# Index:

**Gameplay:**

* TC\_Boat
* TC\_Lane
* TC\_Leg
* TC\_Obstacle

**Client Specifications:**

* TC\_Powerups

**Utilities:**

* TC\_KeyBindings

We decided on ordering the tests by what we felt has higher priority. Firstly, we test all JUnit possible tests essential for the game, Boat, lanes, legs and obstacles. Then proceed to test the client requirements as our main goal is satisfying all the requirements of our client. And finally, we have a test for the assignation of Key Binds that we decided on giving its own category.

# Testing Plan

**Project Name:**

*UMA-ISE24-E1*

Authors:

*Ricardo Juan Allitt López, Manuel Barrios Moreno, Ángel Bayón Pazos, Ángel Nicolás Escaño López, Pablo Hormigo Jiménez, Francisco Javier Jordá Garay, Diego Sicre Cortizo, Muhammad Abdullah Sultán* *Sultán, Juan Torres Gómez.*

## **Gameplay:**

This section includes the most significant tests for the correct execution of the program, in case you want to test functions, these should be the main ones.

## Test Case Overview

**Test Case ID:**

*TC\_Boat*

**Purpose:**

*Verify that boats can be created properly, ensure correct movement and test for inconsistencies when modifying their attributes.*

*This test cases involves FR002(Boats).*

**Test Case Description:**

*This test case checks the correct functioning of the class boats and its checkable methods.*

*JUnit tests related to this test case will check methods related to the creation, attributes, movement and destruction of the boat.*

### Pre-Conditions

**Prerequisites:**

*Class Boat and its methods are properly implemented.*

**Test Data:**

*Requires boat type and their map positions for initial setUp function.*

### Test Steps

**Step Description:**

1. *SetUp function is called before every test to create a boat of each type.*
2. *The first test method tests the createBoat method from class Boat to ensure correct functioning of the createBoat method.*
3. *The second method and third methods test for the assignation and change of boat attributes.*
4. *The fourth method testInvincibility ensures the boat can become invincible.*
5. *The fifth method checks that the boat is destroyed correctly.*
6. *The final method checks the boat's capability to move on the map.*

### Post-Conditions

**Expected Outcome:**

*JUnit passes every one of the previous tests meaning the boat's main functions work correctly.*

**Cleanup:**

*Boat objects created for the test are destroyed.*

## Test Case Overview

**Test Case ID:**

*TC\_Lane*

**Purpose:**

*Verify that lane can incorporate and remove objects from the screen.*

*This test cases involves FR002(Boats), FR006(Power-Ups), FR004(Obstacles).*

**Test Case Description:**

*This test evaluates the correct functioning of the class Lane and its checkable methods.*

*JUnit tests related to this test case will check the correct functioning of the creation of a lane and the ability to create and remove objects in a lane(obstacles,boats,powerups).*

### Pre-Conditions

**Prerequisites:**

*Class Lane and its methods are properly implemented.*

**Test Data:**

*Lane Id, a set of obstacles, a set of powerups, boat type and boat positioning.*

### Test Steps

**Step Description:**

1. *SetUp function is called before every test to create a boat, a set of powerUps, a set of obstacles and a lane.*
2. *The first test method tests the creation of a lane ensuring the correct attributes have been passed.*
3. *The second method checks the method addGameObject to check if the desired obstacle is added to the lane.*
4. *The third method checks the method remove GameObject to check if the desired obstacle is deleted from the lane.*
5. *The fourth and fifth methods check the correct functioning of the addGameObject/deletedGameObject methods when referring to powerUps to see that they are added and deleted correctly;*
6. *The sixth and seventh methods check that boats are added and deleted correctly via the addGameObject/deletedGameObject methods.*

### Post-Conditions

**Expected Outcome:**

*JUnit passes every one of the previous tests meaning the lane´s main functions work correctly.*

**Cleanup:**

*None*

## Test Case Overview

**Test Case ID:**

*TC\_Leg*

**Purpose:**

*Verify that Leg is created properly along with its parameters.*

*This test case involves FR001(Player).*

**Test Case Description:**

*This test case evaluates the functionality of the Leg class within the constraints of Junit.*

*Related JUnit tests will check methods relative to the creation of the Leg class.*

### Pre-Conditions

**Prerequisites:**

*Class Leg and its methods are properly implemented.*

**Test Data:**

*Leg Id and an object of type User.*

### Test Steps

**Step Description:**

1. *First, we initialize* *Leg and player with the setUp method to use them in the following test.*
2. *The method testInitialization checks that the object of type Leg is created properly checking the correctness of its attributes.*

### Post-Conditions

**Expected Outcome:**

*JUnit passes every one of the previous tests meaning leg is created properly.*

**Cleanup:**

*None.*

## Test Case Overview

**Test Case ID:**

*TC\_Obstacle*

**Purpose:**

*Verify that obstacles spawn/de-spawn correctly and that they move to the desired position.*

*This test case involves FR004(Obstacles).*

**Test Case Description:**

*This test case evaluates the creation of obstacles and their ability to move in the lanes.*

*Related JUnit tests will check methods relative to the creation of the obstacles and their movement.*

### Pre-Conditions

**Prerequisites:**

*Class Obstacle and its methods are properly implemented.*

**Test Data:**

*Initial positions and size of the three different types of obstacles and their respective damage.*

### Test Steps

**Step Description:**

1. *The method setUp creates the hitboxes of our three obstacles and then creates them to be used in the later tests.*
2. *The second, third and fourth method test the creation of all three obstacles.*
3. *The fifth, sixth and seventh methods test the destruction of all three obstacles.*
4. *The eighth and ninth method check the movement function of the class obstacle works correctly for the obstacle of type duck.*

### Post-Conditions

**Expected Outcome:**

*JUnit passes every one of the previous tests meaning Obstacle´s main functions work properly.*

**Cleanup:**

*None.*

## **Client Specifications:**

All the tests belonging to this section are those requirements defined by the clients, they have a lower rank of importance compared to the gameplay because they are not essential for the correct execution of the game.

## Test Case Overview

**Test Case ID:**

*TC\_Powerups*

**Purpose:**

*Verify that powerups are modifying a boat´s attribute correctly.*

*This test cases involves FR006(Power-ups), FR002(Boats)*

**Test Case Description:**

*This test case evaluates the functionality of Speed, HP and Invincibility Power-ups within the game interface.*

*Related JUnit tests will check the creation and appliance of the three different Power-ups.*

### Pre-Conditions

**Prerequisites:**

*Class Boat and its methods are properly implemented.*

*Class HealthBoost and its methods are properly implemented.*

*Class SpeedBoost and its methods are properly implemented.*

*Class InvincibilityBoost and its methods are properly implemented.*

**Test Data:**

*Height and width of hitbox for the power up creation.*

### Test Steps

**Step Description:**

1. *The method Creation is used to create all three power ups for them to be used in the other tests.*
2. *For all three power ups, there is a method Apply that tests the ability of each power up to modify the attributes of a boat.*

### Post-Conditions

**Expected Outcome:**

*JUnit passes every one of the previous tests meaning all three different Power-Up´s main functions work properly.*

**Cleanup:**

*None*

## **Utilities:**

The utilities of the program are those specifications that make the program more complete and give the user a wider range of possibilities when navigating through the system.

## Test Case Overview

**Test Case ID:**

*TC\_KeyBindings*

**Purpose:**

*Verify that the user can customize the controls.*

*This test cases involves FR018(Controls), FR001(User).*

**Test Case Description:**

*This test case checks whether the control related tests can be performed with different control settings.*

*JUnit tests related to this test case will check the methods that update the variables related to keyboard listeners.*

### Pre-Conditions

**Prerequisites:**

*Class KeyBindings and its methods are properly implemented.*

**Test Data:**

*none.*

### Test Steps

**Step Description:**

1. *The function setUp is executed in order to create an object of type KeyBindings for the other tests.*
2. *The second function tests to see that there are default key bind values and that they are set as soon as the object KeyBindings is created.*
3. *The third method checks the ability to change the value of a specific keybind to another key on the keyboard.*

### Post-Conditions

**Expected Outcome:**

*JUnit passes every one of the previous tests meaning KeyBinding´s main functions work properly.*

**Cleanup:**

*None*