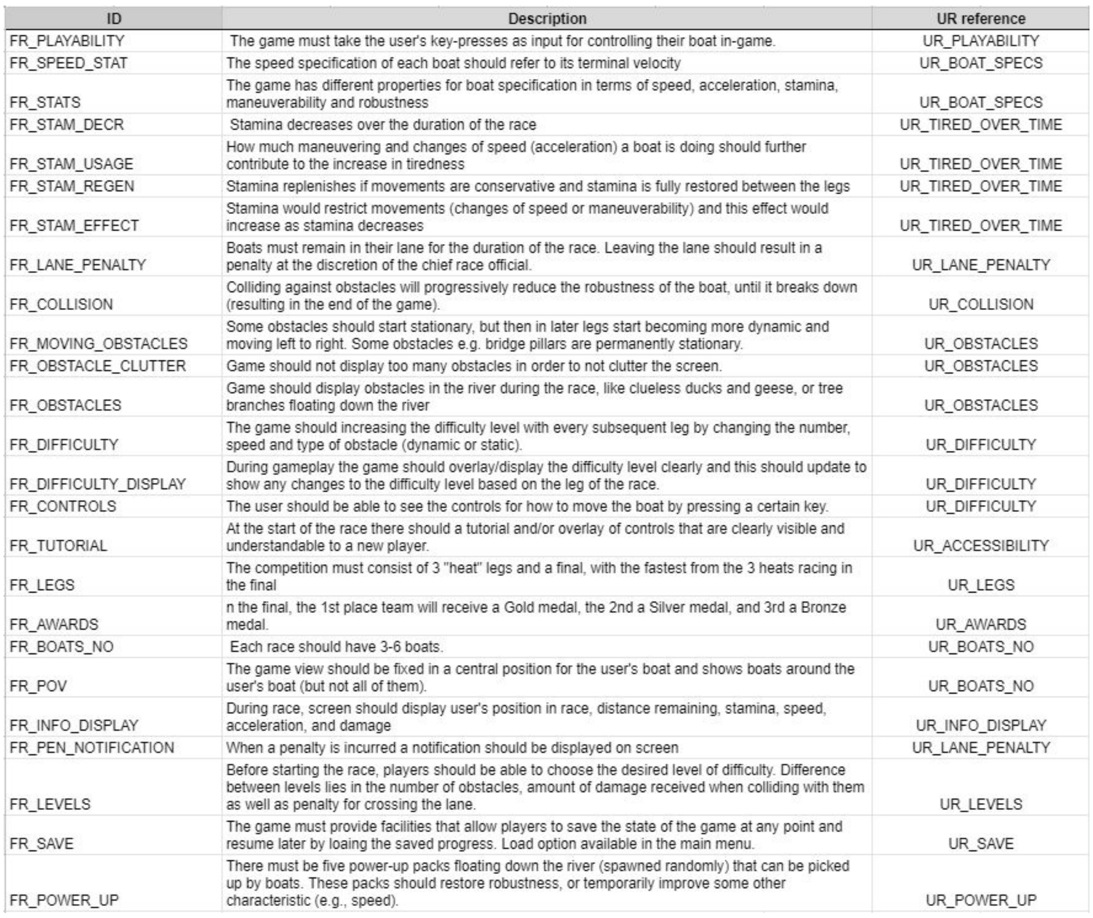
They can also be found in a table at:
https://eng1-team-11.github.io/ENG1-Team-12/Documentation2/ReqTable.pdf

All of the new requirements were set and defined by the customer, meaning that they are necessary for the game to be complete and successful.



SSON: A single-player boat racing game that captures the excitement of the annual York Dragon Boat Race		
ID	Description	Priority
UR_PLAYABILITY	The game must be playable with a keyboard and mouse	M
UR_BOAT_SPECS	Every boat must have a unique spec in terms of speed, acceleration, maneuverability and robustness.	M
UR_TIRED_OVER_TIME	Over time paddlers get tired, speed, acceleration and meneuverability decrease progressively during every leg	M
UR_LANE_PENALTY	Every boat must remain in its lane for the duration of the race. Leaving the lane may result in a penalty at the discretion of the chief race official.	M
UR_OBSTACLES	Teams may find obstacles in the river during the race, like clueless ducks and geese, or tree branches floating down the river.	M
UR_COLLISIONS	Colliding against obstacles will progressively reduce the robustness of the boat, until it breaks down (resulting in the end of the game).	M
UR_DIFFICULTY	Every subsequent leg will increase in difficulty level.	M
UR_LEGS	The competition must consist of 3 'heat' legs and a final, with the fastest from the 3 heats racig in the final.	M
UR_AWARDS	In the final, the 1st place team will receive a Gold medal, the 2nd a Silver medal and the 3rd a Bronze medal.	M
UR_PERFORMANCE	The game must look smooth whrn played (30fps +).	M
UR_JAVA	Must be coded in Java programming language.	M
UR_BOATS_NO	The number of teams should be consistent with the number of legs so that races have an appropriate no. of boats and the race should not be cluttered.	S
UR_INFO_DISPLAY	During races, the user should be able to information such as position in race, distance to go, acceleration, stamina and damage taken.	S
UR_ACCESSIBILITY	The first leg/level should be more accessible so that beginners can learn the dynamics of the game.	S
UR_FINALS_PLACING	The fastest time of the three legs will be used to place teams in the finals.	M
UR_EXCITEMENT	The game must be exciting and engaging to the player, simulating the experience of the real York dragon race.	M
UR_LEVELS	The game must allow players to choose between different levels of difficulty: Easy, Normal and Hard.	M
UR_SAVE	The game must provide facilities that allow players to save the state of the game at any point and resume the saved game later.	M
UR_POWER_UP	Must have five power-up packs, which can be found floating down the river and be picked up by boats.	M

Updated User Requirements Table



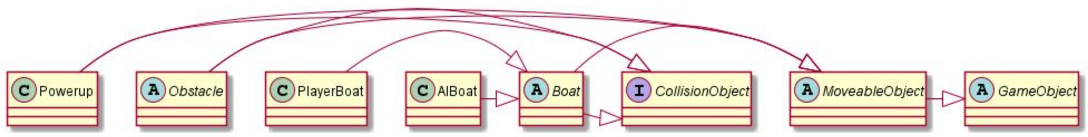
ID	Description	UR reference
FR_PLAYABILITY	The game must take the user's key-presses as input for controlling their boat in-game.	UR_PLAYABILITY
FR_SPEED_STAT	The speed specification of each boat should refer to its terminal velocity	UR_BOAT_SPECS
FR_STATS	The game has different properties for boat specification in terms of speed, acceleration, stamina, maneuverability and robustness	UR_BOAT_SPECS
FR_STAM_DECR	Stamina decreases over the duration of the race	UR_TIRED_OVER_TIME
FR_STAM_USAGE	How much maneuvering and changes of speed (acceleration) a boat is doing should further contribute to the increase in tiredness	UR_TIRED_OVER_TIME
FR_STAM_REGEN	Stamina replenishes if movements are conservative and stamina is fully restored between the legs	UR_TIRED_OVER_TIME
FR_STAM_EFFECT	Stamina would restrict movements (changes of speed or maneuverability) and this effect would increase as stamina decreases	UR_TIRED_OVER_TIME
FR_LANE_PENALTY	Boats must remain in their lane for the duration of the race. Leaving the lane should result in a penalty at the discretion of the chief race official.	UR_LANE_PENALTY
FR_COLLISION	Colliding against obstacles will progressively reduce the robustness of the boat, until it breaks down (resulting in the end of the game).	UR_COLLISION
FR_MOVING_OBSTACLES	Some obstacles should start stationary, but then in later legs start becoming more dynamic and moving left to right. Some obstacles e.g. bridge pillars are permanently stationary.	UR_OBSTACLES
FR_OBSTACLE_CLUTTER	Game should not display too many obstacles in order to not clutter the screen.	UR_OBSTACLES
FR_OBSTACLES	Game should display obstacles in the river during the race, like clueless ducks and geese, or tree branches floating down the river	UR_OBSTACLES
FR_DIFFICULTY	The game should increasing the difficulty level with every subsequent leg by changing the number, speed and type of obstacle (dynamic or static).	UR_DIFFICULTY
FR_DIFFICULTY_DISPLAY	During gameplay the game should overlay/display the difficulty level clearly and this should update to show any changes to the difficulty level based on the leg of the race.	UR_DIFFICULTY
FR_CONTROLS	The user should be able to see the controls for how to move the boat by pressing a certain key.	UR_DIFFICULTY
FR_TUTORIAL	At the start of the race there should a tutorial and/or overlay of controls that are clearly visible and understandable to a new player.	UR_ACCESSIBILITY
FR_LEGS	The competition must consist of 3 "heat" legs and a final, with the fastest from the 3 heats racing in the final	UR_LEGS
FR_AWARDS	n the final, the 1st place team will receive a Gold medal, the 2nd a Silver medal, and 3rd a Bronze medal.	UR_AWARDS
FR_BOATS_NO	Each race should have 3-6 boats.	UR_BOATS_NO
FR_POV	The game view should be fixed in a central position for the user's boat and shows boats around the user's boat (but not all of them).	UR_BOATS_NO
FR_INFO_DISPLAY	During race, screen should display user's position in race, distance remaining, stamina, speed, acceleration, and damage	UR_INFO_DISPLAY
FR_PEN_NOTIFICATION	When a penalty is incurred a notification should be displayed on screen	UR_LANE_PENALTY
FR_LEVELS	Before starting the race, players should be able to choose the desired level of difficulty. Difference between levels lies in the number of obstacles, amount of damage received when colliding with them as well as penalty for crossing the lane.	UR_LEVELS
FR_SAVE	The game must provide facilities that allow players to save the state of the game at any point and resume later by loaing the saved progress. Load option available in the main menu.	UR_SAVE
FR_POWER_UP	There must be five power-up packs floating down the river (spawned randomly) that can be picked up by boats. These packs should restore robustness, or temporarily improve some other characteristic (e.g., speed).	UR_POWER_UP

Updated Functional Requirements Table

Architecture:

Abstract architecture

We decided to add three new entities to the abstract architecture, most notably class *Powerup* which represents the power-ups that float down the river and can be picked up by boats. *Powerup* inherits from the abstract class *MoveableObject*, since it is supposed to be moving along the lane and is similar to the *Obstacle* class. Like *Obstacle*, it also implements the *CollisionObject* interface, although certain functions within the Interface were changed such that any object colliding with some other object, *other*, will know the type of *other* and can react accordingly (ie. Power-ups won't damage the boat, obstacles will). Classes *SaveManager* and *Difficulty* were also added.



Concrete architecture

The concrete architecture itself has not changed much as an outline of classes, although actual implementation details differ significantly. Much of the superfluous and unnecessary code has been removed or re-implemented in a cleaner or more efficient way.
Major changes to the code include:

1. The addition of the *Powerup* class which behaves similarly to the *Obstacle* class but grants bonuses to the boats rather than penalising them. Required for UR_POWER_UP.
2. All *CollisionObject* instances now have a "value" that the AI uses to decide whether it should steer away from it or towards it, which allows the AI to play more competitively.
3. The original artificial intelligence has been stripped out and replaced with one which is much more competitive with the player, and far better optimised. As beating these constitutes 90% of the gameplay, this helps with UR_LEVELS, UR_EXCITEMENT, and UR_PERFORMANCE.
4. The *Difficulty* class has been added, which is implemented as a Singleton and tracks the global difficulty setting. This is accessed by many classes and allows changes to difficulty to be made quickly and centrally. Required for UR_LEVELS
5. A more robust UI system has been implemented, including *Button*s, *Switch*es, *Scene*s, *Label*s, and *Image*s. This represents a significant improvement over the original approach which was to manually define these in game states themselves.
6. Many classes have had variables and functions removed where they have been unnecessary, primarily in *Scene** classes and in *BoatRace* where a lot of sprites were stored, and references to single objects allow for the removal of checks against entire lists to find one instance of a class.
7. The project itself has been updated to Java 8 to take advantage of new (2014) language features for quicker and cleaner development.

8. The project has been refactored from snake_case to camelCase for fields and variables as this is Java convention and less confusing.
9. Large if-statements connected by && have been removed to improve readability as well as performance in some cases
10. Almost every class field has been moved from the protected to private access level to prevent accidental change of values by external functions that should not be able to access these.
11. The entire game now calculates the time between frames and uses this for movement-related calculations rather than assuming the game runs at 60 frames per second, ensuring consistent movement even if the game is performing poorly. Whilst it doesn't directly improve performance, it should make the game feel smoother if it does perform poorly, somewhat helping mitigate if UR_PERFORMANCE is not met.
12. All non-inner classes have been made public so that they can be used in unit testing, and because exposing classes is generally a lot safer than exposing fields.
13. Collisions have been reworked to use a quadtree for storage of objects and lookup rather than originally as a list, which has served to make some critical parts of the code up to 100x less CPU time intensive. This has almost completely eliminated anything blocking UR_PERFORMANCE.

Methods and Plans:

Method and Tool Selection

Both our team and Team 12 have opted to follow an agile development methodology, specifically Scrum. The Scrum framework is designed for smaller teams, making it perfect for us and our project. In Scrum, goals are broken down into small tasks which are completed within a set amount of time known as a sprint. We decided to have small week-long sprints as we had official supervised practical sessions once every week.

Most of the tools used by Team 12 coincided with our choice. Just like Team 12 we used the following tools for completing the tasks in our project:

- Google Drive for storing documents and other files
- Discord for communication
- GitHub for source control
- Google Docs for writing up documents

Additionally, we used PlantUML for creating class diagrams and Javadoc for generating HTML documentation of the source code.

For task management we decided to use GitHub Projects. It is a kanban board-style to-do list allowing for the creation of boards and cards with tasks, which all developers can access and modify. This was ideal for our chosen development methodology as it allows for tasks to be broken down into smaller parts and put on a board where anyone can pick and develop them, then move to the “completed” board. We favoured GitHub Pages over Trello, which was used by Team 12, as we didn’t need any additional features provided by Trello (e.g. colour coding and labelling) and having everything in one place was more convenient for us.

In addition, we supplemented the Method Selection and Planning document with additional information such as information about the selected tools, justification, and possible alternatives to some tools.

Team Management Approach and Plan

We decided not to change the plan and team management approach written by Team 12. Instead, we added our own approach and plan as an extension.

Our approaches are quite similar due to the fact that both our teams used Scrum development framework along with Discord as the main way of communication. Unlike Team 12, who split their team into sub-teams for tackling different parts of the assignment, we divided tasks between individual members of the team. In the approach of Team 12, each task had a leader as well as one or many collaborators which ensured good implementation and involved frequent communication. In our approach, each member of the team was primarily responsible for one task such as writing and updating one of the deliverables or implementing a specific feature of the game. One of the advantages of our method is that everyone had to focus on one or few specific tasks, which allowed for more thorough and deep research and better understanding of each task.

For Assessment 2 we created a new Gantt chart to keep all things separate and focus on the new tasks. We decided to keep Gantt Chart created by Team 12 and supplemented the document with our own Gantt Chart (can be found on our website: https://eng1-team-11.github.io/ENG1-Team-12/Documentation2/Gantt.pdf) as well as screenshots evidencing our work on the assignment.

We deleted some of the content provided by Team 12, namely sections ‘Key Event List’ and ‘Announcements’, as it overran the page limit allowed for Method Selections and Planning deliverable and went beyond what was required.

Risk Assessment and Mitigation:

Our Approach and Presentation

Coincidentally, Team 12 had a similar approach of assessing and presenting risks. We both used Low/Moderate/High scale to assess the likelihood and severity of possible risks, as well as tabular format to present them. This system is clear and easy to understand, which makes it perfect for small projects like ours.

Team 12 and our team both divided risks into three categories: “Project”, “Product” and “Business”. “Project” category was assigned to the risks that might affect our project schedule and progressing speed. “Product” category is for the risks that may potentially affect the quality of the game, for example, any hardware or software issues. “Business” category was assigned to the risks that may affect the success of our product in the market, in our case - success of using the game for promotional activities organized by The University of York Communications Office.

In order to identify the risks associated with the new requirements for Assessment 2, we first reviewed Assessment 1 requirements, specifically those that had not been implemented. Then, we went through our project plan to determine all possible factors that can affect the quality of the final product.

Furthermore, we chose a different format of presenting IDs of the risks, since the old format, used by Team 12, took up too much space and yet was not informative, making it inconvenient for us. We also removed the ‘Backup Owner’ column, since having one risk owner was enough for our team.

Risks and Mitigation

We decided to keep all the risks identified by Team 12 as they all were possible and coincided with the risks that had been found by our team. Additionally, we supplemented the table with other risks that we identified while working on Assessment 1&2 as well as after reviewing the new requirements for Assessment 2 (R10-R21)

The entire table can be found at: https://eng1-team-11.github.io/ENG1-Team-12/Documentation2/Risk1.pdf, with R10 starting at page 4

ID	Type	Description	Likelihood	Severity	Mitigation	Owner
R10	Project	Miscommunications and lack of clarity, e.g. several members doing the same task, misunderstanding design decisions	M	M	Every team member must inform others about the parts of the project they are working on, ask for clarification when uncertain about something	All
R11	Project	Team is running out of time	H	H	Meet up more frequently and make sure every team member is involved	All
R12	Project	Conflicts within the team	L	M	Try to resolve a conflict or minimize its effect on the projects	All
R13	Project	Misunderstanding some of the assessment questions	L	H	Contact the lecturer and make sure all question are understood and answered	All
R14	Business	The customer is not satisfied with the game design/implementation	L	H	Set up a team meeting and discuss possible	All

R15	Product	Product doesn’t meet some user, functional or nonfunctional requirements	M	H	Review the requirements at each stage of the project and make sure all of them are met	All
R16	Product/ Business	Product doesn’t meet the customer expectations, e.g game is too complex, too simple or not enjoyable	L	H	Organize a team meeting and discuss possible changes or improvements	All
R17	Project	Lack of skills/ knowledge of team members	M	H	Team members must learn/improve their skills through taking online courses, reading relevant literature or using any other resources	All
R18	Product	Game is not playable on a low spec computer (i.e. dual-core laptop with 4GB of RAM)	L	M	Set up a team meeting and discuss what adjustments can be made to the game to run on a low spec machine	Coders
R19	Product	Game logic does not work as expected, e.g. durability doesn’t reduce when colliding with obstacles, difficulty does not increase/decrease when changing the level of difficulty etc.	L	H	Members responsible for the implementation must go through the code and find any logic errors	Coders
R20	Product/ Project	Misunderstanding the new requirements or some of the questions of Assessment 2	L	H	Contact the lecturer and make sure all question are understood and answered, and new requirements are clear	All
R21	Product	New requirements are not implemented for some reason	L	H	Review the requirements at each stage of the project and make sure all of them are met. Make sure someone is working on each new feature of the game.	All