# Implementation

**Assessment 2**


## Team 10: Hard G For GIFs

Dragos Stoican

Rhys Milling

Samuel Plane

Quentin Rothman

Bowen Lyu

Project's GitHub repository:
https://github.com/hardgforgifs/second-term-game
Executable jar:
https://github.com/hardgforgifs/second-term-game/releases/download/1.2/Dragon.Boat.Racing.jar
Source-code:
https://github.com/hardgforgifs/second-term-game/archive/1.2.zip

Every addition to the code has been marked with the comments “Added block of code for assessment 2” and “End of added block of code for assessment 2”. If the code is heavily based on the old implementation, we used “Modified block of code for assessment 2” and “End of modified block of code for assessment 2”. Every significant change made is documented here with justifications, and related to relevant requirements where necessary. More insignificant changes are still marked in the code, but are not covered here.

## New significant features:

- As seen in the architecture documentation, we added seven new classes that implement the pickup powerups. The classes follow the rules of inheritance used by the other team. One of the classes is used for the collidable object found in the river, and the others are used to apply a random effect to a boat that collides with such an object
- We added the feature of pausing the game. From this pause menu the user can save the game and return to menu, or just resume the game
- The save and load features use the Preferences library offered by libGdx to save information about the game in a XML-like format, with key-value pairs.
- We use Music objects to play music during the game. Settings for music are saved in a XML-Like format in a preference file.

## Systematic report of significant changes:

**Boats spec_id and boat stats:**
Justification: In the previous implementation the spec id of the player was passed from the select boat screen, and the AI had no spec_id, so all AI were the same. We modified by making the spec_id an attribute of each boat. Based on this attribute we are creating a boat with certain specifications and textures. The AI boats have this attribute randomised, while the player boat has it modified based on the user’s input in the select boat screen. This allows us to easily create boats with the specifications and textures we want.
Class: Boat (Methods: setSpec, setTexture, setStats)
Requirement: UR_BOAT_SPECS, FR_STATS

**New PowerUp class:**
Justification: We had to create a new class for a new object in the game. This class handles the creation of the powerups in the race, including the boundaries of it’s collision body. This is also where we created a method that randomises the effect of the powerup. Each powerup is assigned an Effect on creation.
Class: PowerUp
Requirement: UR_PICKUP_BOOST, FR_RANDOM_BOOST

**New Effect Class:**
Justification: This is the parent class for five types of effects a boat can get from a powerup. An effect has a duration, texture, and an attribute “is_active”. The duration defaults to five and is reduced on every update. Once the duration reaches 0, the effect becomes inactive and the boat is not affected by it. These behaviours are implemented in the applyEffect method of the Effect class, and this code is accessed by all subclasses of Effect through a super() call. We get the sprite to display on the screen through the getSprite method.

Class: Effect
Requirement: UR_PICKUP_BOOST, FR_BOOST_DURATION

**New Class for each type of effect:**
Justification: We made a new class for each type of effect because it makes applying, and displaying current effect very easy. These classes inherit from the Effect class and their constructors initialize the texture and sprite attributes based on the type of boost. These classes also override the applyEffect method to add their modifications to it, for example, the speed boost would need to increase the maximum speed of the boat. This method ends with a super call to the body of the method in the Effect class that handles reducing duration.
Class: SpeedEffect, StaminaEffect, InvulnerabilityEffect, ManeuverabilityEffect, RepairEffect
Requirement: UR_PICKUP_BOOST, FR_RANDOM_BOOST

**Boat's list of effects and applying effects:**
Justification: We had to add code to deal with applying the effect of pickups on boats. To do this we have a list of effects that are currently affecting the boat. When the boat collides with a pickup we receive the randomised type of effect that is applied to the boat. This effect is added to the list, then at every step in the race we call the method updateBoostEffects to apply each effect in the list, using the applyEffect method. We chose this implementation because it allows the boat to be affected by multiple effects at once.
Class: Boat (Methods: checkCollisions, updateBoostEffect) for handling collisions and applying effects respectively
BoatRace (Methods: runStep) for checking collisions
Requirements: UR_PICKUP_BOOST, FR_MULTIPLE_EFFECTS

**Modified effect of colliding with obstacles:**
Justification: In the previous implementation, colliding with obstacles caused the maximum speed of the boat to decrease, we changed it so it now affects the current speed. We believe this makes colliding with obstacles affect the player in a more realistic manner.
Class: Boat (Methods: hasCollided)
Requirement: UR_COLLISIONS, FR_HIT_OBSTACLE

**Leg difficulty:**
Justification: As required in the first part of the assessment, we added the increased difficulty of each leg, as this was not implemented by the other team. To add in this feature we now increase the number of obstacles, as well as the speed they move at after each leg.
Class: BoatRace (Methods: setLegDifficulty)
Requirement: UR_LEG_DIFFICULTY, FR_MOVING_OBSTACLES

**Saving and loading the game:**
Justification: We added the necessary methods for saving and loading the game in the main class of the game, PixelBoat. The saving is done by storing all information about the race, the collidable objects and the boats into a preference file using the Preferences library offered by libgdx. In the saveGame method we take the current state of the game and save it in a preference file called "save" using key-value pairs, similar to an XML document. In the loadGame method we use the information in the save file to reconstruct the same game state. We also check if there is a saved game when we start the game. If there is no saved game the load button on the screen is inactive. A new load game button was also added in the start menu. When the game is closed, the dispose

method is called and the changes to the preferences files are saved, so the game can be loaded even after the application was closed.
Class: PixelBoat (Methods: saveGame, loadGame, create, render)
StartGameScene (methods: draw, update) for creating the load game button
Requirement: UR_SAVE_GAME, UR_LOAD_GAME, FR_NO_SAVE_FILE, FR_SAVE, FR_LOAD

**Modified boat selection screen:**
Justification: We added a lot here because the previous implementation had a very limited boat selection screen, with only three different options, and only to boat specs. We added a sprite for each type of boat, and we now have 5 types of boats. Clicking one of the displayed boats will make it the player's selected boat and display it's associated stats. The game starts when the continue button is pressed. In the game, the color and the specs of the boat match the one that was chosen in the selection screen. We also chose to add the difficulty selection on this screen. The user can choose the difficulty between easy, normal and hard, with normal difficulty selected by default.
Class: SceneBoatSelection (Methods: SceneBoatSelection constructor, update, draw)
Requirement: UR_BOAT_SPECS, FR_BOAT_ASPECT, UR_DIFFICULTY, FR_DIFFICULTY_SELECTION

**Randomising AI boat types:**
Justification: The previous implementation didn't add different types of AI, instead making it use the same boat all the time. We changed that so the AI is now different every race by randomising the type of boat each one of them use, making the races feel more unique and engaging.
Class: SceneMainGame (Methods: SceneMainGame constructor)
Requirement: UR_ENEMY_BOATS, FR_AI_VARIETY

**Pause game feature:**
Justification: During the game, the player can now press the 'p' button to pause the game. When the game is paused, there is an option to save the game and return to the main menu, and an option to resume the game. We implemented this by creating a boolean attribute is_paused in the SceneMainGame class. If this attribute is true, then we are not calling the runStep method of the race, instead we display the two new buttons on the screen. This way, everything related to the race stops advancing, including boats and timers.
Class: SceneMainGame (Methods: SceneMainGame constructor, draw, update)
Requirement: UR_PAUSE_MENU, FR_PAUSE

**Select Difficulty:**
Justification: We added the feature of selecting the difficulty of the game in the boat selection screen. The difficulty affects the delay before the stamina starts to regenerate after a boat has stopped accelerating. Each boat has a difficulty attribute that is used in the setStats method to change the values of time_to_recover which controls the aforementioned delay. The player's difficulty is passed from the SceneBoatSelection update method when the user is clicking the buttons
Class: Boat (Methods: setStats, setDifficulty)
SceneBoatSelection (draw, update) for changing difficulty based on input
Requirement: UR_DIFFICULTY, FR_DIFFICULTY_EFFECT

**Modified option screen:**
Justification: We added a new option screen where the user can change the volume of the sound. We used Slider objects from libgdx to implement the sliders, with user preferences files saving the user's settings in XML-like format. This means settings persist after closing and opening the game.
Class SceneSettings
Requirement: UR_SETTINGS, FR_CHANGE_VOLUME

**Stamina changes:**
Justification: To implement the changes made to stamina we added new attributes to check if the boat has spent enough time before the stamina starts to regenerate again to prevent the stamina recovery rate from being exploited. We are also checking if the boat runs out of stamina completely, in which case acceleration becomes unavailable until stamina charges back up.
Class Boat (Methods: updatePosition, accelerate)
Requirement: FR_STAM_REGEN, FR_STAM_DEPLETED

**Durability:**
Justification: The old implementation didn't implement boats being destroyed when their robustness hits 0. We added this feature and now running out of health causes a boat to be removed from the current leg, receiving a leg time of five minutes.
Class BoatRace(Methods: runStep)
Requirement: FR_HIT_OBSTACLES

**Changes to result screen:**
Justification: The previous implementation's result screen wasn't aesthetically pleasing and the results weren't ordered properly. We modified this by reusing the background from other scenes to continue the theming from the rest of the game and add some colour, creating a table asset in which to display the results in a clearer manner and ordering the results based on how quickly the last leg was completed. Once all the legs have been completed, the fastest results from the day are displayed. Only the top 12 results are shown, adding to the sense of competition and reducing the sense of being presented with a wall of text, again improving the aesthetics.
Class: SceneResultScreen (Methods: draw, drawStatic), PixelBoat(Methods: render)
Requirement: UR_RESULTS

**Notification when penalised:**
Justification: We made a small change to display a small notification next to the timer if the player is outside the lane.
Class: BoatRace (Methods: drawTimeDisplay)

## Unimplemented features:

**UR_INFO_DISPLAY:** Speed, acceleration, position in race, distance to go are not displayed on the screen

**FR_FULLSCREEN:** We have a settings menu, but fullscreen options wasn't implemented yet

**Fr_DIFFICUITY_DISPLAYED:** The difficulty is not displayed on the screen once it was chosen. Nor is the leg difficulty displayed in any way