Engineering 1

Group Assessment 2

Implementation Document

Cohort 2

Team 11

Implementation

Difficulty

UR_LEVELS refers to the new requirement that the game must allow players to choose between different levels of difficulty: Easy, Normal and Hard.

There are 3 different difficulty levels implemented into the game. They are Easy, Medium (Normal) and Hard. The difficulty you wish to play on can be selected in the settings screen on the main menu and then that difficulty is implemented when you start a game.

When changing between difficulty settings four values are changed to different values depending on the selected difficulty level. BOAT_TARGET_SPEED which represents the AI boat's maximum speed and is set to the lowest value of 0.80 when easy difficulty is selected, is increased for medium to 0.88 and is highest when hard difficulty is selected at 0.99. OBSTACLE_COUNT which represents the number of obstacles that are spawned randomly throughout the map, this is set to the lowest value of 50 when on easy and is increased on medium to 100 and set to the max value of 200 when on hard. LEG_OBSTACLE_MODIFIER which adds a modifier to the number of obstacles spawned per leg depending on difficulty, on easy up to 2x by the final leg, medium up to 3x by the final leg and on hard up to 4x by the final leg. The final variable changed when altering difficulty is POWER_UP_COUNT, this represents the number of powerups spawned in a game and is set to its highest value when on easy at 50 and is decreased to 25 for medium and hard difficulty.

Values for this are stored in Difficulty.java but used where necessary, including in AIBoat.java, and BoatRace.java. This implements the Difficulty class from the game architecture.

Game Save

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.

The game can be saved and specific values are stored into a save file to record the required data about the race which can be loaded at a later time for the player to resume their game. The values saved are the current leg number that the player is on, the number of boats in the race currently, the spec of the boat that the player is using, the times for the previous legs of the player and the times of the previous legs for each ai boat.

In order to save the game data representing the current leg (stored as an integer), boat count (stored as an integer), players boat spec (stored as an integer), players leg times (stored as a list of integers) and AI leg times (stored as a list of a list of integers for each boat) are all collected and serialised so that they can be saved to a binary file. This file can be loaded and the data will be de-serialised so that the player can resume their race by loading in the saved values of each aspect of the race into the current game.

This is implemented in the SaveManager.java file and implements the SaveManager class from the game architecture. Data is saved to the state.sav file.

Power-ups

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

There are 5 unique power-ups implemented into the game, they are Repair (refills a portion of the players lost durability), Stamina (refills a portion of the players lost stamina), Boost (Gives the player a temporary speed boost), Time (reduces the time taken on the current leg) and Teleport (jumps the player forward a set distance). When a boat collides with one of the powerups they collect it and its related effect is immediately applied to the player's boat. Each powerup has a unique look so that players can tell which powerup they are collecting.

Power-ups are implemented as an extension of the MovableObject class and their collision is implemented using the CollisionObject class. In this way power-ups act as an obstacle as they are spawned randomly throughout the map and float around with random direction until they are collided with by a player. The main difference is that when collided with instead of reducing the player's speed and durability instead they help the player.

This was implemented in the Powerup.java file. This implements the Powerup class from the game architecture.

Other Changes

There are some other changes to the game that help to improve the experience overall. These are listed below

- Added a short delay after stopping accelerating as without the delay a player could rapidly press the accelerate button on and off and travel at close to full speed while barely losing any stamina. This was done in order to prevent a player exploiting this in order to win easily (UR_TIRED_OVER_TIME).
- Created a new, more robust UI system which the old menus were refactored to use, to make them easier to navigate and more responsive to user input. This was done to improve the user experience overall whilst using the menus (UR_ACCESSIBILITY). This was implemented throughout the UI Package from the game architecture.
- The AI was completely replaced with a more optimised version, and collisions are now looked up against a Quadtree rather than a List, which resulted in the game using approximately 100x less CPU time and improving performance overall (UR_PERFORMANCE). The AI was also tweaked to make smarter and more aggressive decisions such that it would play better in general (UR_EXCITEMENT). This was implemented in the AIBoat Class from the game architecture.
- Minor changes including reformatting the code inline with the Google Java Style Guide, reducing the number of public and protected fields, updating the project to Java 8, unwrapping complicated if statements, and removing unused code.

Unimplemented Features

No features are unimplemented, and all user requirements defined by the client have been fulfilled to an acceptable level.