Module	Engineering 1(ENG1) - COM00019
Assessment Title	Assessment 2 - Cohort 2
Team	Dragonite (Team 21)
Members	Okan Deniz, Rhianna Edwards, Omar Omar, Omar Galvao Da Silva, Craig Smith, Joel Wallis
Deliverables	Implementation

Code modifications followed the requirements and architecture set out by the team. The team focused on making necessary changes to the code when needed. As such, the extensions made were to satisfy the requirements[1] and follow the structure outlined in the architecture[2]. The comment labels "ASSESSMENT2:START" and "ASSESSMENT2:END" were used to indicate where new code was added throughout.

The changes to the code can be summarized as follows. As per the requirements[1], New classes were created to implement boat power ups in the game. Furthermore, New functionality was added to certain classes including GameScreen to allow the player to save their game while new classes were added to provide the user with the necessary graphical interface to interact with the game. This can be seen in the WelcomeScreen and how it serves its main use of allowing the player to load their saves or set the game difficulty. Minor changes were made to implement different difficulty modes. All these changes can be traced back to the requirements[1] and architecture[2] documents. The table explains these changes in detail:

Change title	Description	Classes involved	Related requirements
New PowerUp class	This inherits from Obstacles, as it requires similar methods to the Obstacles class, so it allows for software reuse. The damage is set to 0 as it is a power-up. And everything else takes in the same values for the constructor as the Obstacles class except instead of Obstacle type it is power-up type. Extra methods implemented are: ● GenerateTextureFrames() – returns all the textures based on the type of power-up. ● Update(float deltaTime) – updates the frame based on how much time has passed. ● getType() – returns type of power-up. ● GetMysteryTexture() – returns a mystery box so the player doesn't know what the power-up is until it is picked up. 4 extra variables are used to have the power-ups animated: ● currentFrameTime – stores how long the current frame has been displayed in the render function ● noOfFrames – stores the number of sprites based on type and the assignment is in GenerateTextureFrames() ● frames – created using the GenerateTextureFrames() method ● frameCount – keeps track of which frame the animation is currently on and increments after the currentFrameTime is equal/above maxFrameTime. As well as one constant: ● maxFrameTime – max amount of time each frame is to be displayed and is set to 0.05.	PowerUp Obstacle	UR_POWER_UPS NFR_ANIMATION
New PowerUp child classes	New classes are created for each power-up: TimeReduction, Maneuverability, SpeedBoost, Invincibility and Repair.	TimeReduction, Maneuverability, SpeedBoost, Invincibility and Repair	UR_POWER_UPS UR_POWER_UPS_COUNT
Spawn power-ups	Firstly 2 variables were created in the DragonBoatGame class:	DragonBoat Game	UR_POWER_UPS

	● powerUpTimes – y values of when the obstacle is going to appear in each lane. ● noOfPowerUps – number of power-ups per lane Using the same method as for obstacles for consistency. In create() It adds random y values into powerUpTimes and the number of y values is dependent on noOfPowerUps which is 4. Each of the lists in the array is sorted so it is in order. Also, in AdvanceLeg() the same methods are used for power-ups as obstacles to instantiate the obstacles for the next leg, which is the same method as create() in which it generates other y random values for power-ups. Then in GameScreen, for each lane, it generates a random x coordinate in the variable xCoord and passes this and a random choice from the powerUpTypes to SpawnPowerUp() to instantiate the power-up in the lane. In Lane 2 new variables are created for power-ups: ● PowerUps – ArrayList storing the power-ups, this is because there will be a varying number of power-ups during the game. ● powerUpLimit – this an integer set to 4, so the lane is limited to show 4 power-ups at a time. Along with one method: ● SpawnPowerUps- works same as spawn obstacles, instantiate the power-up with the correct type and then adds the power-ups to PowerUps		
Power-up collisions	In the Boat class, variables added: ● boatPowerUps – an array which can store up to 2 power-ups, which is picked up after the boat collides. ● isInvincible – a boolean value that's only true after the player activates the invincible power-up. ● invCounter – an integer that counts the number of obstacles player collided after activating invincibility so after 3 it can be turned off ● reductions – stores the time reduction given to player Changed CheckCollisions method to include powerUps, loads all powerups in the boat's lane into a local variable PowerUps. It checks if it collides the same way as obstacles – by if the two textures overlap, then adds the powerUp to boatPowerUps using AddPowerUp. After, it is removed from PowerUps in lane. Before checking for obstacle collisions, it first checks if invincibility is activated and decreases the invCounter by 1 until when it hit 0 then, isInvincible is set back to false. Added 1 method for power-up collisions: ● AddPowerUp – checks if there's empty values in boatPowerUps, if so add the power-up. This is for better modularity and makes the code more reusable.	Boat GameScreen Lane	UR_POWER_UPS FR_COLLISIONS

	In Lane class, one method is created to remove power-ups from lane: ● RemovePowerUp – takes in the power-up to remove and removes it from the ArrayList (PowerUps)		
Using Power-ups	In GetInput in Player class, a new condition is added so that when the player presses 'e' it will apply the power-up if boatPowerUps is not empty. ApplyPowerUp takes in the power-up and it applies the relevant effect by changing variable values.	Player Boat	UR_POWER_UPS UR_CONTROLS
Changing maneuverability to a variable	Replaced MANEUVERABILITY with maneuverability so it can be changed for the maneuverability power-up. originalMan – used to store what the maneuverability is before it could be changed in applyPowerUp(). Maneuverability is set to originalMan in the reset method.	Boat	UR_CONVENTIONS UR_POWER_UPS
Displaying graphics for power-ups	New powerUpEmpty is a constant Texture which stores an image of an empty box displaying that the slot is empty if there is nothing on top of the Texture. It is assigned in the constructor. Then in render(), it draws the boxes then if the player has any power-ups in boatPowerUps for the player, it is then drawn on top of the empty slots. Extra loop is added with obstacles to move power-ups down the screen as power-ups are stored in a different ArrayList to obstacles in lane. When it is rendered in the lane, getMysteryTexture() method is used to get the itemBox image so that the game is more fun - the actual texture of the power-up is not displayed to show what power-up it is before the player picks it up. Finally, all textures for the powerUps stored in the Lane class are disposed in the end, so there will not be a memory leak.	GameScreen Lane	UR_POWER_UPS
Created levels	The player can choose the mode by clicking on easy/medium/hard. This is then stored as a String variable for easier comprehension of the code.	DragonBoat Game	FR_GAME_MODES
AI changes based on difficulty	Level is passed into the ai function as a parameter. So AI runs slightly slower if it is easy and slightly faster for hard.	Opponent	FR_GAME_MODES
Number of obstacles increase based on mode/level	Depending on the level, the number of obstacles increase is different, with easy increasing by only 1 each leg, medium 2 and hard 3. The game starts with 8 obstacles regardless of difficulty level as otherwise the player doesn't need to worry about dodging them as it is unlikely to break the boat.	DragonBoat Game	FR_GAME_MODES
Goose movement depending on level	In GameScreen, before the obstacles all move down the screen, it is checked if it is a Goose and if then it will pass an extra parameter as game level so the goose can move depending on the level, they are less likely to move as randomly on easy.	GameScreen Goose	FR_GAME_MODES
Fixed AI	In the ai function the bounds are fixed so checkInLane now works, and the boats are less likely to leave their lane. New parameter added to pass in course for the coundaries	Opponent	FR_AI

Fixed boundaries for collisions	CheckCollisions - the bounds for collisions based on a threshold which was removed and replaced by the dimensions of the obstacle/power-up so it adjusts to the size of the power-up.	Boat	FR_COLLISIONS
More variety of obstacles	Changed the obstaclesTypes in GameScreen to have more logs and rocks. Updates Lane's SpawnObstacles so that it includes the different types. Added the images to the assets file.	GameScreen Lane	CON_ACCESSIBLE
Durability	Checks if the player's durability is less than or equal to 0 in GameScreen and ends the game if it is. Then in DragonBoatGame, a new condition for when durability is less than 0, it will display a screen telling the user that their boat is destroyed.	GameScreen DragonBoat Game	UR_LOSS
Saving the game	A function was added to the GameScreen class to allow the game to save. It works by grabbing all the values for the game variables. This includes everything from the player's health to how many power ups does an opponent hold. This data is then saved in one of three slots. The slots are represented as xml files as they are saved using the library's saving method. This way also allows for better modularity when porting the game to other platforms. The oldest slot is overwritten when all the slots are full. To notify the user of this, the render behaviour had to change to allow the player to pause at any point. This can be seen in the render function and the specific UI actions can be seen in the new functions outlined in the architecture such as pauseMenuInput. An enum was also used to manage the current state of the game.	GameScreen	UR_SAVE, FR_PAUSE_SCREEN
New Screen	Following from the requirements, an additional screen was needed to allow the player to choose to start a new game or load a previous save. The loading function implemented here retrieves all the saved values, initialises a new game and injects the values in the new game. Extra UI was also added to allow the player to choose the preferred difficulty when starting a new game.	WelcomeScreen	UR_SAVE, FR_TITLE_SCREEN, UR_GAME_DIFFICULTY

As seen above, the product extends to meet the entire brief. All the requirements requested by the customer were met.

File Links:

- Requirements File [1]: https://omar-h-omar.github.io/ENG1-Dragonite-Assessment-2.github.io/docs/deliverables2/Req1.pdf
- Architecture File [2]: https://omar-h-omar.github.io/ENG1-Dragonite-Assessment-2.github.io/docs/deliverables2/Arch1.pdf
- Github Repository: https://github.com/omar-h-omar/ENG1-Dragonite-Assessment-2.github.io