Group 2: Bump Boat Game Groups

GitHub link
https://github.com/ernmri/UMAsoftwareEngineeringGroupProject

Members

Gonzalo García Rojas; gonzalogrojas@uma.es

Youssef Merroun; youssefmerroun333@gmail.com

Daniil Gumeniuk; gumuma@uma.es

Alejandro Puerto Criado; alejandropc2004@uma.es

Eron Imeri; eron.imeri@uma.es | eronimeri@hotmail.com

Gianluca Nanni; gianlu.nanni@uma.es | gianlu.nanni@gmail.com

Pablo García Moreno; pablogarciamoreno11@gmail.com

Francisco Galisteo Guzmán; fran2@uma.es | frangalisteo2@gmail.com

Table of Contents

Introduction

Dragon Boat Race is a single-player game that leverages advanced game engine technology to deliver a dynamic and immersive experience. The game features multiple choices of boats with different specs, fatigue simulation and obstacles to avoid.

The goal of the game is to win boat races throughout your ability, that will increase while playing, as well as the characteristics of the boat, that you will be able to upgrade temporarily between one race and another, thanks to the featured minigame.

Roles

Developers:

- Gonzalo García Rojas
- Eron Imeri
- Daniil Gumeniuk
- Gianluca Nanni

Project managers:

- Youssef Merroun
- Alejandro Puerto Criado
- Francisco Galisteo Guzmán

Graphic designers:

- Alejandro Puerto Criado
- Gonzalo García Rojas
- Pablo Garcia Moreno
- Eron Imeri

Testers:

- Gianluca Nanni
- Youssef Merroun
- Pablo García Moreno
- Francisco Galisteo Guzmán

Scrum masters:

- Daniil Gumeniuk

Product Owner:

- Gonzalo García Rojas

Possible risks

RISK	TYPE	PROBABILITY	EFFECTS	SOLUTION
Underestimation of the complexity of the project	Estimation risk	Moderate probability	Serious effects	To mitigate this risk, our teams must plan each of the steps of the project before starting them and have good communication between all of the members in order to know if we are advancing according to our deadlines.
Quality risk at the end of the project	Requirements risk	High probability	Catastrophic effects	To mitigate this risk, we should check the quality of everything we implement in the software, just after implementing it. In order to do it, some of us will have the role of 'tester', so that is their duty to test the code and the

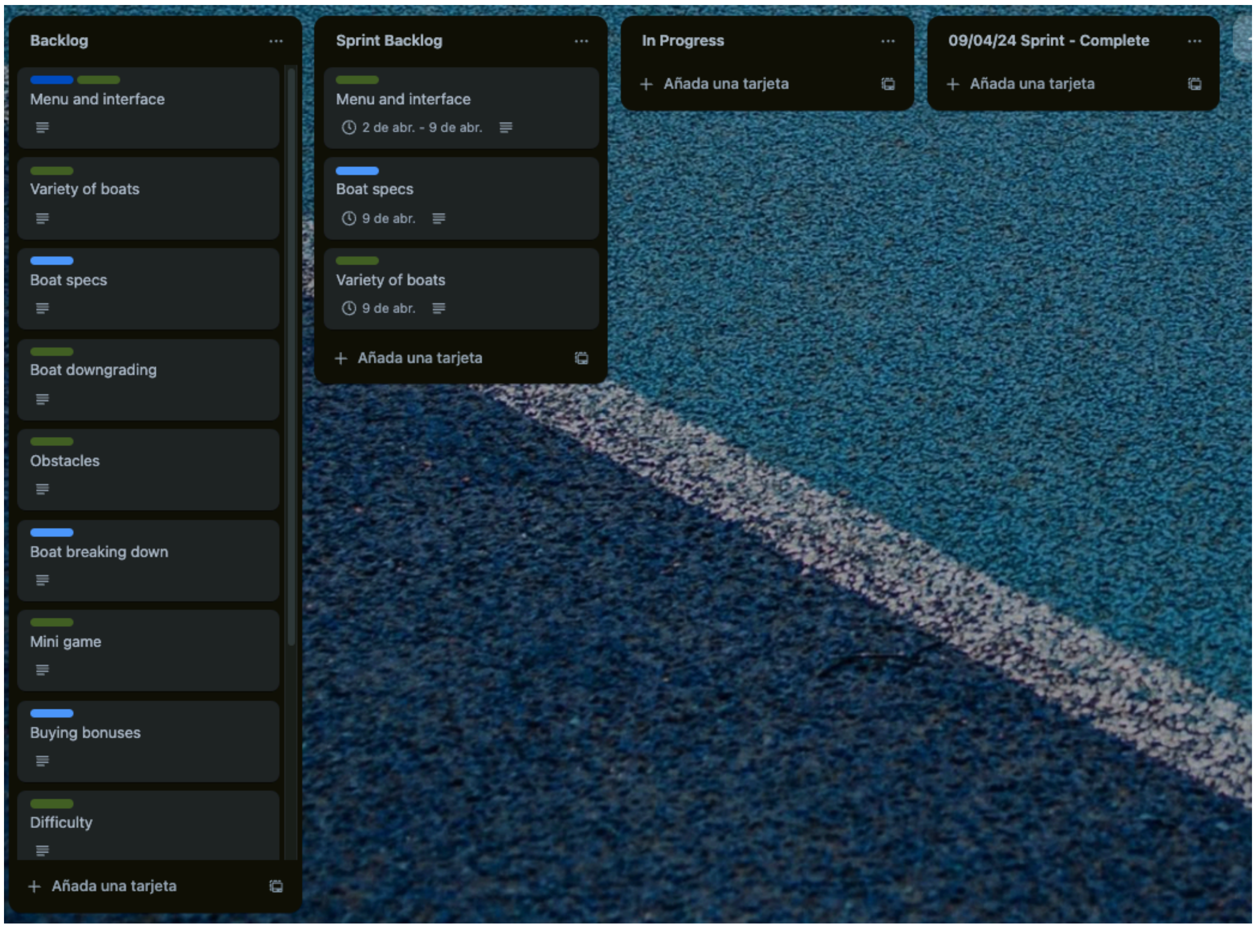
				other parts of the project.
Insufficient qualifications to complete the project	Personnel risk	Moderate probability	Tolerable effect	To mitigate this risk we will apply research online and we will apply a well-thought of understanding of the tools we will need so that the research takes less time; we will also work and be dependent on each other as a team so that we can combine the skills each of us has.
Dependency between team members	Personnel risk	Low probability	Serious effect	To mitigate this risk, we should have a contingency plan to divide the work of a team member if they are not available for major reasons, for example, disease.
Lack of time	Estimation risk	High probability	Catastrophic effects	To mitigate this risk, we should have a good planification and communication between all the members.

Technologies employed	Technology risk	Moderate probability	Serious effects	To mitigate this risk, we should always check the correct functioning of the technologies employed, and don’t get them overused

Planning

We have selected Scrum as our software process model due to its facilitation of effective communication among team members, encourages everyone's participation throughout all stages, aligns us towards common goals, allows for project changes and its efficacy in time optimization. We have planned to organize our meetings to check the advancements on the project and to see if anyone is having difficulties with their part of the job once a week. Our scrum master (that is the leader of the team who also must assure that we follow the scrum method) is Daniil Gumeniuk and the product owner (the person in charge of making sure that requirements are fulfilled and that the quality of the final product is the best possible) is Gonzalo García Rojas.

Trello boards:



Backlog

Menu and interface

Variety of boats

Boat specs

Boat downgrading

Obstacles

Boat breaking down

Mini game

Buying bonuses

Difficulty

Añada una tarjeta

Sprint Backlog

Menu and interface
2 de abr. - 9 de abr.

Boat specs
9 de abr.

Variety of boats
9 de abr.

Añada una tarjeta

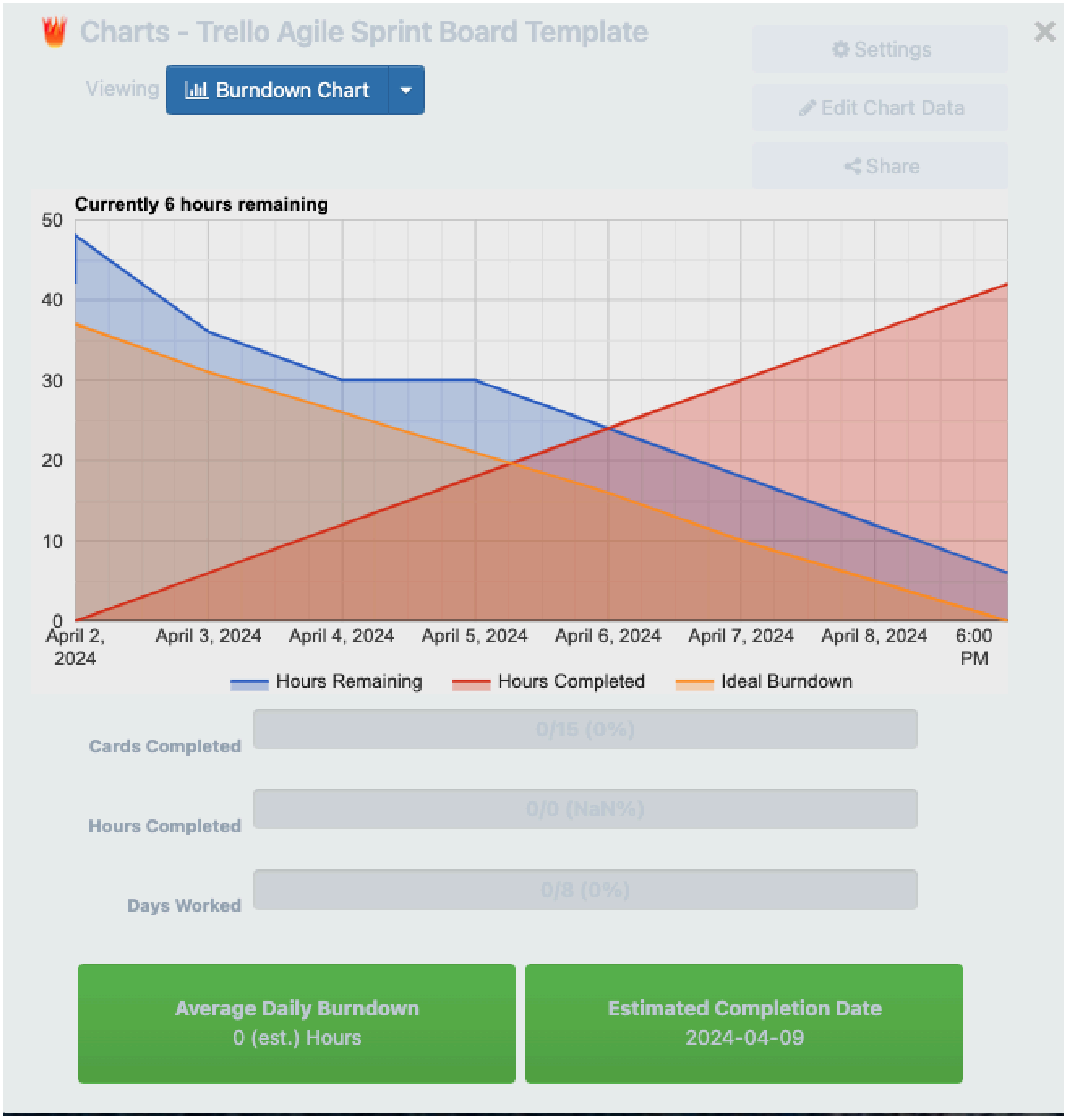
In Progress

Añada una tarjeta

09/04/24 Sprint - Complete

Añada una tarjeta

Burndown Chart for first sprint



Charts - Trello Agile Sprint Board Template

Cards Completed 0/15 (0%)

Hours Completed 0/0 (NaN%)

Days Worked 0/8 (0%)

Average Daily Burndown
0 (est.) Hours

Estimated Completion Date
2024-04-09

Software tools used during project's realization

- Collaborative work:
    - Github: Code updating.
    - Trello: Tracking progress.
    - Visual Studio Code : IDE.
    - Visual Paradigm: creating diagrams to describe and organize the project.

- Communication:
    - Discord: Meetings to work on common tasks.
    - Whatsapp: Documents and ideas sharing, with setting-up meetings.

- Document Elaboration:
    - Google Documents: project description and specification, admits group-working on the same document
    - PDF online converter : To convert .docx into .pdf in order to submit.

Requirements

ID	REQUIREMENT	PRIORITY	Test
FR1	There must be a start menu and a graphic interface to indicate different aspects of the game	**High**	
FR2	Players can choose between boats with different specs to compete in the leg	**Medium**	**TC001**
NFR1	These boats are differentiated by four specs: speed, acceleration, maneuverability and robustness.	**High**	**TC002**
FR3	Speed, acceleration and maneuverability decrease	**High**	**TC003**

	progressively during a leg.		
NFR2	The robustness is expressed in terms of points.	**Medium**	
FR4	There must be obstacles during the leg that can reduce the robustness of the boat.	**High**	
FR5	There are three kinds of obstacles: the logs, the rocks and the ducks.	**Medium**	
FR6	The log is a static obstacle that removes two points of robustness	**Medium**	**TC004**
FR7	The rock is a static obstacle that removes three points of robustness.	**Medium**	**TC005**

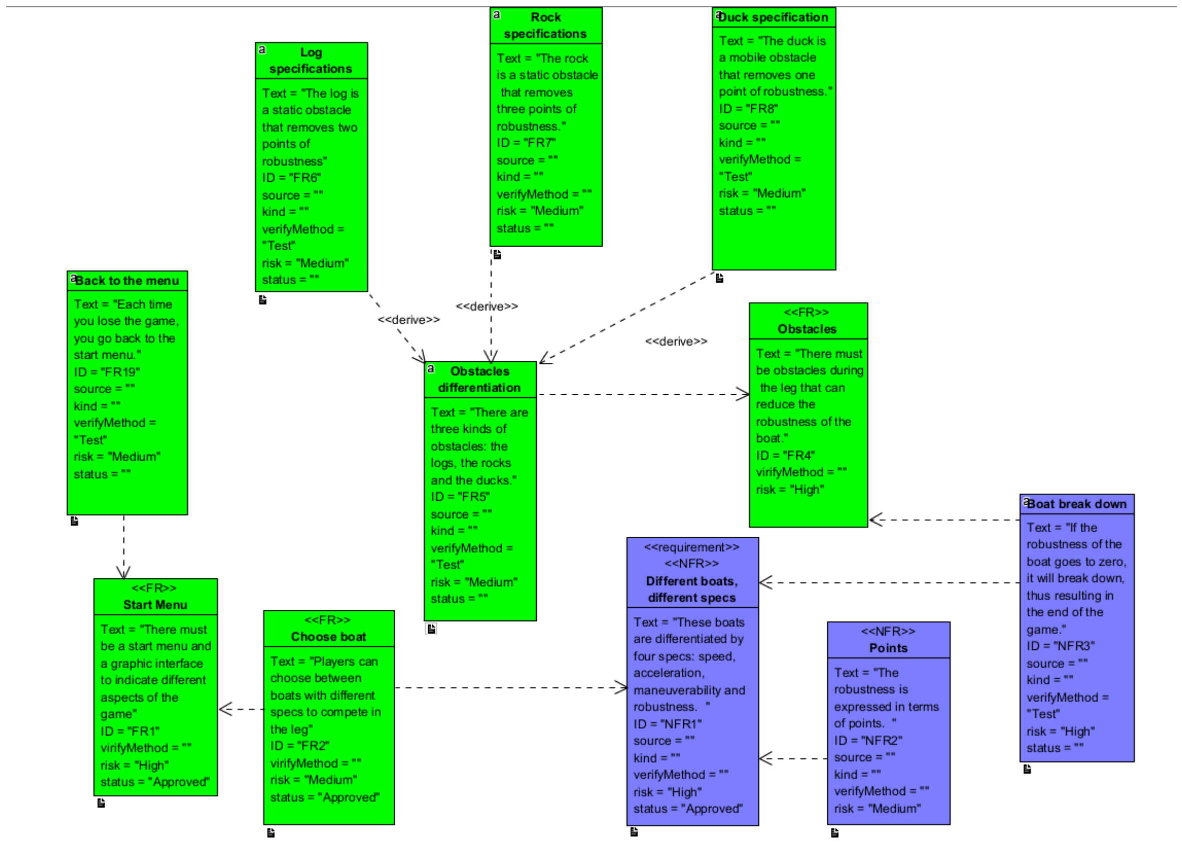
FR8	The duck is a mobile obstacle that removes one point of robustness.	**Medium**	**TC006 TC007**
NFR3	If the robustness of the boat is null, it will break down, thus resulting in the end of the game.	**High**	**TC008**
FR9	There must be a minigame between each of the legs.	**High**	**TC009**
NFR4	There must be a timer in the minigame.	**Low**	
FR10	The minigame takes place in a lake, where you still control a boat.	**Low**	
FR11	The only mechanic of the minigame is obtaining coins	**Low**	

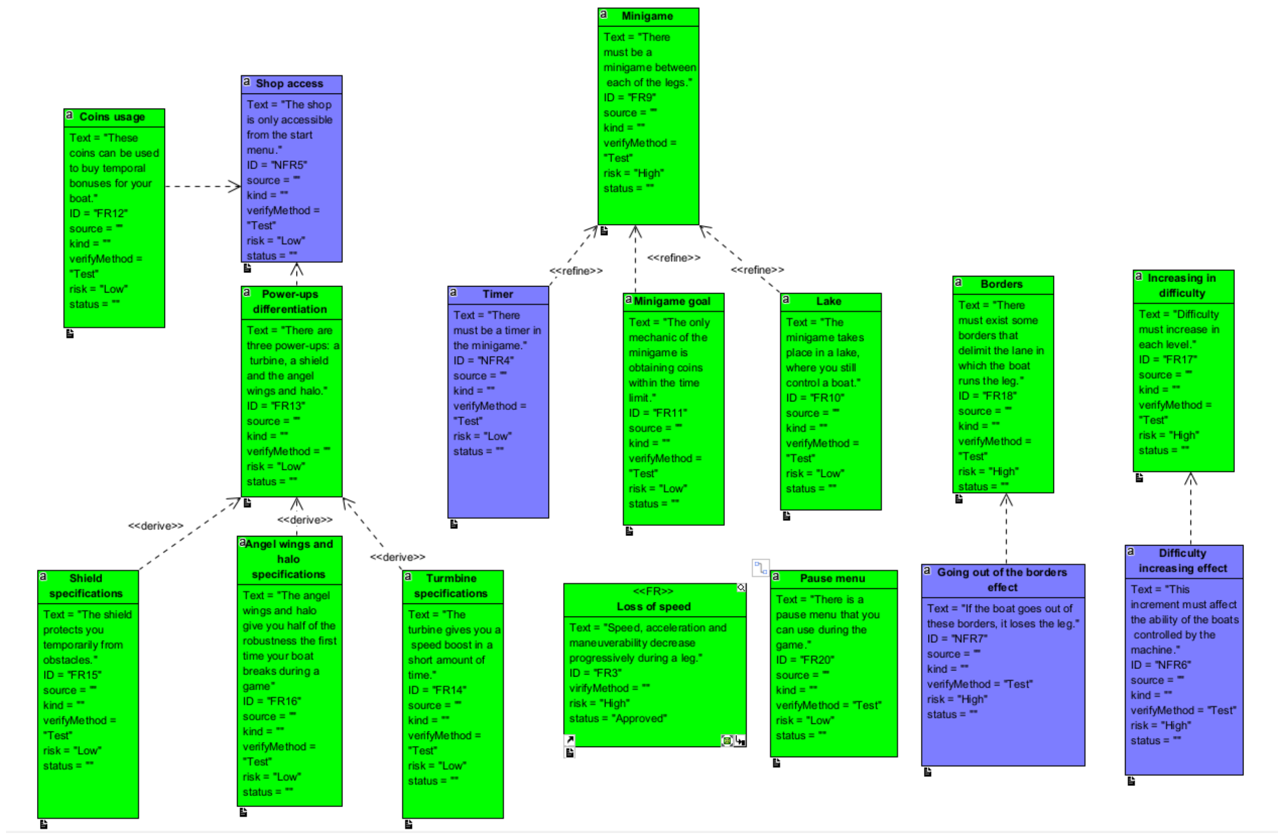
	within the time limit.		
FR12	These coins can be used to buy temporal bonuses for your boat.	**Low**	
NFR5	The shop is only accessible from the start menu.	**Low**	
FR13	There are three power-ups: a turbine, a shield and the angel wings and halo.	**Low**	
FR14	The turbine gives you a speed boost in a short amount of time.	**Low**	**TC010**
FR15	The shield protects you temporarily from obstacles.	**Low**	**TC011**
FR16	The angel wings and halo give you half of the robustness if your	**Low**	**TC012**

	boat breaks during a leg.		
FR17	Difficulty must increase in each level.	**High**	**TC013**
NFR6	This increment must affect the ability of the boats controlled by the machine.	**High**	
FR18	There must exist some borders that delimit the lane in which the boat runs the leg.	**High**	
FR19	If the boat goes out of these borders, it loses the leg.	**High**	**TC014**
FR20	Each time you lose the game, you go back to the start menu.	**Medium**	**TC015**

FR21	There is a pause menu that you can use during the game.	**Low**	

Requirements diagram:


Log specifications
Text = "The log is a static obstacle that removes two points of robustness"
ID = "FR6"
source = ""
kind = ""
verifyMethod = "Test"
risk = "Medium"
status = ""
Rock specifications
Text = "The rock is a static obstacle that removes three points of robustness."
ID = "FR7"
source = ""
kind = ""
verifyMethod = ""
risk = "Medium"
status = ""
Duck specification
Text = "The duck is a mobile obstacle that removes one point of robustness."
ID = "FR8"
source = ""
kind = ""
verifyMethod = "Test"
risk = "Medium"
status = ""
Back to the menu
Text = "Each time you lose the game, you go back to the start menu."
ID = "FR19"
source = ""
kind = ""
verifyMethod = "Test"
risk = "Medium"
status = ""
<<derive>>
<<derive>>
<<derive>>
<<FR>>
Obstacles
Text = "There must be obstacles during the leg that can reduce the robustness of the boat."
ID = "FR4"
virifyMethod = ""
risk = "High"
Obstacles differentiation
Text = "There are three kinds of obstacles: the logs, the rocks and the ducks."
ID = "FR5"
source = ""
kind = ""
verifyMethod = "Test"
risk = "Medium"
status = ""
Boat break down
Text = "If the robustness of the boat goes to zero, it will break down, thus resulting in the end of the game."
ID = "NFR3"
source = ""
kind = ""
verifyMethod = "Test"
risk = "High"
status = ""
<<requirement>>
<<NFR>>
Different boats, different specs
Text = "These boats are differentiated by four specs: speed, acceleration, maneuverability and robustness. "
ID = "NFR1"
source = ""
kind = ""
verifyMethod = ""
risk = "High"
status = "Approved"
<<FR>>
Start Menu
Text = "There must be a start menu and a graphic interface to indicate different aspects of the game"
ID = "FR1"
virifyMethod = ""
risk = "High"
status = "Approved"
<<FR>>
Choose boat
Text = "Players can choose between boats with different specs to compete in the leg"
ID = "FR2"
virifyMethod = ""
risk = "Medium"
status = "Approved"
<<NFR>>
Points
Text = "The robustness is expressed in terms of points. "
ID = "NFR2"
source = ""
kind = ""
verifyMethod = ""
risk = "Medium"



Unit Test Documentation Template:

Project Name:

Group 2: Bump Boat Game Groups

Author(s):

Gonzalo Garcia Rojas

Youssef Merroun 0611100042@uma.es | youssefmerroun333@gmail.com

Daniil Gumeniuk gumuma@uma.es | gumenyukdp@gmail.com

Alejandro Puerto Criado alejandropc2004@gmail.com

Eron Imeri eronimeri@hotmail.com | eron.imeri@uma.es

Gianluca Nanni gianlu.nanni@uma.es | gianlu.nanni@gmail.com

Pablo Garcia Moreno

Francisco Galisteo Guzmán fran2@uma.es | frangalisteo2@gmail.com

Test cases:

Test Case ID: TC001

Purpose:
This test should test the possibility of choosing different boats, and that the selected boat specs are different. It is related to FR2.

Test Case Description:
We are selecting a boat from a list of objects of type 'Boat' and we are showing its characteristics in order to check if the function to select boats works.

Pre-Conditions

Prerequisites:
Having a list of available boats, with different specs, from which you can choose.

Test Data:
A list of available boats and the positions (different integer numbers) of the boat we want to choose from the list.

Test Steps

Step Description:

1. *Select a boat from the list, by calling the related function*
2. *Show the specifications of the boats on screen to demonstrate that the function works*

Post-Conditions

Expected Outcome:
It is expected that the specifications of the selected boat are correctly shown on screen.

Cleanup:
Nothing in particular.

—---

Test Case ID: **TC002**

Purpose:
This test should test the possibility of choosing different boats, and that the selected boat specs are different. It is related to NFR1.

Test Case Description:
We are selecting two different boats from a list of available boats. Then, by checking if the specifications of the boats are not the same we test that the selected boats are different.

Pre-Conditions

Prerequisites:
Having a list of available boats, with different specs, from which you can choose.

Test Data:
A list of available boats and the positions (different integer numbers) of the boats we want to choose from the list.

Test Steps

Step Description:

1. *Select two different boats from the list, by calling the related function two time*
2. *Check that the boats are different by comparing their attributes, which we will get by calling the 'Get' methods provided by the class 'Boat'*

Post-Conditions

Expected Outcome:
All the boats must differentiate in at least one aspect, such as speed, robustness, maneuverability.

Cleanup:
Nothing in particular.

—---

Test Case ID: **TC003**

Purpose:
It should prove that during a leg there exists a decay of the specs of the boats. It's related to FR3.

Test Case Description:
We are going to check the specs of the boats during a leg, and see if the specs are different from the specs before the game started.

Pre-Conditions

Prerequisites:
Having the leg designed and having the instance of the boat chosen available (the values of the specifications of the boat have to be greater than 0).

Test Data:
An object of type boat should be passed as input argument.

Test Steps

Step Description:

1. *Store the specs of the boat in locale variables at the beginning of the leg*
2. *Wait for a certain amount of time*
3. *Check that the specs decay by comparing the new specs with the variables containing the original ones*

Post-Conditions

Expected Outcome:
The specs values at the end of the test have to be worse that the original ones

Cleanup:
Nothing.

—---

Test Case ID: TC004

Purpose:
It should prove that hitting a log reduces the robustness of the boat by 2 points. It is related to FR6.

Test Case Description:
We will test if the boats get reduced in robustness after hitting an obstacle of type 'Log'. We will check the specs of the robustness before and compare it with the specs of the boat after hitting the obstacle. If the robustness of the boat has decreased by exactly 2 points after the collision, it has been reduced successfully.

Pre-Conditions

Prerequisites:
There should be obstacles of type 'Log' programmed into the game. The boat must have its initial robustness greater than 2.

Test Data:
The test should receive as arguments one object of class 'Boat' and one of type 'Log'.

Test Steps

Step Description:

1. *Create a local variable 'initial_robustness' to store the robustness of the boat at the beginning of the test*
2. *Make the boat collide with a log*
3. *Check if the robustness of the boat is now equal to the value of expected_final_robustness = initial_robustness - 2*

Post-Conditions

Expected Outcome:
It is expected that the value of robustness of the boat is reduced by a value of 2 after the collision.

Cleanup:
Nothing in particular.

—---

Test Case ID: TC005

Purpose:

It should prove that hitting a rock reduces the robustness of the boat by 3 points. It is related to FR7.

Test Case Description:

There should be obstacles of type 'Rock' programmed into the game. The boat must have its initial robustness greater than 3.

Pre-Conditions

Prerequisites:
The robustness of the boat has to be greater than 3.

Test Data:
The test should receive as arguments one object of class 'Boat' and one of type 'Rock'.

Test Steps

Step Description:

1. *Create a local variable 'initial_robustness' to store the robustness of the boat at the beginning of the test*
2. *Make the boat collide with a rock*
3. *Check if the robustness of the boat is now equal to the value of expected_final_robustness = initial_robustness - 3*

Post-Conditions

Expected Outcome:
It is expected that the value of robustness of the boat is reduced by a value of 3 after the collision.

Cleanup:
Nothing in particular.

—---

Test Case ID: **TC006**

Purpose:
It should prove that hitting a duck reduces the robustness of the boat by 1 point. It is related to FR8.

Test Case Description:
We will test if the boats get reduced in robustness after hitting an obstacle of type 'Duck'. We will check the specs of the robustness before and compare it with the specs of the boat after hitting the obstacle. If the robustness of the boat has decreased by exactly 1 point after the collision, it has been reduced successfully.

Pre-Conditions

Prerequisites:
There should be obstacles programmed into the game. The boat must have its initial robustness greater than 1.

Test Data:
The test should receive as arguments an object of class 'Boat' and one of type 'Duck' .

Test Steps

Step Description:

1. *Create a local variable 'initial_robustness' to store the robustness of the boat at the beginning of the test*
2. *Make the boat collide with the duck*

3. *Check if the robustness of the boat is now equal to the value of expected_final_robustness = initial_robustness - 1*

Post-Conditions

Expected Outcome:
It is expected that the values related to the robustness of the boat is reduced by a value of 1 after the collision.

Cleanup:
Nothing in particular.

—---

Test Case ID: **TC007**

Purpose:
In this test case we check if the duck is moving throughout the time. It is related to FR8.

Test Case Description:
We will check if the position of the duck changes by storing it into two local variables (x and y) and then comparing it with the final position of the duck.

Pre-Conditions

Prerequisites:
The environment without any obstacle except for the duck

Test Data:
This test case requires only the object of the class duck and a timer

Test Steps

Step Description:

1. *The initial position of the duck is stored into local variables (one for storing the position x and the other one for storing the position y)*
2. *The timer is set up for a certain amount of time*
3. *After the timer exceeds the the variables are compared with the attributes storing current position of the duck*

Post-Conditions

Expected Outcome:
The position stored must differ from the final position of the duck at the end of the test (not both the attributes should differ, it is enough that only the value of one of them is different from its previous one).

Cleanup:
Nothing in particular.

—---

Test Case ID: ***TC008***

Purpose:
This test is intended to test if the boat breaks down when its robustness value reaches 0. It is related to NF3.

Test Case Description:
The player will collide with any obstacle and the boat will run out of robustness resulting in a breakdown.

Pre-Conditions

Prerequisites:
The robustness of the boat equals 3.

Test Data:
An object of type 'Boat' and one object of type 'Rock' will be passed as an argument.

Test Steps

Step Description:

1. *The boat initializes with an amount of robustness equal to 3*
2. *The boat collides with the rock and its robustness reaches the value of 0*
3. *Check if the boat gets broken when it happens (by checking the state of the related boolean attribute provided by the Boat object)*

Post-Conditions

Expected Outcome:
The boat is expected to break down once it runs out of robustness.

Cleanup:
Nothing in particular.

—---

Test Case ID: ***TC009***

Purpose:
This test is intended to verify that a mini-game starts properly after each leg. This test is related to FR9.

Test Case Description:
When a player wins a leg he/she gets an opportunity to win extra coins by playing a mini-game. This test is created using a boat and an empty lane so the boat can reach the end safely. After that, the main's game attribute 'minigame_running' is checked to be true, making sure the minigame has correctly started.

Pre-Conditions

Prerequisites:
A leg has to be just ended. There must be no obstacles on the leg

Test Data:
No input data is needed

Test Steps

Step Description:

1. *A leg is started and is won by the player*
2. *Check if the mini game takes place by checking that the related boolean value minigame_running equals 'true'*

Post-Conditions

Expected Outcome:
After a leg is won, the mini game starts.

Cleanup:
Nothing in particular.

—---

Test Case ID: ***TC010***

Purpose:
Check if the turbine gives the boat a speed boost in a short period of time. It is related to FR14.

Test Case Description:
We have to test that this power-up increases the speed of the boat during a short period of time. This will be done by passing a boat with and a power-up (which is an enumeration) and storing the initial speed of the boat and then comparing it with the speed after applying the power up.

Pre-Conditions

Prerequisites:
Having available the boat with the robustness greater than 0.

Test Data:
The test method should receive as an argument the boat and a power-up of type TURBINE.

Test Steps

Step Description:

1. *Store the value related to the velocity of the boat in a local variable*
2. *Make a call to the function 'Activate_power_up' with the parameter related to the power-up set to the value 'TURBINE'*
3. *Wait for a reasonable amount of time and check if the speed has been correctly increased, by comparing the new value to the original one*
4. *Wait another little amount of time, in order to check if the velocity of the boat has increased again*
5. *Wait until the power-up effect should be finished, and check if it happened correctly, by checking the final value of the velocity of the boat*

Post-Conditions

Expected Outcome:

The speed of the boat is expected to be increased in respect to the original one after calling the function, and then it is expected to return to its original value, when we check its value for the second time

Cleanup:

Nothing in particular.

—---

Test Case ID: ***TC011***

Purpose:

The shield should protect you temporarily from obstacles. It is related to FR15.

Test Case Description:

We will test if the shield evitates reductions in terms of robustness when it is activated. As programmed, it should not reduce the value of robustness for the boat when activated.

Pre-Conditions

Prerequisites:

An object of type 'Boat' has to be available, and it has to have a value related to robustness different from 0.

Test Data:

The test should receive as arguments an object of class 'Boat' the robustness of which is different from 0, an instance of the class log (every obstacle would be good for this test).

Test Steps

Step Description:

1. *Put the value of the robustness of the boat into a locale variable*
2. *Make a call to the function 'Activate_power_up' with the parameter related to the power-up set to the value 'SHIELD'*
3. *Make the boat collide with the log*
4. *Check that the robustness value of the boat hasn't changed after each collision*

Post-Conditions

Expected Outcome:

It is expected that the values related to the robustness of the boats do not change as the shield is activated.

Cleanup:

Nothing in particular.

—---

Test Case ID: ***TC012***

Purpose:
Verified the correct functioning of a power-up called “angel wing and halo”. It is related to the FR16.

Test Case Description:
This test should check that, if this power-up is activated, when the boat breaks down, the robustness of the boat is restored to the value of a half of its maximum (only one time).

Pre-Conditions

Prerequisites:
The test should receive as arguments an object of class ’Boat’ with its maximum amount of robustness (its original value), an instance of the class duck and a power-up of type ‘ANGEL_WING_AND_HALO’.

Test Data:
The test should receive as arguments an object of class ’Boat’ with its maximum amount of robustness (its original value) and an instance of the class duck.

Test Steps

Step Description:

1. *Store the value of robustness of the boat in a variable called ‘max_robustness’*
2. *Set the robustness of the boat to the value of 1, by calling the set method provided by the ‘Boat’ class*
3. *Make a call to the function ‘Activate_power_up’ with the parameter related to the power-up set to the value ‘ANGEL_WING_AND_HALO’*
4. *Make the boat collide with the duck and consequently get broken down*
5. *Check that the power-up has correctly worked by checking that the robustness of the boat is now equal to the half of its original value (current_robustness = 0.5 * max_robustness)*

Post-Conditions

Expected Outcome:
The boat has to be restored after applying the power up, the robustness of the boat must be exactly half of the value stored in the local variable.

Cleanup:
Nothing in particular.

—---

Test Case ID: ***TC013***
Purpose:
It should prove that difficulty increases with each level. It is related to the FR17.

Test Case Description:
We will test if any of the opponents’ attributes have increased compared to the previous level.

Pre-Conditions

Prerequisites:
Winning the previous level

Test Data:
The instance of the opponent's boat

Test Steps

Step Description:

1. *Start one race and store the velocity of the computer-controlled boats into an adequate number of variables*
2. *Start another race after winning the previous one and check that the velocities of the boats have increased in respect to the variables stored before*

Post-Conditions

Expected Outcome:
It is expected that the speed of the opponents' boats is greater than it was in the previous level.

Cleanup:
Nothing in particular.

—---

Test Case ID: ***TC014***

Purpose:
Its purpose is to check that if the boat gets out of the borders of its line the leg is lost. It is related to FR19.

Test Case Description:
This test should simulate that the boat goes outside of the lane and that consequently, the leg is immediately lost.

Pre-Conditions

Prerequisites:
A leg should have been started.

Test Data:
An instance of the type 'Boat' and the integer values of the position of the lines in the space should be provided as input values.

Test Steps

Step Description:

1. *The boat has to be moved to the right or to the left, using the method provided by the class 'Boat', until it overcome one of the values representing the position of the borders of the lane*
2. *It has to be checked that the leg is correctly lost, by checking that the value of the boolean variable in_the_start_menu is equal to 'true', which states that the player is now in the start menu*

Post-Conditions

Expected Outcome:
It is expected that the leg is lost, since the boat has touched the lanes.

Cleanup:
Nothing in particular.

Test Case ID: *TC015*

Purpose:
This test would check if losing a leg ends correctly, i.e. returning to the start menu. It is related to FR20.

Test Case Description:
A game should be started and lost, as an example by reaching the value of 0 for what regards the robustness of the boat. Then it has to be checked that the start menu is correctly opened.
.

Pre-Conditions

Prerequisites:
A leg has just been lost by the player.

Test Data:
The input values should be two instances of types 'Boat' and 'Rock' (any other type of obstacle would be good as well).

Test Steps

Step Description:

1. *A leg has to be started and the robustness of the boat has to be set to the value of 3 (or below)*
2. *A collision between the boat and the rock has to be executed, so that the boat will break down and the leg will be lost*
3. *It is checked that the value of the boolean variable in_the_start_menu is equal to 'true', which states that the player is now in the start menu*

Post-Conditions

Expected Outcome:
The start menu is displayed correctly after the player loses a leg.

Cleanup:
Nothing in particular.