Implementation 2

Assessment 2 Team

Team 12

Richard Liiv

Umer Fakher

William Walton

James Frost

Olly Wortley

Joe Cambridge

b) Explanation Of How The Code Implements Requirements

Along with the 3 new requirements as part of assessment 2, the game had one more unimplemented requirement, NFR_ATTRIBUTES.

Difficulty - UR_DIFFICULTY_BEFORE_GAME and FR_DIFFICULTY_SELECTION

Different levels of difficulty were implemented through a new variable in the main game class, DragonBoatRace. This was chosen because the existing implementation stored the variables that are persistent, even when changing the screen, in this class - for example, the round the user was currently on.
The difficulty level was stored as an integer from 0 to 3, with 0 being the easiest. This allowed the spawn rate to use this value as a modifier, spawning more obstacles on higher difficulties, making the game harder.

This was implemented in the Lane class' function populateList. The existing code was changed from spawning a max number of n obstacles to spawning a value of $n + difficulty\_mod$ new obstacles where $difficulty\_mod$ is a number determined by the difficulty, shown below:

```
switch (difficulty){
   case 0: difficulty_mod = 0; break;
   case 1: difficulty_mod = 2; break;
   case 2: difficulty_mod = 10; break;
   case 3: difficulty_mod = 30; break;
}
```

This was chosen as it aligned with the existing implementation for storing the round number and provided a simple way of increasing the difficulty without re-engineering the project.

Allowing the user to choose a difficulty level was implemented through a new class called DifficultySelectScreen. The name of this class was chosen to match the existing naming convention for classes that inherit from screen. The functionality of the class is very similar to the BoatSelectScreen so it was used as a skeleton, with difficulty related changes made such as changing the images to match the required difficulties.

The reason for making a new class for selecting difficulty, rather than adding onto the already existing selection screen for boats, was because the team decided that the customer's requirement listed in NFR_POSITIVE_UX would be violated through a cluttered user interface. By separating the difficulty and boat selection to different screens, the interface remained clear and positive.

Power-Ups - UR_POWERUPS and FR_POWERUP_RATE

Power-ups were implemented by adding new types of obstacles. This was done to minimise the amount of new code needed and maximise the already existing code usage. The team decided to do this as it minimised the amount of new code that would need to be tested for problems and kept the codebase concise for any new team that may work on the game.

New variables were added to the ObstacleType enum class to allow instances of the Obstacle class to store collision effects for things other than health. For example, the variable staminaMod was implemented to allow the obstacle to change a boat's current stamina. New enum options were also added to ObstacleType to represent the power-ups and allow creation of them such as ObstacleType.PU_SPEED.

The game already had functionality to apply changes to a colliding boat in the Boat checkCollisions function. This was also extended to allow boats to modify other attributes such as their speed if they hit a speed powerup. This was appropriate as it minimised the need to add new convoluted functionality such as a specific check to see if the boat collided with powerups specifically.

The spawn rate of new obstacles was changed to fulfill FR_POWERUP_RATE. This was done by changing the randomObstacle function in Lane which is responsible for deciding on what new obstacle to spawn. The old implementation spawned a random obstacle, however this created a game with too many power-ups. The team decided this violated FR_POWERUP_RATE so randomObstacle was changed to choose a random obstacle based on a custom probability table:

```
// random variable from 0 to 99
int randNumber = (ThreadLocalRandom.current().nextInt(0, 100));
int i = 0;  // index of new obstacle to make

//convert the random number into an index
if (randNumber < 20)
   i = 0; // 20% chance of LEAF
else if (randNumber < 40)
   i = 1; // 20% chance of ROCK
...  // middle cases hidden to reduce size
else if (randNumber < 100)
   i = 4; // 3% chance of PU_ALL
```

This was chosen as the solution to FR_POWERUP_RATE as it allows for fine control over the spawn rate of every obstacle type while minimising the amount of changed code, thus minimising the need for testing.

Saving - UR_SAVE_RESUME_GAME

To implement loading for UR_SAVE_RESUME_GAME, changes were made to the MainMenuScreen class. A new function, loadSaveFile(), was created that will take a file object as input and load the stored settings into the current game instance. This was chosen as it aligns with what was already implemented for the new game button and aligns with common video game practice to have the load button on the main menu.

To implement saving, a new class MainGamePauseScreen was created. This was made because it was decided to implement “saving at any point” whilst maintaining NFR_POSITIVE_UX, a pause screen was needed. This allows the user to pause the game to save, or take a break, then resume at the point they paused from. To implement the actual save functionality, a new static function MainGamePauseScreen.saveToFile() was created. This is static to allow it to be called from anywhere and because it is not reliant on any class.

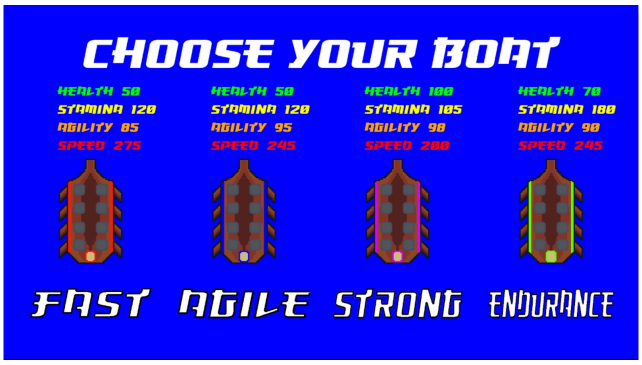
The format for the save file was decided to contain the chosen boat type, the player’s total round time, the player’s current round and the difficulty chosen. Details such as the boat’s position were not included as the team decided that using a checkpoint system (saving after each leg) would make the code much simpler and less prone to failed tests. The team also decided that this was appropriate because it reduces the ability to “save scum” midway through a race, allowing the intended difficulty to be applied.

By saving the state to a file, the saved data will remain even after the user closes the program. It also allows a more advanced user to store multiple saves and switch them when needed. This feature wasn't implemented to the game however as it wasn't a requirement given, but it would be easy to implement if the client wanted to add it.

Exceptions were used to ensure that the save file being loaded was valid. This was to continue the convention set by the java library for file operations. If the file reader encountered an error, it would throw an exception. By throwing exceptions if the contents of the file aren't valid, one catch block can be used to check the entire loading process at once.

Attributes - NFR_ATTRIBUTES

NFR_ATTRIBUTES was left unimplemented by the previous team, so we decided it would be necessary to finish it. To show the boat statistics, the BoatSelectScreen was changed to include extra font textures displaying each boat's stats. This was done in a dynamic way, reading the values stored in the BoatType enum for the display. This was done as it allows a new team, or the client, to change statistics they want for each boat whilst having the display update automatically. A screenshot from the finished screen is shown above.



Whilst doing this, there was a change made to how Buttons and Textures were stored in BoatSelectScreen. They were moved from all benign explicitly defined to being defined in a list. This was done to allow the program to loop over all boat types, removing the copied lines used previously. However, this wasn't done for any other class as no other class' render methods required modification and it was decided by the team that it wasn't worth changing what already works for no performance gain.

Background and text UI changes

Due to the previous team's choice of colours the text was hard to distinguish from the background in places. As a result, the information that was displayed by the health and stamina bars made was hard to read for the user and as this is important information the UR_UX requirement and specifically NFR_POSITIVE_UX non-functional requirement could be affected. The game may not have provided a pleasant user experience and could have affected the enjoyability and user engagement negatively.

Thus, we modified the colours of the health and stamina bars to make it easier to tell apart from the background. The background itself was changed as well to complement the colours of the text. We have included a test to further check readability and accessibility of UI elements. Please see TM_POSITIVE_UX.

Here are some examples of UI element changes.[1]

[1] UI elements changes:
https://github.com/UmerFakher/ENG1Project/commit/26989e8d0ef2160e2f6360886a16414fccf2b94e
https://github.com/UmerFakher/ENG1Project/commit/26471aabebc3b24739c57b9d6090e1727e8512fd

Other Changes

Round Setter

The setter for DragonBoatRace.round was identical to the method upRound(), both incrementing the current round. This was considered confusing by the team, so the setRound() method was changed to a standard setter. All uses of setRound() were then replaced with upRound().

Removing Redundant Code

There were variables and methods within the existing codebase that were never used. To increase the readability of the code for both our team and any new team that may take on the project, these were removed. For example, the getSpeed() method in Obstacle was never used and could cause confusion, thus was removed.

Changing Lane Constraints

The previous team set the limit on how far you could move out of your lane to the width of the lane, so that you could never leave your lane and would never receive a penalty for such an action. To fix this, the cap on horizontal movement was increased to allow the user to exit their lane.

Changing Lists

The team decided it would be best if the simplest possible class was used for storing lists of elements. We decided this to allow us to change the type of list used. For example, if a custom sorted list was implemented, this could be easily swapped out without needing to change the base variable type. Because of this, all ArrayLists were changed to be Lists unless an ArrayList was the only possible solution.

Adding Comments

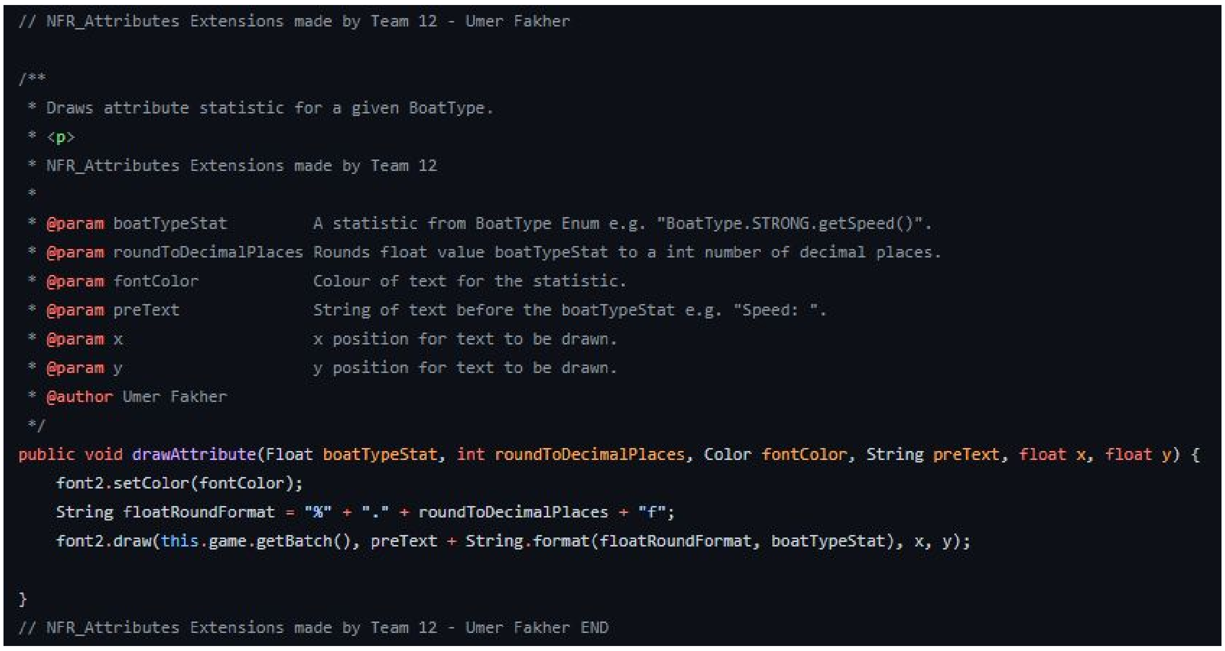
We made sure to add comments in JavaDoc fashion in order to help with code usability for members of our team reading code changes as well as for future use. After inheriting the code from Team 15, we identified that there was a lack of comments so we added them to some classes where we deemed appropriate. You can see an example below in the appendix of a commented function following JavaDoc.

All Changes

Here is a comparison of Team 15's project and changes we have made in assessment 2:
https://github.com/JoeWrieden/ENG1Project/compare/master...UmerFakher:master

Appendix

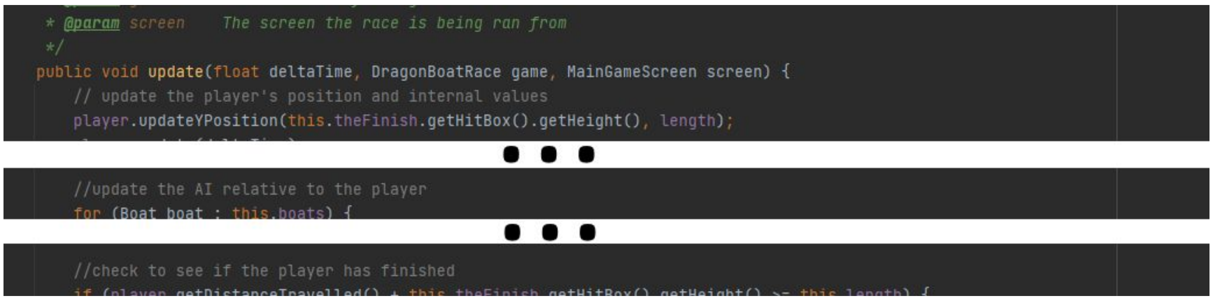
You can see an example below of a commented function following JavaDoc.



```java
// NFR_Attributes Extensions made by Team 12 - Umer Fakher

/**
 * Draws attribute statistic for a given BoatType.
 * <p>
 * NFR_Attributes Extensions made by Team 12
 *
 * @param boatTypeStat         A statistic from BoatType Enum e.g. "BoatType.STRONG.getSpeed()".
 * @param roundToDecimalPlaces Rounds float value boatTypeStat to a int number of decimal places.
 * @param fontColor            Colour of text for the statistic.
 * @param preText              String of text before the boatTypeStat e.g. "Speed: ".
 * @param x                    x position for text to be drawn.
 * @param y                    y position for text to be drawn.
 * @author Umer Fakher
 */
public void drawAttribute(Float boatTypeStat, int roundToDecimalPlaces, Color fontColor, String preText, float x, float y) {
    font2.setColor(fontColor);
    String floatRoundFormat = "%" + "." + roundToDecimalPlaces + "f";
    font2.draw(this.game.getBatch(), preText + String.format(floatRoundFormat, boatTypeStat), x, y);

}
// NFR_Attributes Extensions made by Team 12 - Umer Fakher END
```



```java
 * @param screen    The screen the race is being ran from
 */
public void update(float deltaTime, DragonBoatRace game, MainGameScreen screen) {
    // update the player's position and internal values
    player.updateYPosition(this.theFinish.getHitBox().getHeight(), length);
```

● ● ●

```java
    //update the AI relative to the player
    for (Boat boat : this.boats) {
```

● ● ●

```java
    //check to see if the player has finished
    if (player.getDistanceTravelled() + this.theFinish.getHitBox().getHeight() >= this.length) {
```