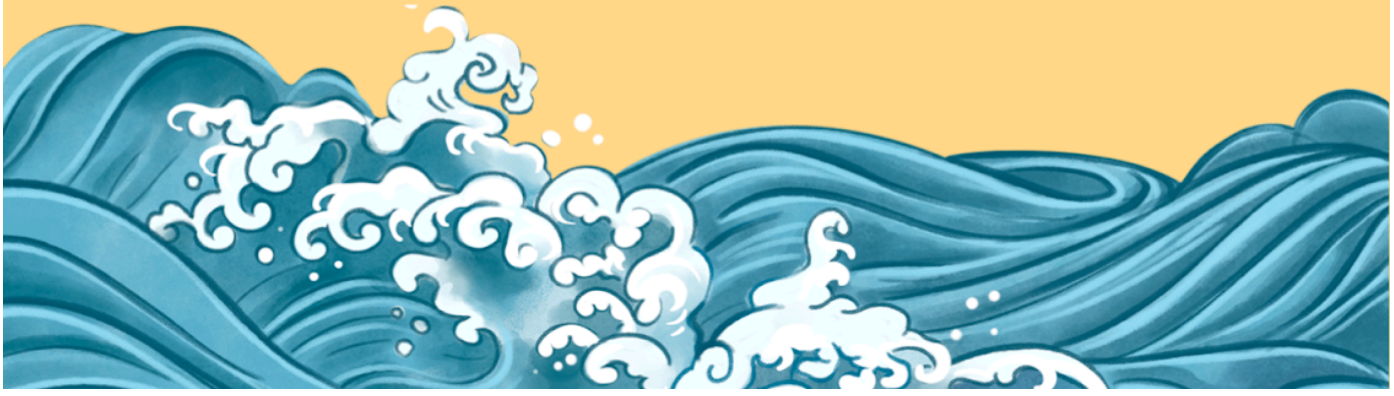
DRAGON BOAT RACE

Jorge Frías Tello - jorgegemanon@gmail.com
Cristina Montañés Peña - cristinamonpe2004@uma.es
Alma García Ramírez - almagarciaramirez@uma.es
Carmen M. Fernández Ferrer - carmenmariafernandez@uma.es
Antonio Manceras Gámez - amanceragamez@gmail.com
Gabriel del Corral Velasco - gabidcor@uma.es
Bianca Valentina Dinu - biancadinu10@gmail.com
Jean-François Senécal - jeanjeansenecal@gmail.com
Nicolás López - n.lopez9418715@gmail.com

https://github.com/jorgefriast/UMA-ISE24-E5



* The text marked like this are the new additions to the submission

* The text marked with blue are the modifications of the final submission

1. INTRODUCTION

Our project, "Dragon Boat Race," aims to simulate traditional dragon boat racing in a single-player gaming experience. Inspired by the historical and cultural significance of this ancient sport, our team is working to develop a software system that provides an engaging and realistic simulation.

Through careful planning and design, we aim to create a game that accurately portrays the challenges and dynamics of dragon boat racing. From managing boat specifications to navigating obstacles and fatigue, our game will offer players an immersive experience as they compete for victory on the virtual river.

2. ROLES

UI/UX Designer & Graphics Artist:

This role involves crafting the visual experience of the game. This includes designing the user interface (UI) to ensure it's intuitive and visually appealing and creating graphical assets such as character designs, backgrounds, and animations to bring the game to life.

Carmen Maria Fernandez Ferrer and Cristina Montañés Peña will be in charge of carrying out this task.

QA Tester:

The members with this role will be responsible for ensuring the quality and integrity of the game. As a QA tester, they'll thoroughly test the game, identifying and reporting any bugs or issues that may arise.

Jorge Frías Tello and Bianca Dinu will be responsible for this task.

Coders:

Their primary responsibility will be developing the game using Java programming language. This includes implementing game mechanics, integrating various components, and ensuring smooth functionality across different platforms.

Gabriel Del Corral Velasco, Antonio Mancera Gamez, Jean-François Senécal and Nicolás López will be responsible for this task.

Documentation Specialist:

They'll be tasked with writing all necessary documentation for the project. This role is crucial for maintaining clear communication and ensuring the project's progress is well-documented and accessible to all stakeholders.

Alma García Ramírez will be responsible for this task.

3. RISK MANAGEMENT

RISK	TYPE	DESCRIPTION	PROBA-BILITY	EFFECTS	MITIGATION STRATEGY
Technical Risks	Technical	Potential challenges with implementing complex game mechanics, integrating different software components, or ensuring compatibility across platforms.	Moderate	Serious	Regular code reviews, continuous integration, and thorough testing to identify and address technical issues early. Additionally, maintaining open communication among team members to quickly address any technical challenges that arise.
Schedule Risks	Project Management	Delays in development, unexpected setbacks, or changes in requirements that impact the project timeline.	High	Serious	Regular monitoring of project progress, frequent updates to stakeholders, and agile development methodologies to adapt to changes efficiently. Establishing contingency plans and allocating buffer time for unforeseen delays.
Commu-nication Risks	Team Dynamics	Miscommunication, conflicts among team members, or breakdowns in collaboration that hinder project progress.	Moderate	Serious	Establishing clear channels of communication, setting expectations for team interactions, and promoting open dialogue among team members. Implementing regular team meetings, progress updates, and conflict resolution mechanisms to address any communication issues promptly.
Quality Risks	Product Quality	Potential issues with product quality, such as bugs, glitches, or user experience shortcomings, that affect the game's usability and appeal.	High	Serious	Implementing rigorous testing procedures, including unit testing, integration testing, and user acceptance testing, to identify and address quality issues early. Soliciting feedback from stakeholders and end-users throughout the development process to ensure that the product meets quality standards and user expectations.

4. PLANNING

EVERYTHING FROM THIS TEXT WAS UPDATED FOR PROJECT SUBMISSION 2, AND THE GRAPHICS WERE ADDED FOR SUBMISSION 3

Planning and Preparation Phase (Week 1-2):

Define project scope and objectives based on the requirements provided by the instructor.

Assign roles and responsibilities to team members, considering their strengths and interests.

Conduct initial meetings to discuss project requirements and establish a shared understanding of the goals.

Create a backlog of tasks and features to be implemented, outlining the prioritization based on project deliverables and deadlines in Trello.

Set up communication channels such as WhatsApp for informal communication and coordination.

Hold regular check-ins or meetings to ensure everyone is on track and address any questions or concerns.

Documentation and Design Phase (Week 2-7):

Develop use cases, class diagrams, sequence diagrams, and other necessary documentation as outlined by the project requirements.

Conduct regular meetings to review and refine the documentation, incorporating feedback from the instructor or peers if necessary.

Utilize tools like Google Docs for collaborative document creation and sharing.

Begin designing the game interface and graphics, considering the visual appeal and user experience.

Development and Testing Phase (Week 7-11/12):

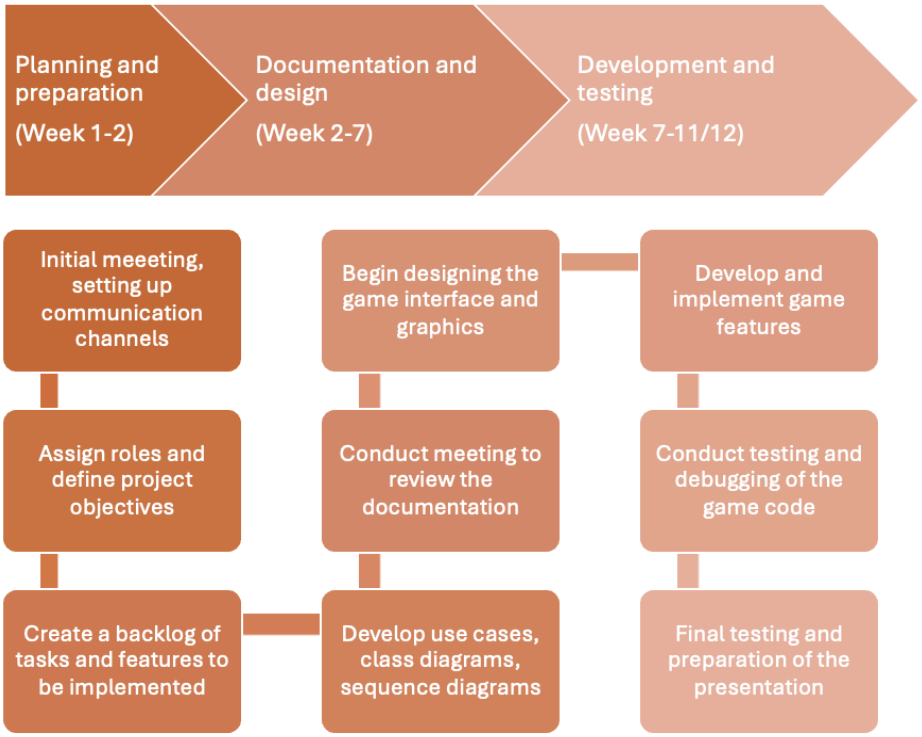
Transition into the development phase, focusing on implementing the game features and functionalities according to the documented requirements.

Break down development tasks into manageable chunks, considering the skills and experience level of team members.

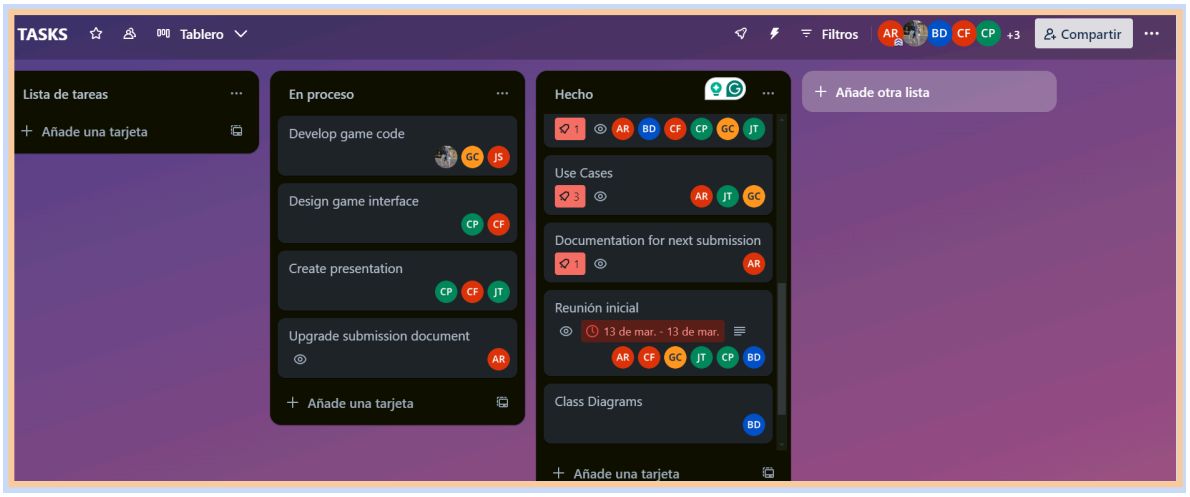
Conduct testing and debugging of the game code, identifying and addressing any issues or bugs.

Allocate time for final testing and quality assurance before submitting the project for evaluation.

Prepare a demonstration or presentation of the completed game for the instructor and peers, showcasing its functionality and features.



Throughout the project, we integrated Scrum practices to manage development effectively. This included conducting Sprint Planning meetings to prioritize tasks from the backlog using tools like Trello. We broke down tasks into manageable chunks and held stand-up meetings to discuss progress, address obstacles, and plan tasks. At the end of each sprint, we conducted Sprint Review meetings to demonstrate completed work and gather feedback. Embracing an iterative approach, we made adjustments to plans and strategies as needed, prioritizing flexibility and continuous improvement to optimize workflow and productivity. Overall, implementing Scrum facilitated effective collaboration, priority management, and the delivery of a high-quality product meeting stakeholder needs.



5. SOFTWARE TOOLS

Throughout the development of "Dragon Boat Race," our team utilized a variety of software tools to facilitate communication, collaboration, and project management. Below is a list of the key tools we employed:

Google Docs: It served as our primary platform for collaborative document creation and editing because it allowed team members to work together in real time on project documentation, such as meeting agendas, minutes, and progress reports.

Microsoft Word: It was used for creating formal project documents due to its robust formatting features and compatibility with other Microsoft Office applications. This ensured the production of professional-quality documents such as the project proposal, requirements specification, and risk management plan, meeting the standards expected for academic and professional presentations.

WhatsApp: WhatsApp served as our main communication tool for quick and informal exchanges among team members. We utilized group chats to share updates, discuss project-related matters, and coordinate tasks in real time, enhancing team collaboration and responsiveness.

Gmail: Gmail was our primary email platform for formal communications with stakeholders, including the project instructor and university representatives. It facilitated seamless communication, file sharing, and scheduling of meetings, ensuring effective and efficient correspondence throughout the project lifecycle.

Trello: Trello was used as our project management tool for organizing tasks, tracking progress, and managing project workflows. Its visual interface provided an overview of project status, allowing us to prioritize tasks, assign responsibilities, and monitor deadlines effectively. Trello's flexibility and customization options adapted well to our agile development approach, facilitating task organization and team coordination.

Visual Paradigm: It was used for creating the requirements diagram, aiding in the visualization and documentation of the relationships between project requirements. Its features enabled us to communicate project scope and objectives effectively, facilitating understanding and alignment among team members and stakeholders.

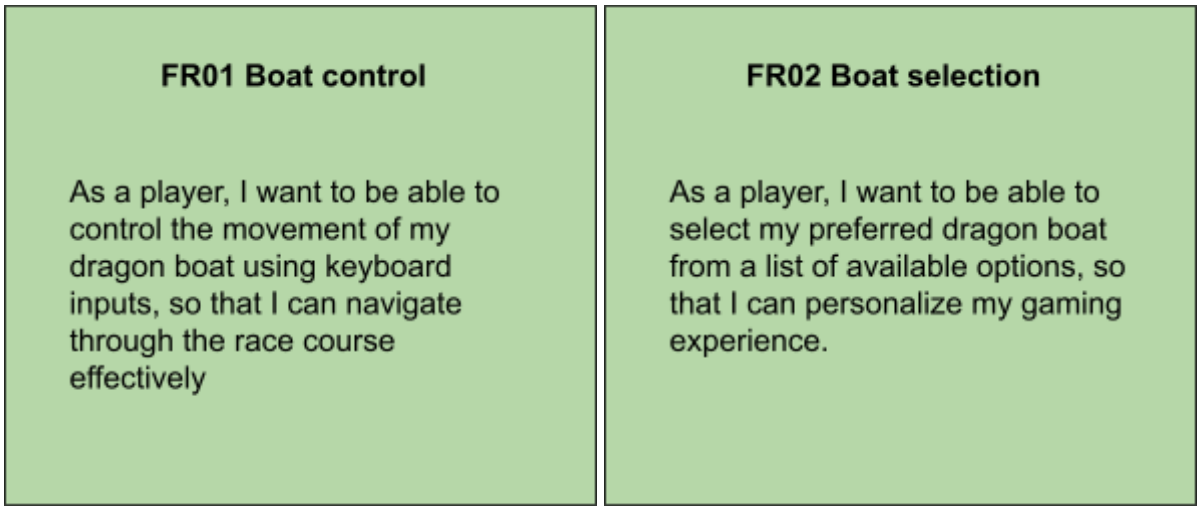
JUnit: It was required to test the behavior of classes and methods. Its use ensured the reliability and correctness of the codebase by automating unit tests, identifying defects early in the development process, and supporting code refactoring and maintenance efforts.

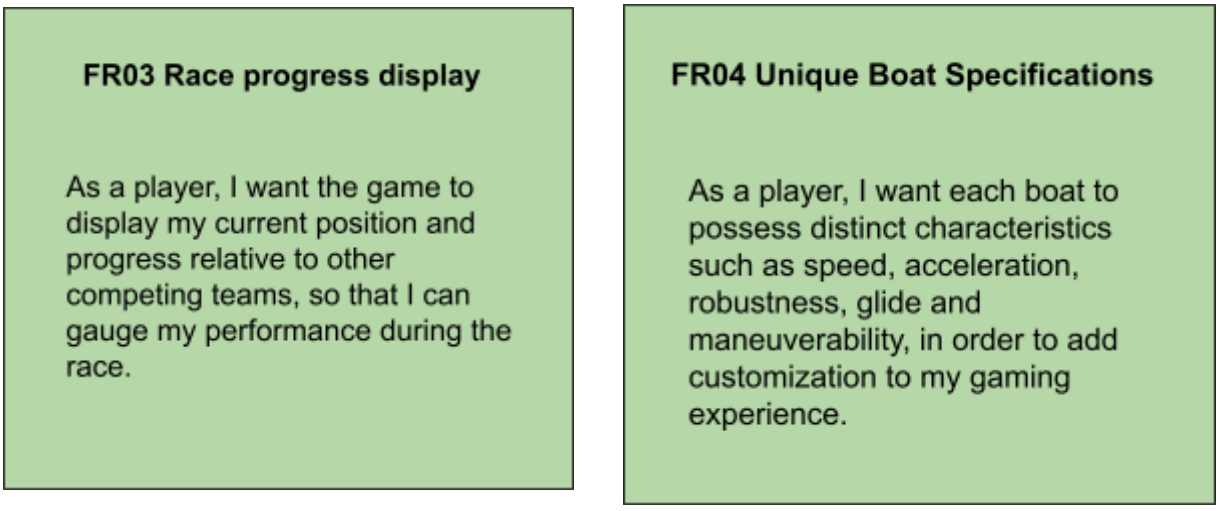
Git: Git was used as a version control tool to organize project files in a common repository. Its features allow for the management of code changes across different stages of the project, enabling collaboration among team members, tracking of project history, and ensuring code integrity and stability.

IntelliJ: IntelliJ (or any preferred Integrated Development Environment) served as an example of an IDE used by team members. While specific IDE choices varied among team members based on individual preferences, the use of IntelliJ highlights the importance of providing developers with tools they are comfortable and productive with, enhancing coding efficiency and quality.

6. REQUIREMENTS

a. FUNCTIONAL REQUIREMENTS (FRs)





FR05 Level of fatigue display

As a player, I want to see the level of fatigue my boat has all along the race in order to manage my race better and so that I have a more realistic gaming experience.

FR06 Encountering obstacles

As a player, I expect to encounter during the race static and dynamic obstacles like tree branches, rocks or even animals which will slow down my boat or reduce its robustness, in order to add challenge to my gaming experience.

FR07 Mini game integration

As a player, I want to play mini games that would help me me gain coins, in order to have additional entertainment and that would help me progress in the game.

FR08 In-game money

As a player, I want to be able to upgrade or buy new boats with the coins I won in the mini game, in order to add to my experience a management of my ressources and a motive to continue playing.

FR09 Power-Up activation

As a player, I want the ability to activate collected power-ups during the race.

FR10 Obstacle variety

As a player, I want to encounter a variety of obstacles during the race.

FR11 Obstacle effects

As a player, I anticipate different effects from encountering obstacles.

FR12 Boat unlock system

As a player, I want to be able to buy new boats using in-game money

FR13 Lane movement

As a player, I expect my boat be able to enter to other lanes

FR14 Progressive difficult levels

As a player, I expect subsequent race legs to increase in difficulty level.

FR15 Real-time stats display

As a player, I want to monitor my boat's state during the race

FR16 Tutorial mode

As a player, I want to a tutorial mode to learn game mechanics as a new player.

FR17 Different controls

As a player, I want to be able to use different keyboard inputs for boat control.

FR18 Easy of use

As a player, I expect simple and easy-to-understand controls for steering, accelerating the dragon boat, enabling me to focus on the race without getting overwhelmed by complex control schemes.

FR19 Randomized Obstacle Placement

As a player, I expect the game to randomly position obstacles within each race segment, ensuring that the arrangement of obstacles varies each time I play the game.

FR20 Obstacle interaction variety

As a player, I expect each obstacle to present different consequences upon encountering them.

FR21 Obstacle: Slowness effect

As a player, I expect some obstacle to slow the movement of the boat, in order to make rewarding avoid them

FR22 Obstacle: endurance effect

As a player, I expect obstacles to reduce the endurance of the boat during the rance, making the possibility to get my boat destroyed if I hit many of them

FR23 Static Obstacles

As a player, I expect some obstacles to be static

FR24 Moving obstacles

As a player, I expect some obstacles to move. This will make the game more challenging, as they are more difficult to avoid

FR25 Shop

As a player, I expect to have a shop in the menus of the game where I can buy power-ups, in order to have advantages on the game

FR26 Opponent lane

As a player, I want to be able to enter in the opponent lanes if I need to avoid obstacles

FR25 Obstacles: Opponents

As a player, I expect to be able to get hit by opponent's boat. Both boats should suffer damages.

FR26 Reward

As a player, I want to receive a medal or a trophy if a finish all the races, to recognize that I have won the game

b. NON-FUNCTIONAL REQUIREMENTS (NFRs)

NFR01 Intuitive navigation

As a player, I want the game interface to be intuitive and easy to navigate.

NFR02 User interface responiveness

As a player, I want the software to provide responsive user interface interactions and smooth transitions, to enhance user satisfaction and usability.

NFR03 Minimal load times

As a player, I want minimal load times between race segments.

NFR04 Save game state

As a player, I want to be able to return to the game later and have the state of my game (coins, acquired boats) saved.

NFR05 Engaging sound effects

As a player, I want engaging sound effects to enhance immersion.

NFR06 Scability

As a player, I want the game architecture to allow for scalability to accommodate future updates.

NFR07 Secure data storage

As a player, I want player data to be securely stored to prevent loss or unauthorized access.

NFR08 Graphics performance optimization

As a player, I want to adjust graphics settings for optimal performance on various devices.

NFR9 Obstacle visibility

As a player, I expect obstacles to stand out prominently against the background, using contrasting colors, shapes, and textures to ensure recognition and differentiation.

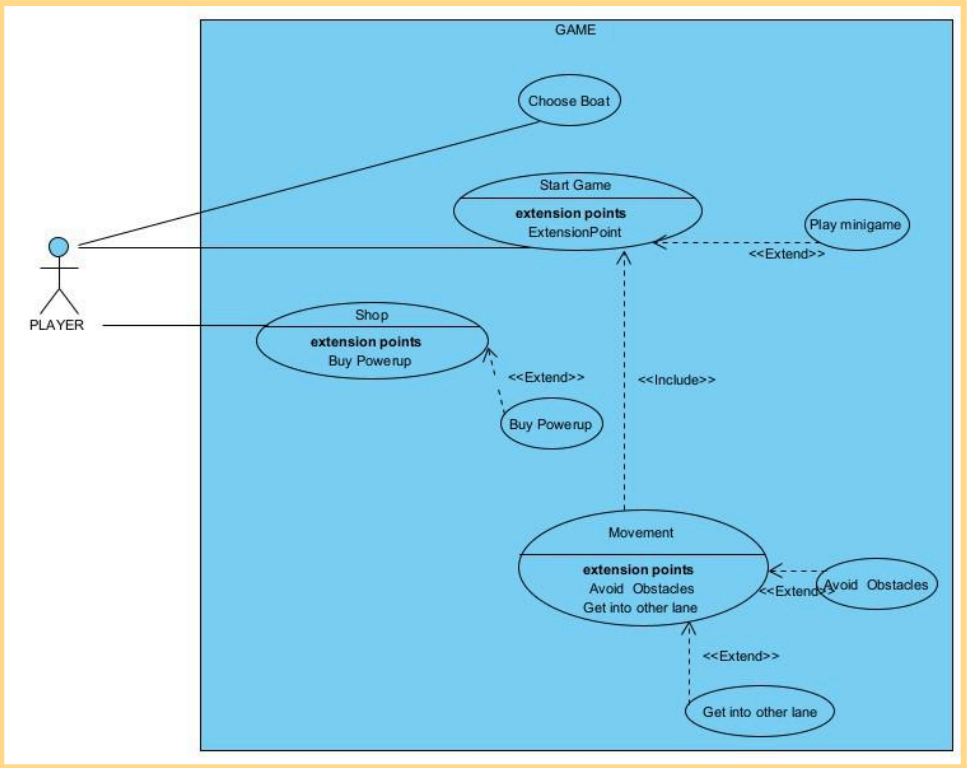
NFR10 Crashes

As a player, I expect the game to not crash due to a bug in the design

7. REQUIREMENTS DIAGRAM

For a more detailed view, see the link to the .vpp on Github.

8. USE CASES

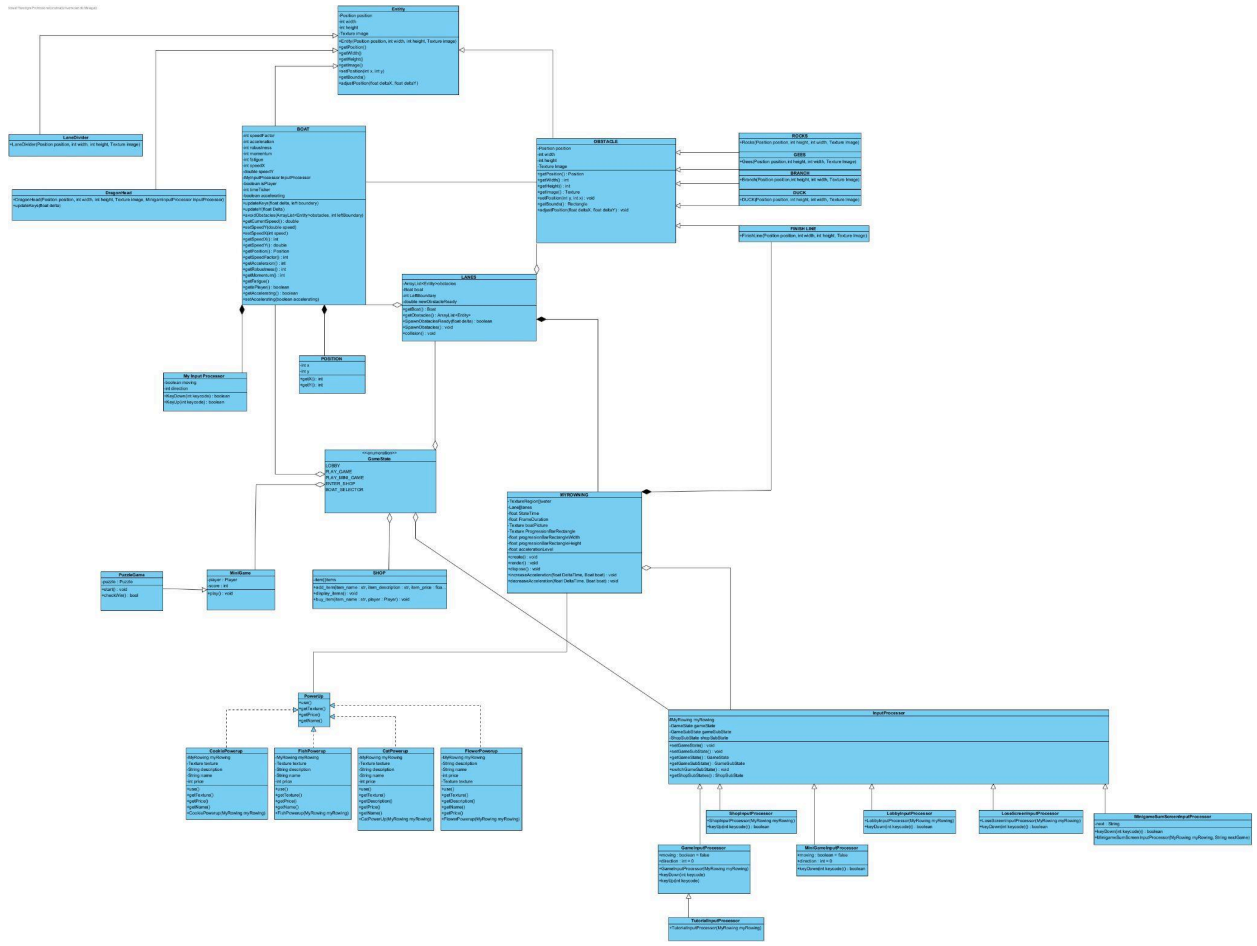


Unique Identifier	Context of Use	Preconditions and Activation	Success Guarantees	Main Scenario	Alternative Scenarios
UC1. Choose Boat	A player can select a dragon boat to compete in the race	The game is in the boat selection menu and the player is on the boat selection screen.	The player has selected a dragon boat to compete in the race.	The player navigates through the available boats list. The player selects a dragon boat. The player confirms their selection.	If the player cancels the boat selection, they return to the main menu.
UC2. Start Game	The player will be able to start the race.	The player has selected a dragon boat and the game is on the race start screen.	The player has initiated the race with the selected boat.	The player presses the button to start the race. The race begins with the selected boat.	If the player decides not to start the race, they return to the boat selection screen.
UC3. Shop	The player wants to purchase upgrades for their boat.	The player is in the shop screen and the player has coins or points to spend.	The player has purchased the selected upgrades for their boat.	The player selects a specific powerup. The player confirms the purchase of the powerup.	If the player cancels the purchase, they remain in the shop screen without buying the powerup.

UC 4. Buy Powerup	The player will buy a specific powerup for their boat.	The player is in the shop. The player has coins or points to spend.	The player has purchased the selected powerup for their boat that they can use in the race.	The player selects a specific powerup. The player confirms the purchase of the powerup.	If the player cancels the purchase, they remain in the shop screen without buying the powerup.
UC 5. Play Minigame	The player has finished a leg and will play a minigame.	The player has finished a leg.	The player completes the minigame and gains points or advantages for the main race.	The player plays the minigame and wins points/coins.	If the player fails to complete the minigame they continue to the next leg.
UC 6. Movement	The player needs to control the movement of the boat during the race.	The game is in progress. The player is playing a leg..	The player successfully navigates the boat.	The player controls the direction boat using controls to stay within its own lane and avoid obstacles.	If the player steers the boat out of the designated lane or crashes with an obstacle, penalties are applied.
UC 7. Avoid Obstacles	The player is competing in the race and needs to dodge to avoid obstacles.	The game is in progress. The player is controlling the boat.	The player has successfully avoided an obstacle.	The player moves the boat to avoid obstacles and navigates successfully without colliding.	If the player collides with an obstacle, the boat's robustness decreases, and the player could fail the race.
UC 8. Get Into Other Lane	The player decides to move the boat into another lane during the race.	The game is in progress. The player is actively racing in the river. The player intends to switch lanes.	The dragon boat remains in its designated lane without moving into another lane to avoid an obstacle	The player changes lanes trying to avoid an obstacle. It's robustness decreases	The player avoids the obstacle successfully without having to change lanes

We have fixed the movement/start game relation

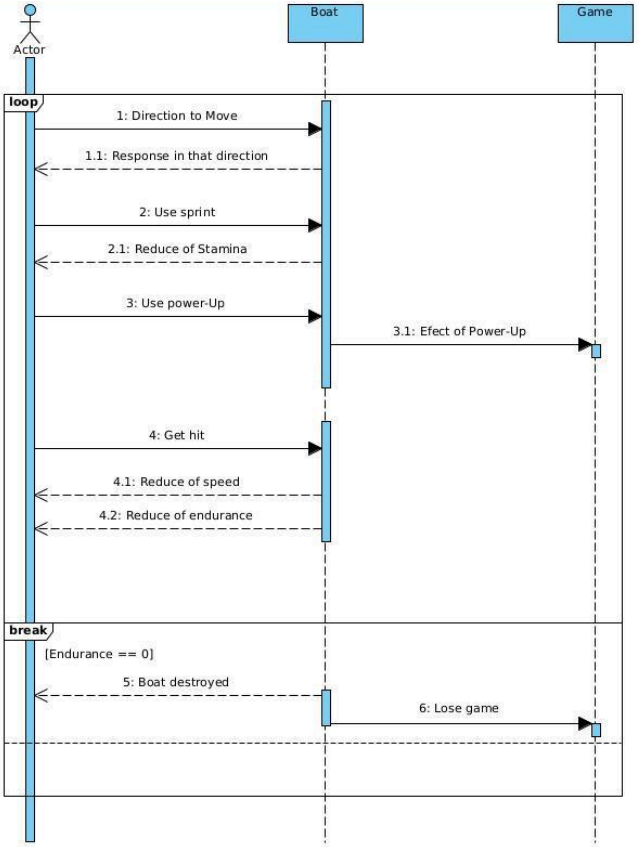
9. DOMAIN MODEL



(To see in a better way, the image is uploaded to the team's Github)
https://github.com/jorgefriast/UMA-ISE24-E5/blob/33c180f1372238b15f0a25f44b0215b8ad34b9d5/Class%20Diagram.jpg

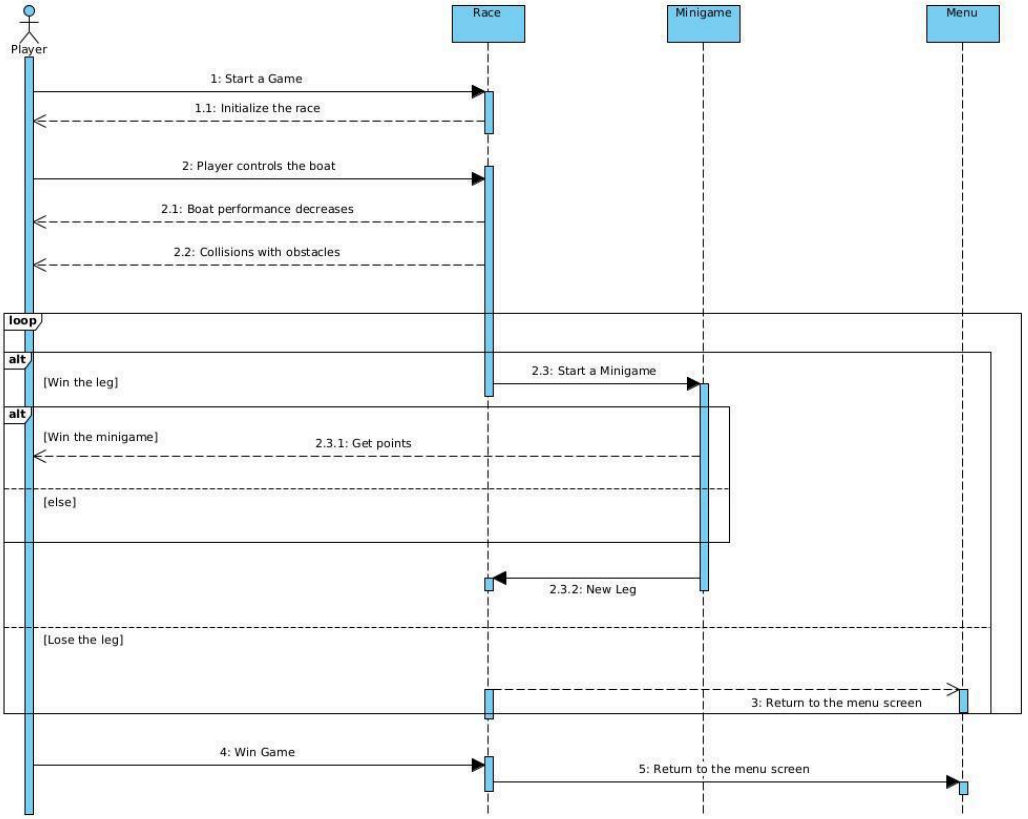
10. SEQUENCE DIAGRAMS

Boat Control:



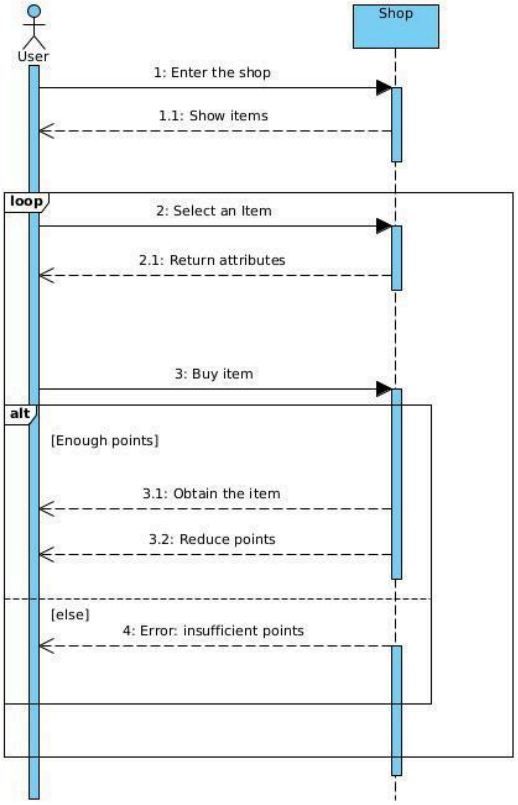
A loop is created where the player chooses the direction in which they want to move the boat, and the boat responds by moving in that direction. When the player uses a sprint, the boat's stamina decreases. If the player uses a power-up, the game responds with the effect of that power-up. If the player gets hit, there is a reduction in the boat's speed and endurance. Finally, if the endurance reaches 0, the loop exits and the boat is destroyed, resulting in a game over.

Races:



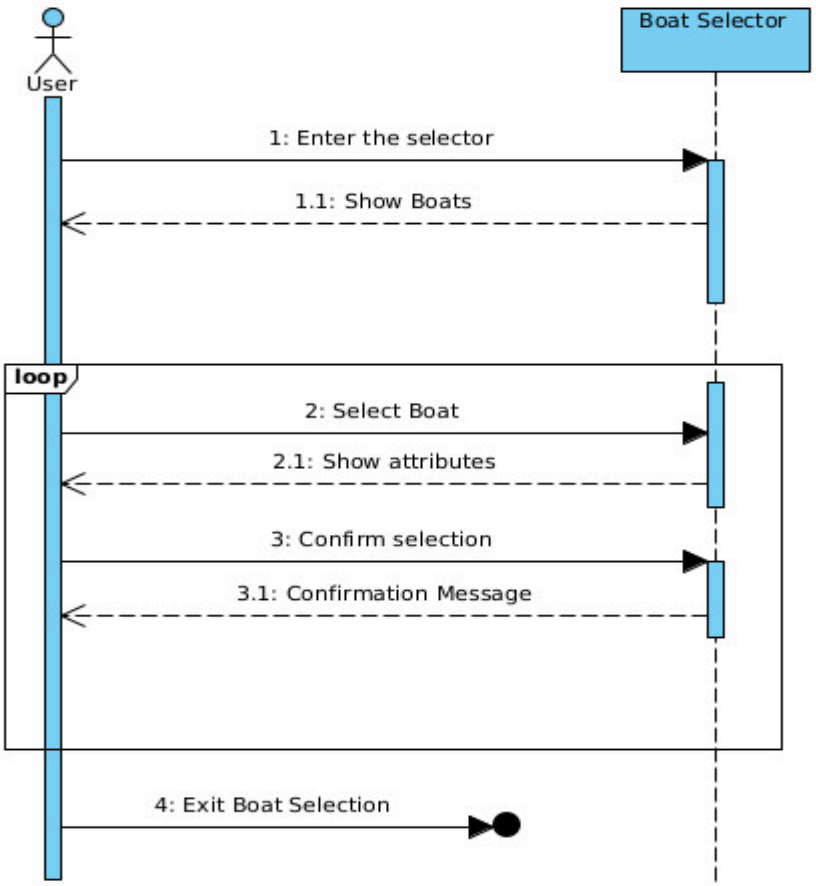
The player begins the game, and the race starts. While racing, the player controls the boat. Boat performance decreases as the game progresses, and collisions with obstacles affect the player. If the player wins the leg, a mini-game starts, where they can earn points. Failing to score points starts a new leg. If the player loses that leg, they return to the home screen.

Shop:



The player begins the game, and the race starts. While racing, the player controls the boat. Boat performance decreases as the game progresses, and collisions with obstacles affect the player. If the player wins the leg, a mini-game starts, where they can earn points. Failing to score points starts a new leg. If the player loses that leg, they return to the home screen.

Boat Selection



The player has the option to enter the selector of boats. Here, the selector would show the possible boats and the user will be able to select one of the boats, see its attributes and decide if to select that boat as the one he/she will use during the next games.

11. TEST CASES

Unit Test Documentation Template

Project Name:
Dragon Boat Race

Author(s):
Jorge Frías Tello
Bianca Valentina Dinu

Test Case Overview

Test Case ID:
TC_001
FR01

Purpose:
Verify that the boat moves correctly to the left when the left key is pressed

Test Case Description:
This case will verify that the boat moves horizontally to the left when the player clicks the left key

Pre-Conditions

Prerequisites:
The development environment must be configured with JUnit
The Mockito dependency must be included in the project
Class Boat and its dependencies should compile without errors

Test Data:
Initial Position: (100, 100)
Texture mocked
Constructor parameters:
IsPlayer: true
SpeedFactor: 5
Acceleration: 3
Robustness: 4
Maneuverability: 2
MomentumFactor: 2
Fatigue: 1
InputProcessor: Instance ofGameInputProcessor

Test Steps

Step Description:
Mock the necessary dependencies (Texture and Position)
Create an instance of Boat with the specified test data
Configure the InputProcessor to simulate left arrow key pressed
Called the method updateKeys and verify that the position x of the sip is reduced

Post-Conditions

Expected Outcome:
*The position x of the ship should be reduced after running updateKeys with the left arrow key pressed. The boat's X-coordinate should increase by speedX * 2 pixels.*

Cleanup:
Ensure any textures or resources used in the test are disposed of after the test.

Notes

This test doesn’t depend on interaction with other external systems or services

Test Case Overview

Test Case ID:
TC_002
FR11

Purpose:
Verify that the method damageBoat from the class Boat correctly reduces ship’s health when taking damage

Test Case Description:
This test case will verify that the method damageBoat reduces the ship's health according to the damage received.

Pre-Conditions

Prerequisites:
The development environment must be configured with JUnit
The Mockito dependency must be included in the project
Class Boat and its dependencies should compile without errors

Test Data:
Initial Position: (100, 100)
Texture mocked
Constructor parameters:
IsPlayer: true
SpeedFactor: 5
Acceleration: 3
Robustness: 4
Maneuverability: 2
MomentumFactor: 2
Fatigue: 1
InputProcessor: Instance ofGameInputProcessor
Damage receives: 10

Test Steps

Step Description:
Mock the necessary dependencies (Texture and Position)
Create an instance of Boat with the specified test data
Configure the damageBoat with the specified damage value
Verify that the ship’s health has been reduced correctly

Post-Conditions

Expected Outcome:
The ship's health must be reduced by the value of the damage received

Cleanup:
Ensure any textures or resources used in the test are disposed of after the test.

Notes

This test doesn’t depend on interaction with other external systems or services

Test Case Overview

Test Case ID:
TC_003
FR01

Purpose:
Verify that the method getPosition from the class Entity returns the correct position of the entity.

Test Case Description:
This test case will verify that the method getPosition returns the correct position of the entity.

Pre-Conditions

Prerequisites:
The development environment must be configured with JUnit
Class Entity and its dependencies should compile without errors

Test Data:
Position: (50, 50)
Texture mocked

Test Steps

Step Description:
Mock the necessary dependencies (Texture and Position)
Create an instance of Entity with the specified test data
Call the method getPosition
Verify that the returned position is (50,50)

Post-Conditions

Expected Outcome:
The method getPosition must return position(50, 50).

Cleanup:
Ensure any textures or resources used in the test are disposed of after the test.

Notes

This test doesn't depend on interaction with other external systems or services

Test Case Overview

Test Case ID:
TC_004
FR01

Purpose:
Verify that the method getPosition returns the correct position of the entity.

Test Case Description:
This test case will verify that the method setPosition returns the correct position of the entity.

Pre-Conditions

Prerequisites:
The development environment must be configured with JUnit
Class Entity and its dependencies should compile without errors

Test Data:
Position: (50, 50)
Texture mocked
New position: (100, 150)

Test Steps

Step Description:
Mock the necessary dependencies (Texture and Position)
Create an instance of Entity with the specified test data
Call the method setPosition with the values (100, 150)
Verify that the entity's position has been updated to (100, 150)

Post-Conditions

Expected Outcome:
The method setPosition must update the position of the entity to (100, 150).

Cleanup:
Ensure any textures or resources used in the test are disposed of after the test.

Notes

This test doesn't depend on interaction with other external systems or services

Test Case Overview

Test Case ID:
TC_005
FR13

Purpose:
Verify that the method getBoat from the class Lane returns the correct position of the entity.

Test Case Description:
This test case will verify that the methodgetBoat returns the correct boat from the track.

Pre-Conditions

Prerequisites:
The development environment must be configured with JUnit
Class Lane and its dependencies should compile without errors

Test Data:
Boat mocked with initial position(100, 100)
leftBoundary: 50

Test Steps

Step Description:
Mock the necessary dependencies (Boat)
Create an instance of Lane with the mocked boat and leftBoundary of 50
Call the method getBoat
Verify that the returned boat is the mocked one.

Post-Conditions

Expected Outcome:
The method getBoat must return the mocked boat.

Cleanup:
Ensure any textures or resources used in the test are disposed of after the test.

Notes

This test doesn't depend on interaction with other external systems or services

Test Case Overview

Test Case ID:
TC_006
FR06

Purpose:
Verify that the method spawnObstacleReady returns true when a new obstacle must be generated.

Test Case Description:
This test case will verify that the method spawnObstacleReady returns true when the accumulated time exceeds the threshold.

Pre-Conditions

Prerequisites:
The development environment must be configured with JUnit
Class Lane and its dependencies should compile without errors

Test Data:
Boat mocked with initial position(100, 100)
leftBoundary: 50
delta: 0.5

Test Steps

Step Description:
Mock the necessary dependencies (Boat)
Create an instance of Lane with the mocked boat and leftBoundary of 50
Call the method spawnObstacleReady several times with delta of0.5 until I return true.
Verify that the method returns true after enough calls.

Post-Conditions

Expected Outcome:
The method spawnObstacleReady must return true when the accumulated time exceeds the threshold.

Cleanup:
Ensure any textures or resources used in the test are disposed of after the test.

Notes

This test doesn't depend on interaction with other external systems or services

Test Case Overview

Test Case ID:
TC_007
FR20

Purpose:
Verify that the method collision successfully removes an obstacle when it collides with the boat.

Test Case Description:
This test case will verify that the method collision from the class Lane removes an obstacle from the list when it collides with the boat and adjusts the position of the boat.

Pre-Conditions

Prerequisites:
The development environment must be configured with JUnit
Class Lane and its dependencies should compile without errors

Test Data:
Boat mocked with initial position(100, 100)
leftBoundary: 50
Obstacle mocked with starting position(100, 100)

Test Steps

Step Description:
Mock the necessary dependencies (Boat and Obstacle)
Create an instance of Lane with the mocked boat and leftBoundary of 50
Add the mocked obstacle to the list of obstacles
Call the method collision
Verify that the obstacle has been removed from the list and the boat's position has been adjusted correctly.

Post-Conditions

Expected Outcome:
The method collision must remove the obstacle from the list and adjust the position of the boat.

Cleanup:
Ensure any textures or resources used in the test are disposed of after the test.

Notes

This test doesn’t depend on interaction with other external systems or services

Test Case Overview

Test Case ID:
TC_008

Purpose:
Verify that the createNewGame method correctly initializes all game components, including creating all instances of Lane and Boat and their initial configuration.

Test Case Description:
This test case ensures that the createNewGame method creates and initializes all instances of Lane and Boat correctly and sets their initial values as expected.

Pre-Conditions

Prerequisites:
The development environment must be configured with JUnit
Class MyRowing and its dependencies should compile without errors

Test Data:
No specific data is required

Test Steps

Step Description:
Create an instance of MyRowing.
Call the createNewGame method.
Verify that all instances of Lane are created and are not null.
Verify that each Lane contains an instance of Boat that is not null.
Verify that the initial positions and other attributes of the boats are correct according to the game's initial setup.

Post-Conditions

Expected Outcome:
All instances of Lane and Boat are created correctly.
The initial positions and other attributes of the boats are set as expected.

Cleanup:
Ensure any textures or resources used in the test are disposed of after the test.

Notes

This test doesn’t depend on interaction with other external systems or services

Test Case Overview

Test Case ID:
TC_009

Purpose:
Verify that the finishLine method correctly draws the finish line on the screen and moves it downward.

Test Case Description:
This test case ensures that the finishLine method correctly draws the finish line at the correct position and moves it down the screen as the game progresses.

Pre-Conditions

Prerequisites:
The development environment must be configured with JUnit
Class MyRowing and its dependencies should compile without errors

Test Data:
No specific data is required

Test Steps

Step Description:
Create an instance of MyRowing.
Call the createNewGame method.
Call the finishLine method.
Verify that the finish line is drawn at the initial position.
Call the finishLine method multiple times to simulate the passage of time.
Verify that the finish line moves downward with each call.

Post-Conditions

Expected Outcome:
The finish line is initially drawn at the correct position.
The finish line moves downward each time the finishLine method is called.

Cleanup:
Ensure any textures or resources used in the test are disposed of after the test.

Notes

This test doesn't depend on interaction with other external systems or services

12. TEST IMPLEMENTATION

Boat Test Class

```
package com.introduction.rowing;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

public class BoatTest {

    private Boat boat;
    private GameInputProcessor inputProcessor;
    private ShopBoat shopBoat;

    @BeforeEach
    public void setUp() {
        // Set up necessary objects for the test
        Position position = new Position(0, 0);
        MyRowing myRowing = new MyRowing(); // Make sure you have appropriate constructor or mock
        inputProcessor = new GameInputProcessor(myRowing);
        shopBoat = new ShopBoat(1, "Standard Boat", 1000, "boat.png", 10, 10, 10, 10, 10, 10, true, false);
        boat = new Boat(1, position, true, inputProcessor, shopBoat);


    }

    @Test
    public void testBoatInitialization() {
        // Verify initialization of Boat attributes
        assertEquals(1, boat.getId());
        assertEquals(10, boat.getSpeedFactor());
        assertEquals(10, boat.getAcceleration());
        assertEquals(10, boat.getRobustness());
        assertEquals(10, boat.getMomentumFactor());
        assertEquals(10, boat.getFatigue());
    }

    @Test
    public void testUpdateKeysMovingUp() {
        inputProcessor.moving = true;
        inputProcessor.direction = 0;
        boat.setIsAcceleratorAvailable(true);

        float initialY = boat.getPosition().getY();
```

```
        boat.updateKeys(1.0f, 0);

        assertTrue(boat.getPosition().getY() > initialY);
    } }
```

CatPowerUp Test Class

```
package com.introduction.rowing;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.mockito.Mockito.mock;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import com.badlogic.gdx.graphics.Texture;

public class CatPowerupTest {

    private MyRowing mockedMyRowing;
    private CatPowerup catPowerup;

    @BeforeEach
    public void setUp() {
        // Mock MyRowing
        mockedMyRowing = mock(MyRowing.class);
        catPowerup = new CatPowerup(mockedMyRowing);
    }

    @Test
    public void testUse() {
        // Verify that use() method correctly sets invulnerability time of player boat
        catPowerup.use();
        assertEquals(5, mockedMyRowing.getPlayerBoat().getInvulnerabilityTime());
    }

    @Test
    public void testGetTexture() {
        // Verify that a non-null texture is returned
        Texture texture = catPowerup.getTexture();
        assertNotNull(texture);
    }
}
```

CoockiePowerUp Test Class

```
package com.introduction.rowing;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.mockito.Mockito.mock;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import com.badlogic.gdx.graphics.Texture;

public class CookiePowerupTest {

    private MyRowing mockedMyRowing;
    private CookiePowerup cookiePowerup;

    @BeforeEach
    public void setUp() {
        // Mock MyRowing
        mockedMyRowing = mock(MyRowing.class);
        cookiePowerup = new CookiePowerup(mockedMyRowing);
    }

    @Test
    public void testUse() {
        // Verify that use() method correctly increases boat health
        Boat playerBoat = mockedMyRowing.getPlayerBoat();
        int initialHealth = playerBoat.getBoatHealth();
        cookiePowerup.use();
        assertEquals(initialHealth + 25, playerBoat.getBoatHealth());
    }

    @Test
    public void testGetName() {
        // Verify that correct name is returned
        assertEquals("Fortune Cookie", cookiePowerup.getName());
    }

    @Test
    public void testGetPrice() {
        // Verify that correct price is returned
```

```
        assertEquals(150, cookiePowerup.getPrice());
    }
}
```

Datamanagert Test Class

```
package com.introduction.rowing;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.mockito.Mockito.mock;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.files.FileHandle;

public class DataManagerTest {

    private DataManager dataManager;

    @BeforeEach
    public void setUp() {
        dataManager = new DataManager();
    }

    @Test
    public void testReadBalance() {
        // Verify that balance is correctly read from the file
        int expectedBalance = 0;
        FileHandle fileHandle = Gdx.files.local(DataManager.MONEY_BALANCE_FILE_PATH);
        if (fileHandle.exists()) {
            expectedBalance = Integer.parseInt(fileHandle.readString());
        }
        assertEquals(expectedBalance, dataManager.getBalance());
    }
}
```

Datamanagert Test Class

```
package com.introduction.rowing;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.mockito.Mockito.mock;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.files.FileHandle;

public class DataManagerTest {

    private DataManager dataManager;

    @BeforeEach
    public void setUp() {
        dataManager = new DataManager();
    }

    @Test
    public void testReadBalance() {
        // Verify that balance is correctly read from the file
        int expectedBalance = 0;
        FileHandle fileHandle = Gdx.files.local(DataManager.MONEY_BALANCE_FILE_PATH);
        if (fileHandle.exists()) {
            expectedBalance = Integer.parseInt(fileHandle.readString());
        }
        assertEquals(expectedBalance, dataManager.getBalance());
    }
}
```

Dragonhead Test Class

```
package com.introduction.rowing;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.mock;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import com.badlogic.gdx.graphics.Texture;

public class DragonHeadTest {

    private DragonHead dragonHead;
    private MiniGameInputProcessor mockedInputProcessor;

    @BeforeEach
    public void setUp() {
        Position position = new Position(0, 0);
        Texture texture = mock(Texture.class);
        mockedInputProcessor = mock(MiniGameInputProcessor.class);
        dragonHead = new DragonHead(position, 50, 50, texture, mockedInputProcessor);
    }

    @Test
    public void testUpdateKeysMovingUp() {
        // Verify that the position updates correctly when moving up
        mockedInputProcessor.moving = true;
        mockedInputProcessor.direction = 0;
        float initialY = dragonHead.getPosition().getY();
        dragonHead.updateKeys(1.0f);
        float updatedY = dragonHead.getPosition().getY();
        assertEquals(Math.max(0, initialY + (250 * 1.0f)), updatedY);
    }


    @Test
    public void testUpdateKeysMovingLeft() {
        // Verify that the position updates correctly when moving left
        mockedInputProcessor.moving = true;
        mockedInputProcessor.direction = 1;
        float initialX = dragonHead.getPosition().getX();
```

```
        dragonHead.updateKeys(1.0f);
        float updatedX = dragonHead.getPosition().getX();
        assertEquals(Math.max(0, initialX - (250 * 1.0f)), updatedX);
    }

    @Test
    public void testUpdateKeysMovingDown() {
        // Verify that the position updates correctly when moving down
        mockedInputProcessor.moving = true;
        mockedInputProcessor.direction = 2;
        float initialY = dragonHead.getPosition().getY();
        dragonHead.updateKeys(1.0f);
        float updatedY = dragonHead.getPosition().getY();
        assertEquals(Math.max(0, initialY - (250 * 1.0f)), updatedY);
    }

    @Test
    public void testUpdateKeysMovingRight() {
        // Verify that the position updates correctly when moving right
        mockedInputProcessor.moving = true;
        mockedInputProcessor.direction = 3;
        float initialX = dragonHead.getPosition().getX();
        dragonHead.updateKeys(1.0f);
        float updatedX = dragonHead.getPosition().getX();
        assertEquals(Math.max(0, initialX + (250 * 1.0f)), updatedX);
    }
}
```

Entity Test Class

```
package com.introduction.rowing;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import com.badlogic.gdx.graphics.Texture;

public class EntityTest {

    private Entity entity;
    private Position position;
    private Texture texture;

    @BeforeEach
    public void setUp() {
        position = new Position(0, 0);
        texture = new Texture("test_texture.png");
        entity = new Entity(position, 50, 50, texture);
    }

    @Test
    public void testGetPosition() {
        // Verify that the position of the entity is returned correctly
        assertEquals(position, entity.getPosition());
    }

    @Test
    public void testGetWidth() {
        // Verify that the width of the entity is returned correctly
        assertEquals(50, entity.getWidth());
    }

    @Test
    public void testGetHeight() {
        // Verify that the height of the entity is returned correctly
        assertEquals(50, entity.getHeight());
    }

    @Test
    public void testSetPosition() {
```

```
        // Verify that the position of the entity is set correctly
        int newX = 100;
        int newY = 200;
        entity.setPosition(newX, newY);
        assertEquals(newX, entity.getPosition().getX());
        assertEquals(newY, entity.getPosition().getY()); }}
```

Input Processor Test Class

```
package com.introduction.rowing;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;

import org.junit.Before;
import org.junit.Test;

public class InputProcessorTest {

    private InputProcessor inputProcessor;
    private MyRowing myRowing;

    @Before
    public void setUp() {
        myRowing = new MyRowing();
        inputProcessor = new InputProcessor(myRowing);
    }

    @Test
    public void testSetGameState() {
        InputProcessor.setGameState(GameState.RACING);
        assertEquals(GameState.RACING, InputProcessor.getGameState());

        InputProcessor.setGameState(GameState.LOBBY);
        assertEquals(GameState.LOBBY, InputProcessor.getGameState());
    }

    @Test
    public void testSetGameSubState() {
        InputProcessor.setGameSubState(GameSubState.RACE_END);
        assertEquals(GameSubState.RACE_END, InputProcessor.getGameSubState());

        InputProcessor.setGameSubState(GameSubState.RACE_LEG);
        assertEquals(GameSubState.RACE_LEG, InputProcessor.getGameSubState());
    }

    @Test
```

```
    public void testSwitchGameSubState() {
        InputProcessor.switchGameSubState();
        assertEquals(ShopSubState.POWERUPS, InputProcessor.getShopSubStates());

        InputProcessor.switchGameSubState();
        assertEquals(ShopSubState.BOATS, InputProcessor.getShopSubStates());
    }

    @Test
    public void testInitialStates() {
        assertEquals(GameState.LOBBY, InputProcessor.getGameState());
        assertEquals(GameSubState.RACE_LEG, InputProcessor.getGameSubState());
        assertEquals(ShopSubState.BOATS, InputProcessor.getShopSubStates());
    }

    @Test
    public void testChangeGameStateAndSubState() {
        InputProcessor.setGameState(GameState.RACING);
        InputProcessor.setGameSubState(GameSubState.RACE_END);

        assertNotEquals(GameState.LOBBY, InputProcessor.getGameState());
        assertNotEquals(GameSubState.RACE_LEG, InputProcessor.getGameSubState());

        assertEquals(GameState.RACING, InputProcessor.getGameState());
        assertEquals(GameSubState.RACE_END, InputProcessor.getGameSubState());
    }
}
```

Lane Test Class

```
package com.introduction.rowing;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.mockito.Mockito.mock;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import com.badlogic.gdx.graphics.Texture;

public class LaneTest {

    private Lane lane;
    private Boat boat;

    @BeforeEach
    public void setUp() {
        boat = mock(Boat.class);
        lane = new Lane(boat, 100);
    }

    @Test
    public void testGetBoat() {
        // Verify that the correct boat is returned
        assertEquals(boat, lane.getBoat());
    }

    @Test
    public void testSpawnObstacleReady() {
        // Verify that spawn obstacle ready status is true after a certain time
        assertTrue(lane.spawnObstacleReady(0.9f));
    }


    @Test
    public void testSpawnObstacles() {
        // Verify that obstacles are added to the lane
```

```
        lane.spawnObstacles();
        assertEquals(1, lane.getObstacles().size());
    }

}
```

Fish Power Up Test Class

```
package com.introduction.rowing;

import static org.junit.Assert.assertEquals;
import static org.mockito.Mockito.*;

import com.badlogic.gdx.graphics.Texture;
import org.junit.Before;
import org.junit.Test;
import org.mockito.Mockito;

public class FishPowerupTest {

    private FishPowerup fishPowerup;
    private MyRowing myRowing;

    @Before
    public void setUp() {
        myRowing = Mockito.mock(MyRowing.class);
        fishPowerup = new FishPowerup(myRowing);
    }

    @Test
    public void testUse() {
        fishPowerup.use();
        verify(myRowing).deleteAllPlayerObstacles();
    }

    @Test
    public void testGetDescription() {
        assertEquals("All obstacles in your lane \n disappear", fishPowerup.getDescription());
    }

    @Test
    public void testGetName() {
        assertEquals("Koi", fishPowerup.getName());
    }

    @Test
```

```
    public void testGetPrice() {
        assertEquals(200, fishPowerup.getPrice());
    }

    @Test
    public void testGetTexture() {
        Texture texture = fishPowerup.getTexture();
        assertEquals("powerups/koi.png", texture.toString());
    }
```

Mini Game Input Processor Test class

```
import org.junit.Before;
import org.junit.Test;
import org.mockito.Mockito;

import static org.junit.Assert.*;
import static org.mockito.Mockito.*;

public class MiniGameInputProcessorTest {

    private MiniGameInputProcessor miniGameInputProcessor;
    private MyRowing myRowing;

    @Before
    public void setUp() {
        myRowing = Mockito.mock(MyRowing.class);
        miniGameInputProcessor = new MiniGameInputProcessor(myRowing);
    }

    @Test
    public void testKeyDownUp() {
        miniGameInputProcessor.keyDown(Input.Keys.UP);
        assertTrue(miniGameInputProcessor.moving);
        assertEquals(0, miniGameInputProcessor.direction);

        miniGameInputProcessor.keyUp(Input.Keys.UP);
        assertFalse(miniGameInputProcessor.moving);
    }

    @Test
    public void testKeyDownLeft() {
        miniGameInputProcessor.keyDown(Input.Keys.LEFT);
        assertTrue(miniGameInputProcessor.moving);
        assertEquals(1, miniGameInputProcessor.direction);

        miniGameInputProcessor.keyUp(Input.Keys.LEFT);
        assertFalse(miniGameInputProcessor.moving);
    }
```

```
    @Test
    public void testKeyDownDown() {
        miniGameInputProcessor.keyDown(Input.Keys.DOWN);
        assertTrue(miniGameInputProcessor.moving);
        assertEquals(2, miniGameInputProcessor.direction);

        miniGameInputProcessor.keyUp(Input.Keys.DOWN);
        assertFalse(miniGameInputProcessor.moving);
    }

    @Test
    public void testKeyDownRight() {
        miniGameInputProcessor.keyDown(Input.Keys.RIGHT);
        assertTrue(miniGameInputProcessor.moving);
        assertEquals(3, miniGameInputProcessor.direction);

        miniGameInputProcessor.keyUp(Input.Keys.RIGHT);
        assertFalse(miniGameInputProcessor.moving);
    }

    @Test
    public void testKeyDownEscape() {
        miniGameInputProcessor.keyDown(Input.Keys.ESCAPE);
        verify(myRowing, times(1)).resetMiniGame();
        assertEquals(MiniGameState.NOT_STARTED, myRowing.miniGameState);
    }

    @Test
    public void testKeyDownEnterInSumScreen() {
        when(myRowing.miniGameState).thenReturn(MiniGameState.SUM_SCREEN);
        miniGameInputProcessor.keyDown(Input.Keys.ENTER);
        verify(myRowing, times(1)).resetMiniGame();
        assertEquals(MiniGameState.NOT_STARTED, myRowing.miniGameState);
    }
}
```