IMPLEMENTATION

Team 20: Tequila Mockingbird

Exam Numbers: Y3872854, Y3877651, Y3872779, Y3873031, Y3869788

Architecture

Our code implements our architecture by using the preset super classes and inheriting from the relative class. This allowed us to create new UI menus and entities easily utilising the previous architecture from assessment 1. For example, the newly implemented bonuses utilise the same features as the obstacles to allow the boats to collide with the bonuses and receive the effects.

Furthermore, there has been a UI addition for a pause UI. This was created through inheriting the UI super class and implementing the different methods relating to pausing. The static GameData class holds global static variables that allow us to easily access data that spans across multiple classes. For instance, a new global variable, isFromSave has been added to this class to allow the game to know if it should be loading the game from a save or not.

The saving of the game has been added using the previous architecture and it's readily available variables, allowing the writing of data to a json file easily.

Requirements

All the requirements for assessment 2 have been met fully met. We have successfully met them with the following implementations.

The UR_DIFICULTY_CHOICE requirement was implemented with a slight modification to some existing static data. The first of which was the AI boats statistics. The statistics now get a multiplicative increase based on the difficulty set in the game. The number of obstacles is no longer set to a static 30, the number of obstacles is now multiplied by some constant based on which difficulty is set. Additionally, the number of bonuses (which are discussed further in this section) are also multiplied by some constant based on the difficulty, however this constant decreases the number of bonuses. This addition is in the Lane class.

The UR_SAVE_ GAME requirement was implemented by utilising a json file with a snapshot of the whole game state. This was implemented by writing the following things to a json string:

- Boat statistics and position
- Obstacle position and type
- Bonus position and type
- The current timer for the race
- The difficulty for that race
- The standings for that race
- A boolean representing whether the game was saved at the end of a race
- The current leg of the competition

Once these were all added to a Save class, they could be written directly to a string by using the gdx json library. When a player wants to load the game again, they can do so by clicking the "load previous game" button on the main menu. This button will load all the json file data into the corresponding classes and switch the state over to

the game state. If there is no save present, then the button on the menu will be “greyed out” allowing the user to be aware that the option is not available.

The UR_BOAT_ BONUS requirement was implemented by utilising a similar system to the obstacles.
The bonuses are implemented by creating a new class, Bonus. This class holds the sprite and body of the bonus as well as the type of bonus that it is. These bonuses are spawned in when then the current leg starts and the number of them depends on the difficulty that has been selected. The system works by first checking if a boat and a bonus have collided, if they have then the boat’s body is added to a list called toGetBonus. This list is then iterated over in the game loop (render() in the Game class) and the bonuses are applied to the boat that collided with it. There is a method in the Boat class that is essentially just a switch statement that checks which bonus was hit and then gives the correct ‘buff’ to the player.

<u>New features</u>

- A pause game menu has been added, when the player presses the escape button, the game is paused and the user can exit to the menu or resume the game. This is an intuitive feature that allows the player to make a decision before leaving the game.
- The game is able to be saved and loaded by escaping the currently playing game and clicking the load last game button on the menu.
- Difficulty settings have been added allowing the player to have more of a challenge. The difficulty increases by increasing the amounts of obstacles, decreasing the number of bonuses and increasing the opponent boats statistics to overall create a more challenging experience. This makes the game harder as the player has to dodge more obstacles, has less chance at getting an advantage from the bonuses and also has to race against much better boats.
- Bonus powerup are now available within a race, there are 5 bonuses which all have their own effects: the “wind” powerup gives the boat a boost to its acceleration, the “cola bottle” gives an increase to the boat’s speed, the “wheel” powerup gives a maneuverability boost, the “toolkit” powerup repairs the boat increasing the robustness and finally the “lightning bolt” powerup gives the player a boost to stamina. These are correctly defined as power ups as they give a hard increase to the boat's stats which only increases the performance of the boat.
- A final screen has been added so that after all the legs have been completed the positions of all the boats at the end of the race are displayed. This was previously left out of the product and was added to give better clarity to who actually won the overall game.

- Volume controls for the soundtrack have been added to the menu allowing the user to reduce the volume if it is too loud. The base volume of the music can be very loud for some users so volume controls were added to make sure the user experience is not impacted by the loud volume.

New algorithms and data structures

A few new algorithms and data structures have been adjoined to the existing code base to aid the implementation of the new requested features. These are discussed below.

Data structures

Every piece of data that needs to be saved for the game to be loaded has been given its own data structure. These structures hold the data that needs to be saved for the corresponding class. For example, BoatSave holds all the data that a single boat needs to hold.

Algorithms

There are two new major algorithms that have been implemented. The first one is the algorithm that inputs the difficulty and calculates all the settings that are connected to that difficulty. For example, with a hard difficulty, the AI boats statistics need increasing and the obstacle/bonus count needs to be adjusted.

The second major algorithm is the saving/loading algorithm. To save the game, all the data that needs to be saved is packed into the SaveX (where X is the related data to that save class) data structures. Once this data is all packed up, the data is written to a json file. To load the game, the json file is simply read and the game objects are created with the values taken from the file. This results in the ability to save and load one previous save.

There is a smaller algorithm worth noting. The first of which is the application of effects when a bonus is picked up. This simply modifies the statistics of the boat that touches the bonus resulting in a positive effect.

Significant changes

- 4 buttons have replaced this 1 play game button, now there are buttons that allow the player to decide which difficulty they want to play, or if they want to load the game. This change was made to meet the difficulty selection requirement and the UR_SAVE_ GAME requirement. The class where these changes have taken place is the MenuUI class.
- The number of obstacles is no longer a fixed number, it is adjusted based on the difficulty level. This was part of the UR_DIFICULTY_CHOICE requirement as the number of obstacles highly determines the difficulty. The class where these take places is the Lane class.
- Multiple bugs were fixed, the main ones to note are: a large memory leak caused by iteratively creating new UI instances faster than the garbage collector could remove them. Also there was an issue with getting the

mouse's position while the game was running, this was caused by the mouse position being in world space and not screen space. The class where both of these changes took place is the Game class. This change does not directly link to a requirement however, these changes had to be made to allow the game to be played overall.

Missing features

All the required features for this product have been implemented.