Change report

Approach to Change Management

Our approach to change management was largely based on applying the new deliverables to the already existing program. Changes were implemented whenever the need for them arose based on the new deliverables. This included designing new requirements, updating the risk assessment, updating our planning methods and designing a new architecture. In the instance of planning we changed the method used by the previous group to be the method we had currently been using, as this is what we were used to.

When adding new features to the code, it was important to keep this documented so we could keep track of which code was new and which was from the previous solution. We extended the already developed classes to fit the new requirements where possible, instead of making entirely new classes. This helped save time and ensure the game ran as smoothly as before. For example, for the power-ups implementation, we extended the obstacle class and implemented the power ups as obstacles, a class which already existed in the program.

Requirements

In order to meet the new assessment 2 deliverables, we have had to create new requirements, both user based and functional. The new deliverables required the implementation of the following:

- A difficulty selection system
- A powerup system
- A save/load system

Therefore the new requirements have been made to ensure these features are implemented. The difficulty selection and power system had requirements previously defined as “may”, so in this instance we only had to change their priority to “shall”. The save/load system required two new user requirements, and three functional requirements to follow this. The requirements link to the new deliverables as follows

Deliverable	USER REQUIREMENT	FUNCTIONAL REQUIREMENT
A difficulty selection system	UR_DIFFICULTY_BEFORE_GAME	FR_DIFFICULTY_SELECTION
A powerup system	UR_POWERUPS	FR_POWERUP_RATE
A save/load system	UR_SAVE_GAME	FR_SAVE_GAME
A save/load system	UR_LOAD_GAME	FR_LOAD_GAME_SELECT FR_LOAD_GAME_PLAY

The new requirements were a very important change that we had to make. Without these concrete requirements it would be very easy to forget to implement the new deliverables. It was important that these new requirements were made as soon as possible, and that they were precise enough to match the deliverables. We continued the format used by the previous team to document the new requirements, simply adding them to the previous table.

Below shows the new requirements we have designed to fit the new deliverables. There are both new user and functional requirements. We didn’t find it necessary to add new non-functional requirements, as those that already existed covered the new deliverables already.

NEW USER REQUIREMENTS

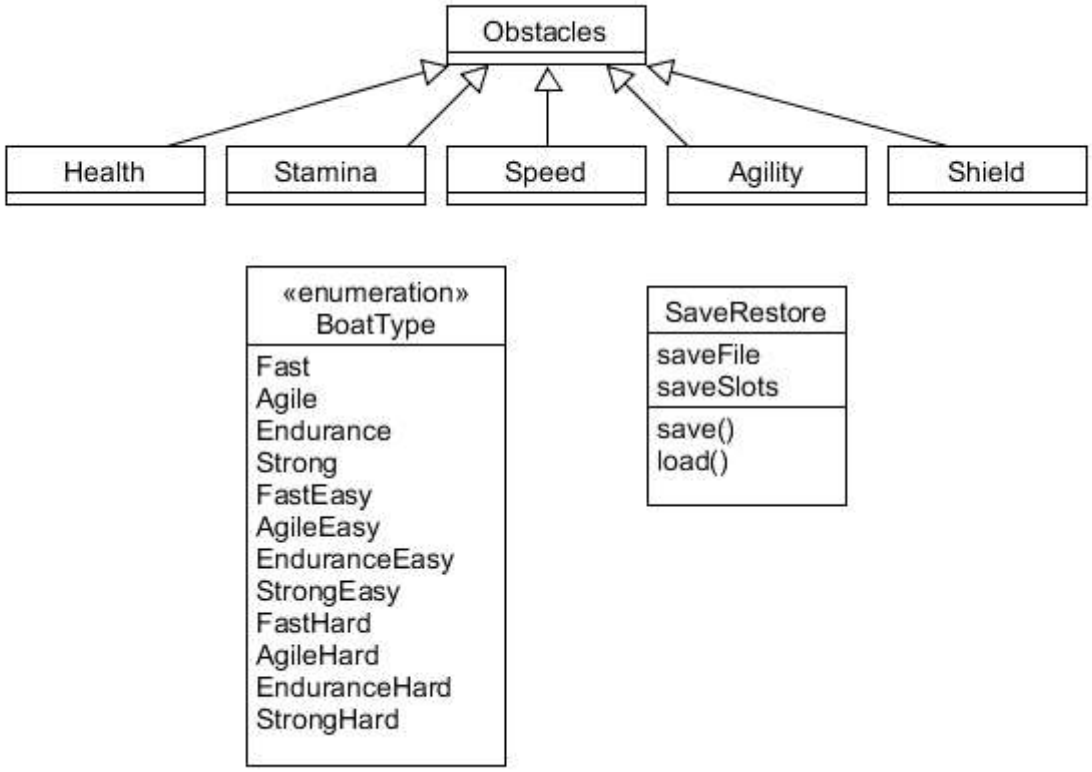
ID	ID Number	Description	Priority	Notes
UR_POWERUPS	UR9	The user may be able to pick up powerups.	Shall	There will be 5 different power ups each with a unique effect on the player
UR_DIFFICULTY_BEFORE_GAME	UR11	The user may be able to choose different difficulty settings before the game.	Shall	There will have to be a difficulty select screen
UR_SAVE_GAME	UR14	The user should be able to save their progress	Shall	The user will have the option to save and quit after each race
UR_LOAD_GAME	UR15	The user should be able to load a saved game and continue playing	Shall	A load game screen will be implemented

NEW FUNCTIONAL REQUIREMENTS

ID	ID Number	Description	User Requirements
FR_DIFFICULTY_SELECTION	FR2	The system allows the user to select a (initial) game level of difficulty from easy, medium or hard.	UR_DIFFICULTY_BEFORE_GAME
FR_POWERUP_RATE	FR11	The system must decide on an appropriate amount of power ups to spawn during a race.	UR_POWERUPS
FR_SAVE_GAME	FR17	The system should provide a feature to save the user's progress, including what race they are up to, the boat they have chosen and the position and times of the boats.	UR_SAME_GAME
FR_LOAD_GAME_SELECT	FR18	The user should be able to select a saved game to load and play with	UR_LOAD_GAME
FR_LOAD_GAME_PLAY	FR19	The saved game should be loaded correctly and be playable with the same features as if it were a new game	UR_LOAD_GAME

Architecture

Abstract Representation UML Diagram

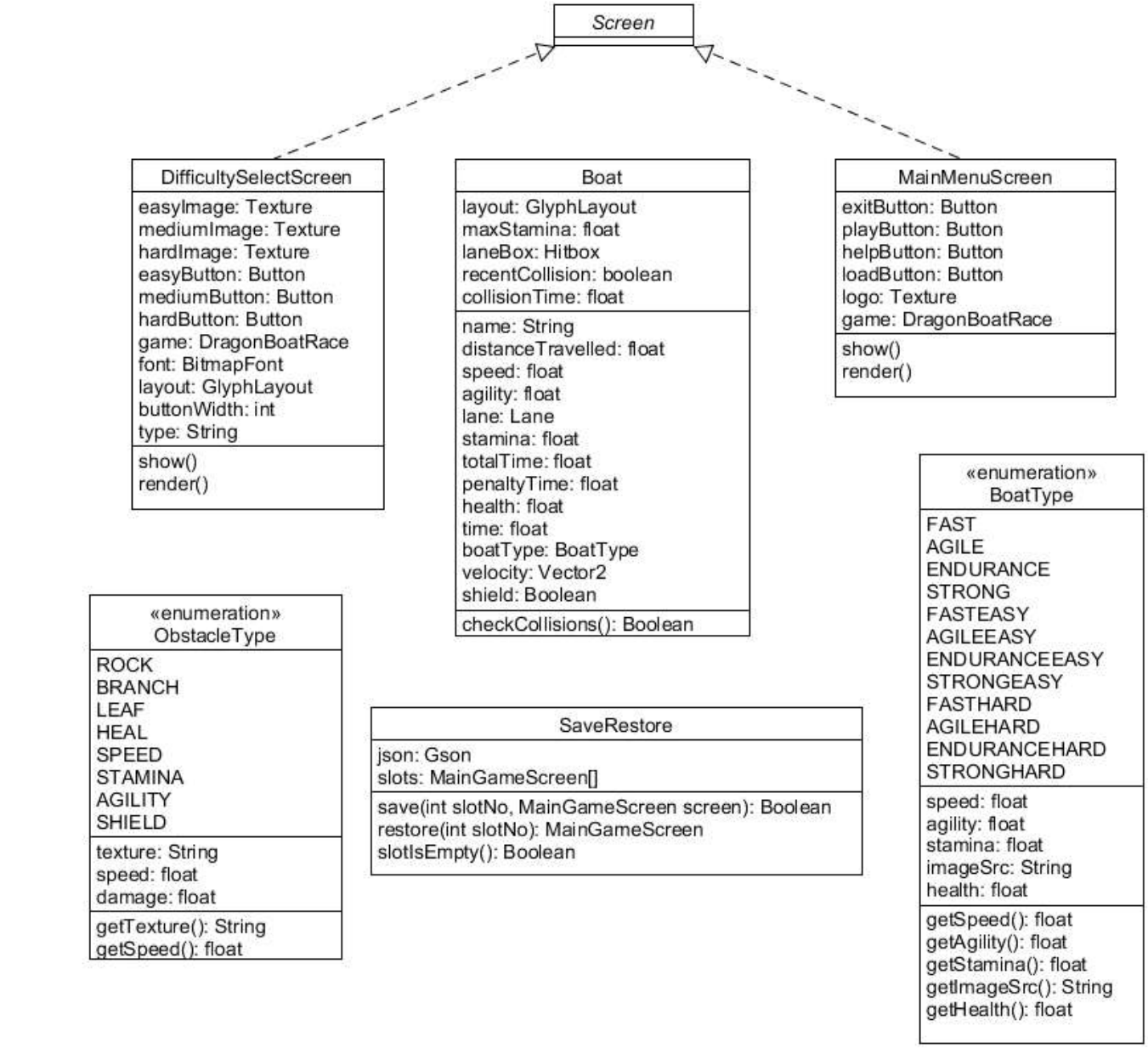


Abstract Representation Justification

For the simplest implementation of the power-ups, it makes sense to utilise the already existing mechanics for obstacles in the game. Powerups work like obstacles, they collide with boats then disappear, except for the effects they have on the boats.

Similarly, for handling difficulty, the player's boat can simply be changed depending on the difficulty that they choose. They will have increased statistics on easy, and decreased on hard.
For the final requirement, saving the game, we have designed a class that is going to have three save slots, and a saveFile format that will go into each one. The two functions save() and load() will be used, one for saving the state of the game to a slot and the other for loading the state of the game to how it was when the game saved.

Concrete Representation UML Diagram



Concrete Representation Justification

Following from the abstract representation, both the BoatType and ObstacleType enumeration have been changed in the concrete UML diagram. The BoatType features all four types of boats with a difficulty addendum of either easy or hard (or none for medium). The effect this will then have is to multiply the values of speed, agility, stamina and health by a constant value (i.e. x 1.1 for easy, x 0.9 for hard).
The ObstacleType now features the five new power ups. All of these power ups will differ from the normal obstacles as they will have damage values of 0 as well as different textures. There are also additions made to the boat class. The boat now has a boolean value for shield which will dictate whether the boat currently has a shield and thus will it take damage from the next obstacle it hits as well as a change to the checkCollisions method which will now contain a switch statement to check if the collided obstacle is a powerup and then take the appropriate action for that powerup (Adding health, stamina etc.).

In order to allow the user to choose the difficulty, we have added the DifficultySelectScreen. This takes elements of the BoatSelectScreen (as well as the type of boat that the user has selected) but replaces the four types of boats with the three difficulties available. The screen will then start the game as it would have done before from BoatSelectScreen but with the modified boat type.

In order to change the architecture to allow for the user to save. We have the SaveRestore class. This class uses the save function to take note of every single attributed associated with the current game state. This function will be accessed by the MainGameScreen class. It will then store this information using the Gson library which converts it to a json file. This file can then be stored locally on the user's machine. From the main menu screen, the user will be able to press the loadButton to choose one of the three save slots and subsequently re-create the game state at the point it was saved using the restore() function in SaveRestore.

Method Selection and Planning

We decided to change the methodology that the previous group used. We have chosen to continue to use Feature-driven development (FDD). We first isolated a list of features that would be required by the new requirements. We gave each of the three features to one member of the team each, who then worked on implementing that feature. This allowed us to avoid interfering with each other's programming and reduced the need to merge overlapping code. After a week, we reconvened to do a group inspection and decide if any changes were needed either to the design or if the team member required help from an additional team member.

- We had considered using other frameworks. The previous group used Scrum but we felt that prioritising elements of work was unnecessary as we believed that with a reduced requirements list, it makes more sense to work on different features simultaneously. Other frameworks like XP don't suit us as we felt that paired programming is not required for this small project.
- This framework ensures that each member of the group would be covering separate elements of the project, therefore avoiding

When it comes to development and collaboration tools, we did not feel it necessary to make any changes as we would still be using libGDX and IntelliJ for programming, GitHub for version control / continuous integration and Zoom / Discord for communication. These are the programs that we used in the first half of the project and are the tools used by the team whose project we took over. Therefore, we are comfortable with these tools and proceeding with their use will allow for the highest productivity and simplest transition.

Risk Assessment and Mitigation

We didn't feel it necessary to make many changes to the risk assessment. For the most part the risks mentioned were general enough to not change if requirements were to change. Many of the risks laid out by the previous team relate to mistakes in management of the team that could result in certain tasks not being completed in time, team members not using time effectively or writing over each other's code. We believe that our chosen methodology will negate these risks in the same manner that the previous group's did. The rest of the risks are applicable to our new task and therefore we will abide by the specified mitigation that the previous group decided on.

However upon play testing the game we noticed a few risks. One of which was related to the water texture used. We felt that this risked losing the user's interest because it wasn't a pleasant texture. We also noticed screen tearing due to the forced framerate and an inability to leave fullscreen. This posed a big risk for people on lower quality machines, and to people who prefer to play their games in windowed mode.

ID	Type	Description	Likelihood	Severity	Mitigation	Owner
R20	Product	User doesn't want to proceed with the game due to ugly assets.	H	M	Assets that are deemed to be jarring should be replaced	Josh Guy
R21	Technology	User experiences screen tearing due to differences in monitor refresh rate and in-game refresh rate.	M	H	Refresh rate should be either locked to a baseline value (i.e 30) or changeable.	Matt Geneux