# Index:

**Gameplay:**

* TC\_Boat
* TC\_Lane
* TC\_Obstacle
* TC\_ObstacleVisitorTest
* TC\_CollidableTest

**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 next seven methods test all methods related to modifying a boat´s attributes such as speed and health.*
4. *The eight method testIsInvincible checks if the boat properly turns invincible if it has been in contact with an invincibility power up*
5. *The night method tests to see If the boat has been destroyed meaning all the hit points have been taken away from the boat*
6. *The final method tests the ability to reset all boats stats to default.*

### 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 testGetBoat to check if a boat is added to the lane correctly*
4. *The third method checks the method testGetBoat Obstacles to check if an obstacle is added to the lane correctly.*
5. *The third method checks the method testGetPowerUps to check if power-ups are added to the lane correctly*

### Post-Conditions

**Expected Outcome:**

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

**Cleanup:**

*Destruction of constructor objects.*

## 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:**

*Three different objects of type Obstacle all three being of different types.*

### 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 method tests the creation of all three obstacles.*
3. *The third, fourth and fifth methods test the creation obstacles creating all three different types with their correct stats.*

### Post-Conditions

**Expected Outcome:**

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

**Cleanup:**

*Destruction of constructor objects.*

## Test Case Overview

**Test Case ID:**

*TC\_*ObstacleVisitorTest

**Purpose:**

Verify that obstacles of all three types register the act of collision with another object.

This test cases involves FR004(Obstacle).

**Test Case Description:**

This test case checks the correct function of the method visitObstacle of the interface ObstacleVisitor.

JUnit tests related to this test case will check methods related to the detection of collision of obstacles whether it´s a duck, log or stone.

### Pre-Conditions

**Prerequisites:**

*Class Obstacle and its methods are properly implemented.*

**Test Data:**

*Three different objects of type Obstacle all three being of different types and an object of type Visitor.*

### Test Steps

**Step Description:**

1. The first method setUp tests sets the state of the previously created obstacles to hit and initializes them.
2. The three following methods check if the obstacles detect correctly the fact that a collision has been caused using the getwasHit method.

### Post-Conditions

**Expected Outcome:**

*JUnit passes every one of the previous tests meaning Obstacles are capable of detecting collisions and therefore respond with the correct changes in attributes.*

**Cleanup:**

*Destruction of constructor objects.*

## Test Case Overview

**Test Case ID:**

*TC\_CollidableTest*

**Purpose:**

Ensure that collisions between boats (player and AI) and other objects (obstacles and powerUps) in the game environment are detected and handled correctly.

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

**Test Case Description:**

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

JUnit tests related to this test case will check the correct functioning of the collision detection and its corresponding response.

### Pre-Conditions

**Prerequisites:**

Classes Boat, Obstacle, PowerUp and their methods are properly implemented.

**Test Data:**

The creation of the boat (independent of its type).

### Test Steps

**Step Description:**

1. First, we initialize Boat with the setUp method to use it in the following test.
2. The second function tests to see if the boat changes collision status when function setWasHit is used.
3. The second function tests to see if the boat changes collision status when function getWasHit is used.
4. Finally, the last method ensures that the boat has a non-null rectangular hitbox, so a collision can be caused.

### Post-Conditions

**Expected Outcome:**

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

**Cleanup:**

*Destruction of constructor objects.*

## **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 setUp 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:**

*Destruction of constructor objects.*

## **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:**

*Object of class KeyBindings*

### 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, fourth and fifth methods check 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:**

*Destruction of constructor objects.*