Requirements

<u>Assessment 1:</u>
<u>Team 14: Bass2</u>

Katie Maison
Saud Kidwai
Jacob Poulton
Cody Spinks
Felix Rizzo French
Joachim Jones

<u>Assessment 2:</u>
<u>Team 6: Team siKz</u>

Ryan Bulman
Frederick Clarke
Jack Ellis
Yuhao Hu
Tom Nicholson
James Pursglove

2.(a)
To update the requirements, we first evaluated the original methodology used by the first group. We decided that it was a suitable system and could keep the framework intact. Following this, we assessed the features needed for assessment 2 and added them following this framework. The original methodology is as follows, along with the process used to provide context. Any changes are noted in the relevant report:

To start the **requirements elicitation process**, we researched options and decided on the following:

- First we carried out a document analysis of the product brief. [1]
- Then had a meeting with the customer who is interested in trying to market and sell the game.
    - This was a semi-structured interview in which a combination of predefined and unplanned questions were asked [1].
    - It was useful to clarify some questions the team had about the product brief and ask if any additional requirements would be necessary.
- The next stage was to do a use case [WB.1] which is the scenario in which an actor interacts successfully with the system, in this case, successfully plays through the game.[2]
    - From this, we have picked out the requirements necessary to make that scenario successful.

We have ensured that each requirement possesses the characteristics laid out in the guidelines laid out in the additional guidance for requirement-related activities in ISO/IEC/IEEE 12207 and ISO/IEC/IEEE 15288 [3]. These characteristics are that each requirement should be: **Necessary, Appropriate, Unambiguous, Complete, Singular, Feasible, Verifiable, Correct, Conforming**. [3]

To make sure the requirements themselves are well worded, we will use the language criteria in the guidance. This involves not using unbounded or ambiguous terms such as superlatives, vague pronouns, and subjective language.

To present the requirements we have used a table consisting of:

- ID
- Requirement description,
- Priority for the requirement,
- Difficulty to implement

And added additional information afterwards for those with:

- Environmental assumptions made
- Risks
- Alternatives

In the priority column, we decided:

- High = required to reach a minimum viable product
- Medium = not strictly required, but greatly improve the products quality
- Low = superfluous features that are just "*nice to have*"

This is so that when implementing the requirements, we can decide which features are integral to meeting the brief and focus on the ones with higher priority.

The difficulty column follows a self-explanatory system of high to low which shows the initial, assumed, level of difficulty of each requirement to implement. This allows us to plan more effectively regarding which requirements should be tackled first.

SSON

The brief requires us to create a game that allows users to roam around a body of water and engage in combat with enemy colleges and ships in order to attempt to take them over all while avoiding a number of obstacles (e.g. Lake Monsters, Bad weather etc.).

2.(b)

Risks and assumptions write up

Many of our technical and user requirements have associated risks in their implementation. For example some of the graphical requirements have both technical and political risks since there is an inherent risk of graphical glitches with implementation but there also may be disagreements about how certain graphics should look. Furthermore, the implementation as a whole has a number of risks related to feasibility, inner team politics, schedule and software.[2]

User Requirements

ID	Requirement	Priority	Difficulty
UR.START_SCRN	User can start the game via a start screen	Low	Low
UR.SCRN_NAME	User can choose a screen name	Low	Low
UR.SEE_POS	The user must be able to see their sprite on the lake	High	Low
UR.TUTORIAL	The user must be able to see a tutorial	Medium	Medium
UR.CLG_POS	User must be able to see where the colleges are relative to them	Medium	Medium
UR.UPDATE_POS	The user must be able to move the ship on the lake	High	Medium
UR.COLLECT_PNTS	The user must be able to collect points	High	Medium
UR.ATK_CLG	The user must be able to attack colleges	High	Medium/High
UR.COLLECT_LOOT	The user must be able to collect loot	High	Low
UR.VIEW_PNTS	User must be able to view their points/score	High	Low
UR.VIEW_LOOT	User must be able to view their loot	Low	Low
UR.CPTR_CLG	The user must be able to capture colleges	High	High
UR.SEE_TASKS	The user must be able to see tasks to complete	Medium	Low
UR.RESTART_GAME	The user must be able to restart the game at any time	Medium	Low
UR.FINISH_GAME	The user must be able to finish the game by returning ‘home’	Low	Medium
UR.LOSE_GAME	The user must be able to lose/die	Low	Medium
UR.AVOID.OBS	The user must have obstacles in the world to avoid	High	Medium
UR.SPEND_LOOT	The user must be able to spend the gold they earn	High	Medium
UR.FIGHT_SHIPS	The user must be able to fight other AI controlled ships	High	High
UR.POWER_UP	The user must be able to collect power ups around the world	High	Low
UR.CHOOSE_DIFF	The user must be able to select a difficulty level	High	Medium
UR.SAVE_LOAD	The user must be able to save and load the gamestate	High	High

Software requirements

Functional requirements

ID	Requirement	Priority	Difficulty
FR.START.SCRN	Software must display a start screen	Low	Low
FR.START.STARTEX	Start screen shall have a ‘start’ and ‘exit button	Low	Low
FR.START.NAME	Start screen shall have a text box for entering a screen name	Low	Low

FR.DISPLAY.GUI	The software must render environment with a Graphical user interface	High	Low
FR.DISPLAY.EDGE	The software must display a lake with boundaries	High	Low
FR.DISPLAY.SHIP	The software must display the user in the form of a privateer sprite	High	Low
FR.DISPLAY.CLG	The software must display colleges	High	Low
FR.DISPLAY.DOCK	The software must display other ships docked at colleges	Medium	Low
FR.DISPLAY.HUD	The software must include loot, points, health and a mini map in HUD	Medium	Low
FR.DISPLAY.CAM	The software's camera must follow the users sprite	Medium	Medium
FR.TUTORIAL	The software must display a tutorial embedded within the gameplay	Medium	Medium
FR.FREEMOVE	The software must allow the users sprite to freely move around the lake via input	High	High
FR.COLLISION	The software must implement a object collision system	Medium	High
FR.BOUNDARY	The software must implement a boundary to the gameplay area	High	Medium
FR.AWRD.POINTS	The software must award points passively and for completing tasks/defeating colleges.	High	Medium
FR.AWRD.GOLD	The software must award gold for completing tasks/defeating colleges.	High	Medium
FR.ATTACK	The users sprite must attack based on the users input	High	Medium
FR.CLG_ATTACK	The college must attack the player	High	Medium
FR.CLG_HEALTH	The college must lose health when attacked	Low	Medium
FR.CLG_CONVERT	When college loses all health, college becomes friendly	Medium	High
FR.CLG_INFO	College must be implemented with [.1] colour, [.2] name and [.3] friendly docked boats [4].Health [5]. Plunder	High	High
FR.OPTNL_TASKS	The game must generate a series of optional tasks that the user may complete	Medium	Medium
FR.KILL_SCRN	The game must display a kill screen on [.1] victory, [.2] loss, [.3] on restart button	Low	Low/Medium
FR.GEN_OBS	The software must generate obstacles in the game world	High	Low
FR.OBS_IMPACT	The generated obstacles must affect the user during gameplay	Medium	Medium
FR.SHOP	The software must have a shop for the player to spend loot in	High	Medium
FR.UPGRADES	The software must upgrade the players capabilities after spending loot	Medium	Low
FR.SHIP_AI	The non-player ships must have some basic AI controlling them	High	High
FR.TMP_UPGRD	The software must temporarily make the player more powerful when collecting power ups	High	Low
FR.CHANGE_DIFF	The software must change certain values based on the players difficulty selection	High	Medium
FR.SAVE	The software must be able to write the current gamestate to a file whenever the player wants	High	High
FR.LOAD	The software must be able to restore the gamestate based on a previously saved filed	High	High

Non Functional requirements

ID	Requirement	Priority	Difficulty
NFR.NETWORK	The program shall not connect to a network	High	Low
NFR.STABLE	The Game shall be stable and not crash	High	Medium
NFR.GAME_TIME	The game shall only last 5-10 minutes - Fit criteria: 9/10 run throughs will last less than 10 minutes	High	Low
NFR.FILES	The game files must be easy to run	High	Low
NFR.CVD	The game must be accessible to those with colour vision deficiency	Medium	Medium
NFR.LOAD_TIME	The software must load quickly - Fit criteria: The game must load in < under 30 seconds	Medium	Low
NFR.SIMPLICITY	The game must be simple enough/have a good enough tutorial to be able to be played by those with no prior experience - Fit criteria: 9/10 players will be able to understand the game by the end of the tutorial	Medium	Medium

Constraint requirements

ID	Requirement	Priority	Difficulty
CR.RESOLUTION	The game should be able to be displayed on range of resolutions and make good use of space - Fit criteria: Display on 13”-27” screens, test with 13”, 47”	Medium	Medium
CR.LOW_SPEC	The game must run on a system with minimum specs 4gb RAM, a standard UK keyboard and mouse - likely a laptop - Fit criteria: Game must be playable on system with these specs	Medium	Low

Environmental Factors:

- **NFR.GAME_TIME -** The game will be used at an open day where time is limited and fast throughput of people is preferable
- **NFR.FILES -** The user may have little to no technical experience, eg. with github and creating JAR files.
- **NFR.SIMPLICITY, UR.TUTORIAL, FR.TUTORIAL -** The user may have little to no game playing experience.
- **UR.RESTART_GAME** The user must be able to restart the game at any time

Associated Risks/Further Comments:

- **UR.START_SCRN:** Any offensive screen names or ones that contain profanity may have to be filtered out. This could be mitigated by a potential filter in the screen box.
- **UR.CLG_POS:** Arrows could be confusing for users, especially colourblind ones and will need identifiers for friendly/enemy colleges.An **alternative** to this could be via the MiniMap
- **UR.VIEW_LOOT:** May take up too much time while being low on our priority list.
- **UR.SEE_TASKS:** Tasks could come across as vague and confusing for some users.
- For any requirements that are about display or graphics there is a potential for internal disagreements about how the UI should look, resulting in unnecessary lost time.
- **FR.COLLISION:** Collision systems can often result in bugs and glitches with clipping resulting in a lowered user experience.
- **FR.FREEMOVE:** Controls must be intuitive and movement must be seamless.
- **FR.AWRD.POINTS/FR.AWRD.GOLD:** Risk of differences between points and gold being unclear to users.
- **NFR.STABLE:** Although the game has low machine requirements, it is hard for us to control the number of crashes that may occur.
- **NFR.GAME_TIME:** There may be a situation in which we compromise gameplay elements in order to reduce game time.
- **CR.RESOLUTION:** Risk of graphical glitches in scaling if not implemented properly.
- **CR.LOW_SPEC:** Performance cap may limit our sprite usage as too many may cause performance issues on lower spec machines

Bibliography

[1]
M. Yousuf and M. Asger, “Comparison of Various Requirements Elicitation Techniques,” *International Journal of Computer Applications (0975 – 8887)*, Apr. 2015.
https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.695.5985&rep=rep1&type=pdf.

[WB.1] https://engteam14.github.io/usecase

[2]
“ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering,” *ISO/IEC/IEEE 29148:2018(E)*, vol. 1–104, 2019, doi: 10.1109/ieeestd.2018.8559686.https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8559686