Infinite-tux Game

Test Plan

In this document we'll provide a detailed document that describes functional and non-functional requirement, test types, identifying risks, test logistic, suspension and exist criteria, test environment, schedule planning, estimation, deliverables, and resources required to apply testing for the software product

Project
Version 4.0
SE401

Test Members

1. Fouad Majd Alkadri |218110075
2.Hisham Adnan | 218110087
3. Abdulaziz Alowain |219110119
4. Serry Sibaee | 218110246
5.Abdullah Abdul Mohimen |218110141

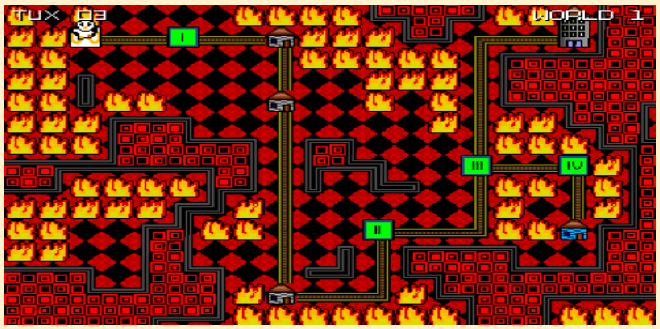
Revision History

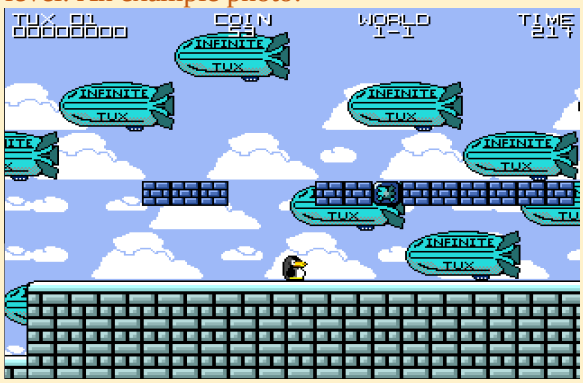
Date	Version	Authors	Notes
3/14/2022	**Version 1.0**	• Fouad Alkadri • Hisham Adnan • Abdulaziz Alowain • Serry Sibaee • Abdullah Abdul Mohimen	Things that are needed to be changed in the next version are the following: • Introduction about test plan, not a game. • Do test scope (more specific than objective). • Class diagrams remove the methods and do a System class diagram and a component diagram. • The activity diagram should reflect the scope. • The environment should include (IDE, Junit... etc). • Functional requirements should include the sources (website, tested the game...).
3/24/2022	**Version 2.0**	• Fouad Alkadri • Hisham Adnan • Serry Sibaee • Abdullah Abdul Mohimen • Abdulaziz Alowain	Problems that are assigned to version 1.0 are fixed in this version.
3/24/2022	**Version 3.0**	• Fouad Alkadri • Hisham Adnan • Serry Sibaee • Abdullah Abdul Mohimen • Abdulaziz Alowain	• Added Logical well written and consistent with the requirements Unit Test cases. • Added Logical well written and consistent with the requirements Integration test cases. • Added Logical well written and consistent with the requirements System (functional) test cases. • Added Clearly describe the techniques used to create the test cases.
5/6/2022	**Version 4.0**	• Fouad Alkadri • Hisham Adnan • Serry Sibaee • Abdullah Abdul Mohimen • Abdulaziz Alowain	• Implementing test cases • Tracing the requirement in improved way. • Some fixes in scope that reflect to our test plan. • Changed exist criteria. • Glossary modified.

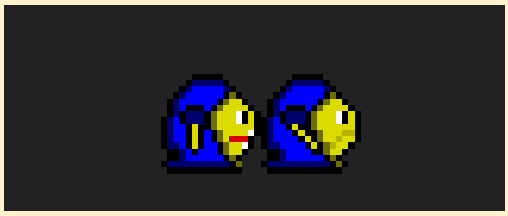
Table of Contents

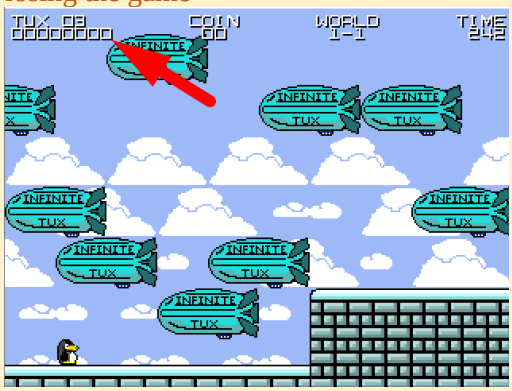
Phase 1:

1. Glossary:

Main menu	The main menu of the game should look like this: 
UI	The graphical User Interface
Level Selection Map	The area where the user selects the level he wants to play. It should look like this: 
Level navigation mario	

Arrow keys	The arrow keys on the keyboard: 
In-game	The actual gameplay experience where the user plays a level. An example photo: 
Cake	An item in the game that increases the size of the player. The cake looks like this:  After eating it the player increases in size (Large character): 

Enemies	**Red Koopa:**  **Green Koopa:**  **Goomba:**  **Spiky:** 
Chili	An item in the game that allows the player to throw fireballs once consumed.  After taking Chili mario becomes red 

Lives	The number of times the player can lose a level without losing the game 
Game over screen	This screen is displayed when the player loses a level without having any "live" left for him 
Invulnerability Time	The time at which Large Mario takes damage, and become small Mario (Normal one), there is like 3 seconds where he will not take any damage during his transformation,
Hurt Box	The invisible area around a character (Mario or Enemy), at which they take damage.

2. Introduction:

In the first phase of this project, we are going to describe the Infinite-Tux game. Then, we will establish the test plan which is going to be used – and updated- as our testing guide. We will describe the game using class diagrams, activity diagrams, functional and non-functional requirements. After that, we will establish our test plan for the game. After describing the game using UMLs and both functional and non-functional requirements, we will decide our testing approach, features to be tested, and features not to be tested. In addition to that, we have the schedule and tasks distribution along with the deliverables for this project.

3. References:

Sources of functional and non-functional requirements:

- https://libregamewiki.org/Infinite_Tux#Origin
- https://github.com/qbancoffee/infinite-tux
- Through playing the game.

4. Functional Requirement:

➢ 4.1 Main menu and UI requirements functional requirements:

4.1.1 The system shall allow the user to start the game by pressing S.
4.1.2 The system shall allow the user to minimize the game.
4.1.3 The system shall allow the user to close the game.
4.1.4 The system shall allow the user to exit by pressing ESC in any screen/view.
4.1.5 The game shall run sounds.

➢ 4.2 Level selection map functional requirements:

4.2.1 The system shall allow the user to navigate through the map using arrow keys.
4.2.2 The system shall allow the user to start a level using S.

4.3 In-game functional requirements:

4.3.1 The system shall allow the user to jump by pressing S.
4.3.2 The system shall allow the user to move right and left using arrow keys (4 right, right with obstacle, left, left with obstacle).
4.3.3 The system shall start the game level with a timer of 250 seconds.
4.3.4 The system shall allow the user to kill all kinds of enemies by jumping on them.
4.3.5 The system shall allow the user to double jump.
4.3.6 The system shall display “game over” when time reaches 0.
4.3.7 The system shall allow the user to jump into blocks.
4.3.8 The system shall spawn Mario with 3 lives.
4.3.9 The system shall spawn Mario without power up
4.3.10 The system shall spawn Mario without fireball
4.3.11 The system shall spawn Mario without coins
4.3.12 The system shall allow the Mario to have 3 “lives” before displaying the game over screen.
4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario.
4.3.14 If Mario collides with a Chili powerup, he should consume it.
4.3.15 If Mario collides with a Cake powerup, he should consume it.
4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
4.3.17 If Mario collides with an enemy, then he should get hurt.
4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
4.3.19 The system shall allow Red Mario to throw Fireballs by pressing A.
4.3.20 The system shall allow large Mario to break blocks when jumping into them.
4.3.21 The system shall make enemies who get hit by fireballs die.
4.3.22 If Mario jumps on Red Kubba then it should become a Shell.
4.3.23 If Mario jumps on Green Kubba then it should become a Shell.
4.3.24 If large/fire Mario gets damaged, he should become invulnerable.
4.3.25 If Mario collides with an enemy during vulnerability, he should not get damaged.
4.3.26 The system shall allow the user start spawn in the LevelScene after clicking S in MapScene.
4.3.27 The system shall allow the user to add 1 life after collecting 100 coins.

5. Non-Functional Requirement:

1. The system shall be compatible with Windows.
2. The system shall be compatible with macOS.
3. The system shall be compatible with LinuxOS.
4. The system shall be responsive to user input in less than 1 millisecond.
5. The system shall allow the user to alter between different programs in minimized mode.
6. The system shall allow the user to alter between different programs in full-screen mode.

6. Test Objectives:

The objective of these testing efforts is to verify the fulfillment of the specified features, and to detect bugs in the implementation of these features. All Features are derived from the game Mario 64. The **features that will be tested are following:**

7. Test Scope:

➢ 7.1 Features to be tested:

1. Mario gets hurt when colliding with an enemy.
2. Red/Green Kubba becomes a shell if Mario jumps on top of them.
3. Mario becomes Large with Cake powerup.
4. Mario becomes Red/Large Mario when consuming Chill powerup.
5. Fireballs that collide with enemies can kill them.
6. The player should be able to navigate from one level to another from the main map.
7. Players should be able to close the game by pressing ESC.
8. Players should be able to start a game by pressing S.
9. Up on collecting the 100th coin, Mario gains one more life.
10. After losing all Mario life's, Mario will respawn with three life's.
11. Mario respawns with zero powerups and coins.
12. Large Mario size decreases when hitting an enemy.
13. While Large Mario is decreasing in size, he should be temporarily invulnerable.

7.2 Feature not to be tested:

1. changing the soundtrack throughout the game.
2. Winning game scene.
3. Red Kuppa can’t fall from a platform.
4. green Kubba can fall from a platform.
5. Large Mario can break blocks.
6. Red Mario can shoot Fireballs.
7. Shell colliding with an enemy should kill him.
8. Red Mario that takes a powerup should just get a coin.
9. Large Mario that takes a Cake powerup should just get a coin.
10. Compatibility issues.
11. minimize the game.
12. Response time.
13. pause game.
14. Generating the main menu map randomly.
15. generating level maps randomly.
16. changing the background music accordingly.
17. Changing from one level to another.

8. System Description:

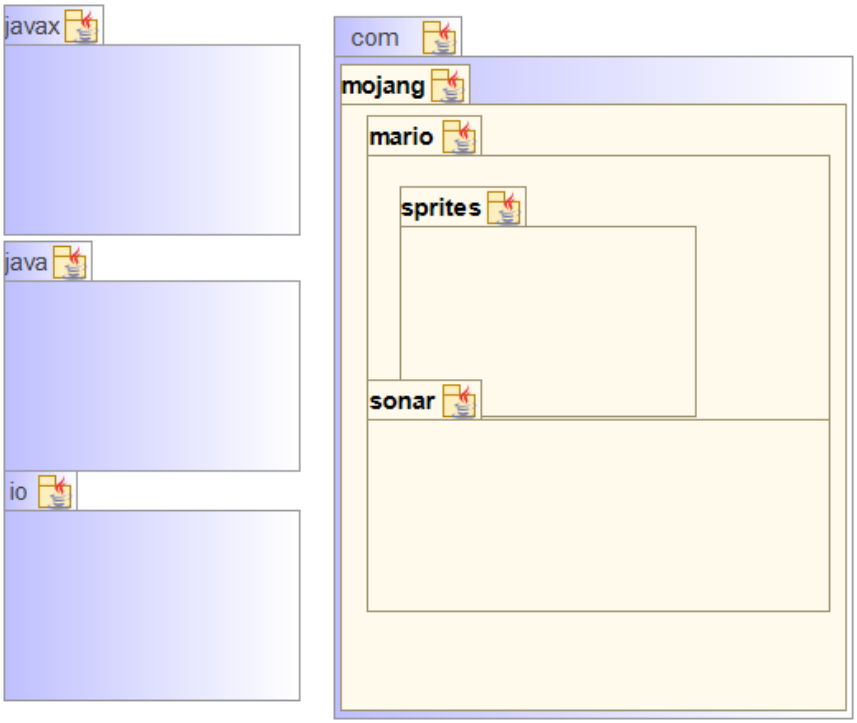
8.1 Package/Component Diagram:

The package diagram shows the arrangement and organization of model elements for a large-scale project.

8.2 Approach:

Describe the system in large-scale projects.

Figure 8.2.1 (Package Diagram without the connection between classes)



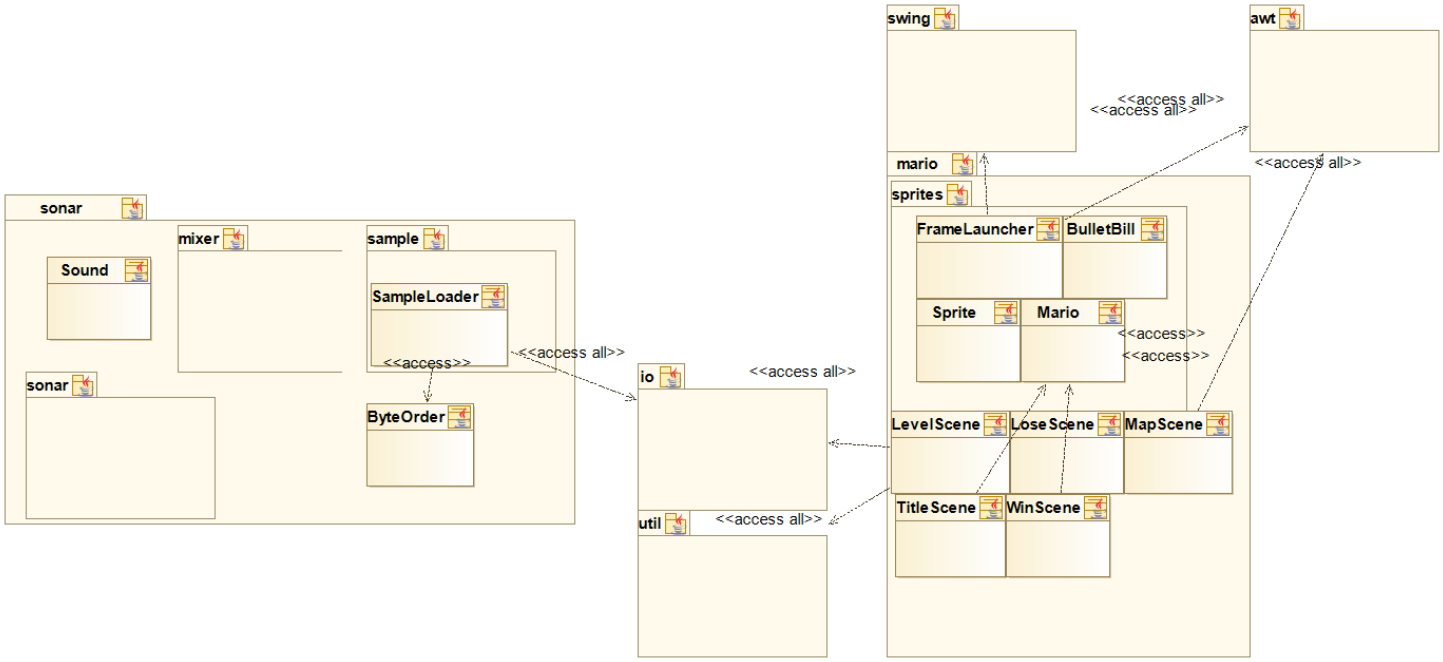
8.2.1 Description:

Figure 8.2.1:

This is a full abstract package view of the program as follows:

- full view of the main packages and the outside ones.

Figure 8.2.2 (Package Diagram with the connection between classes)



8.2.2 Description:

Figure 8.2.2:

The main connections in the main parts of the system are as the following:

- Sonar system for the sound
- Mario package for two things
  - spirits
  - levels views
- Also, it mentions the outside packages that have been used in the program for instance IO, util, swing, and awt.

8.3 Class Diagram:

The class diagram provides the overview and structure of a system, attributes, methods, and the relationships between different classes.

8.4 Approach:

Our goal is to represent the most important classes that show the important structure of the system, attribute, method, and relationships between classes. And to understand how our classes work related to what we're going to test the game.

8.5 Overview explanation:

The class diagram was chosen according to the following:

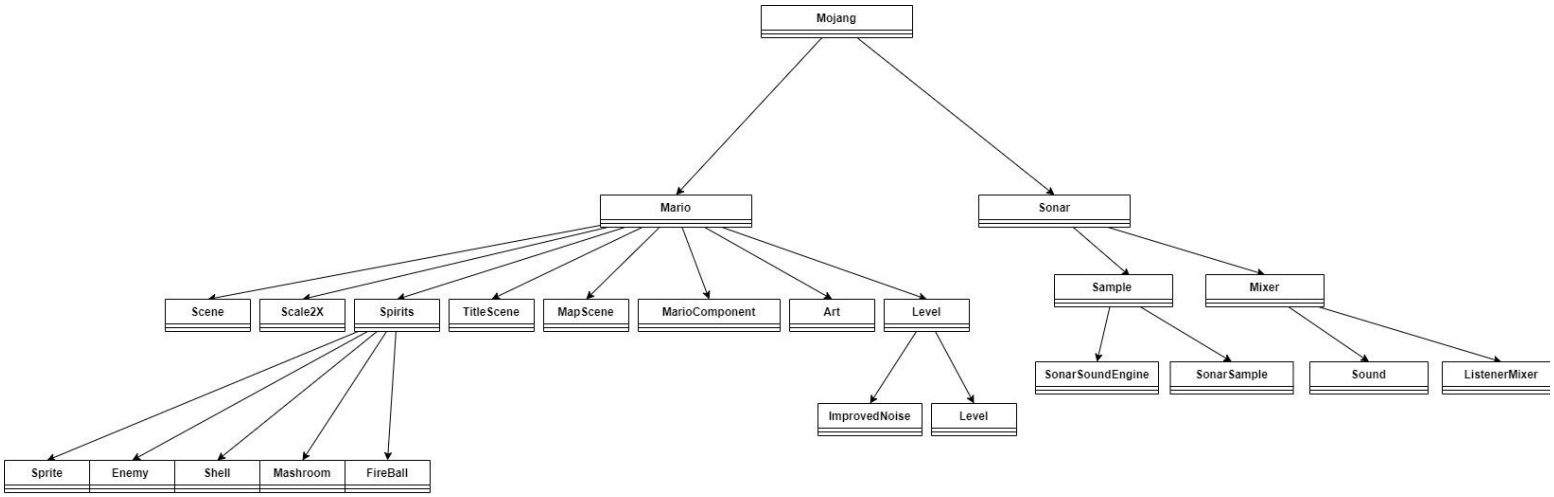
Choosing the three main classes in the project in three aspects:

- The graphical representation of the game (game component)
- The sound representation in the game (Sonar)
- The object in the game (Spirits) is as follow:
    - The spirts were shown in a generic way (not in detail) where it shows the main player and the enemy and the spirit main object (super).

Class Diagram:

The class diagram consists of the following Classes:

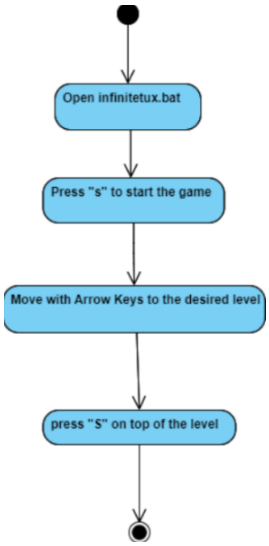
Mojang	Mario	Sonar	Scene.	Scale2X	Spirits	MarioComponent
TitleScene	Art	MapScene	Level	Sample	Mixer	Sprite
Enemy	Shell	Mushroom	Fireball	FireFlower	LevelScene	Improvednoise
Level	SonarSoundEngine	SonarExample.	Sound	Listener		



8.6 Activity Diagram:

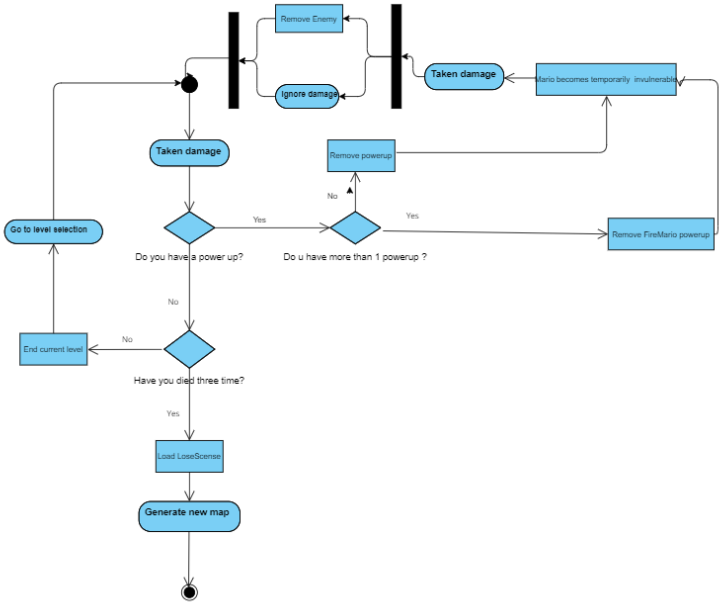
Activity Diagram describes the dynamic aspect of the system and that's reflected in our test scope.

Figure 8.6.1



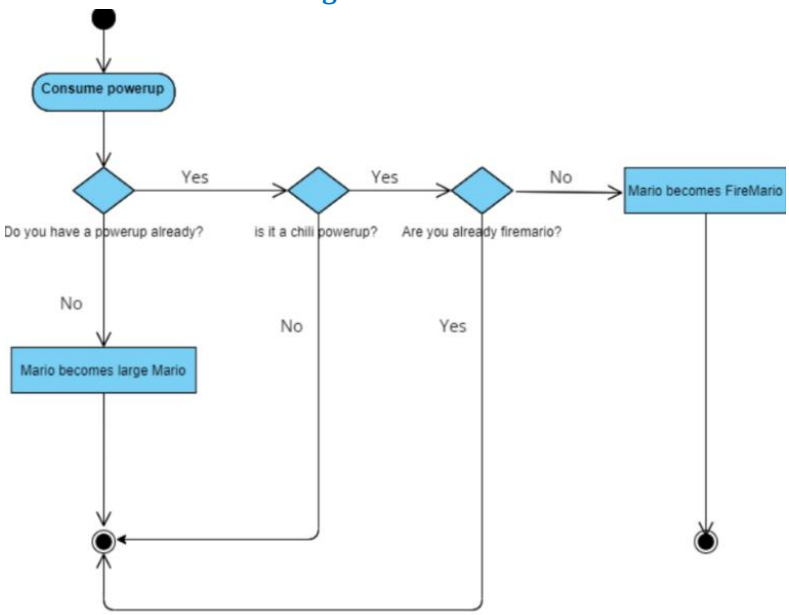
Start_Game Activity Diagram

Figure 8.6.2



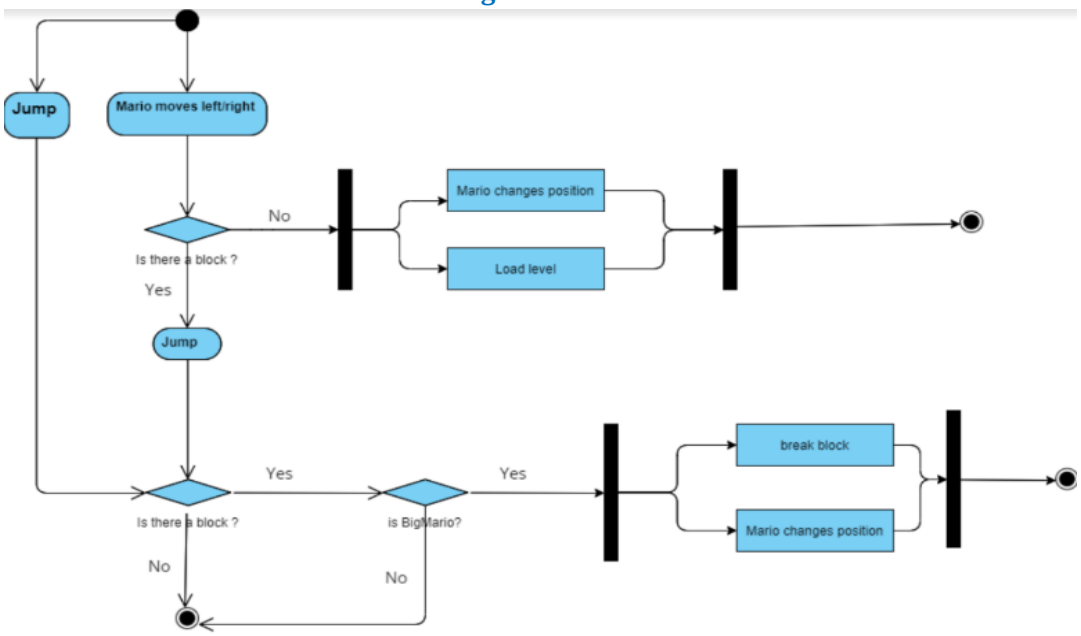
Taken_Damage Activity Diagram

Figure 8.6.3



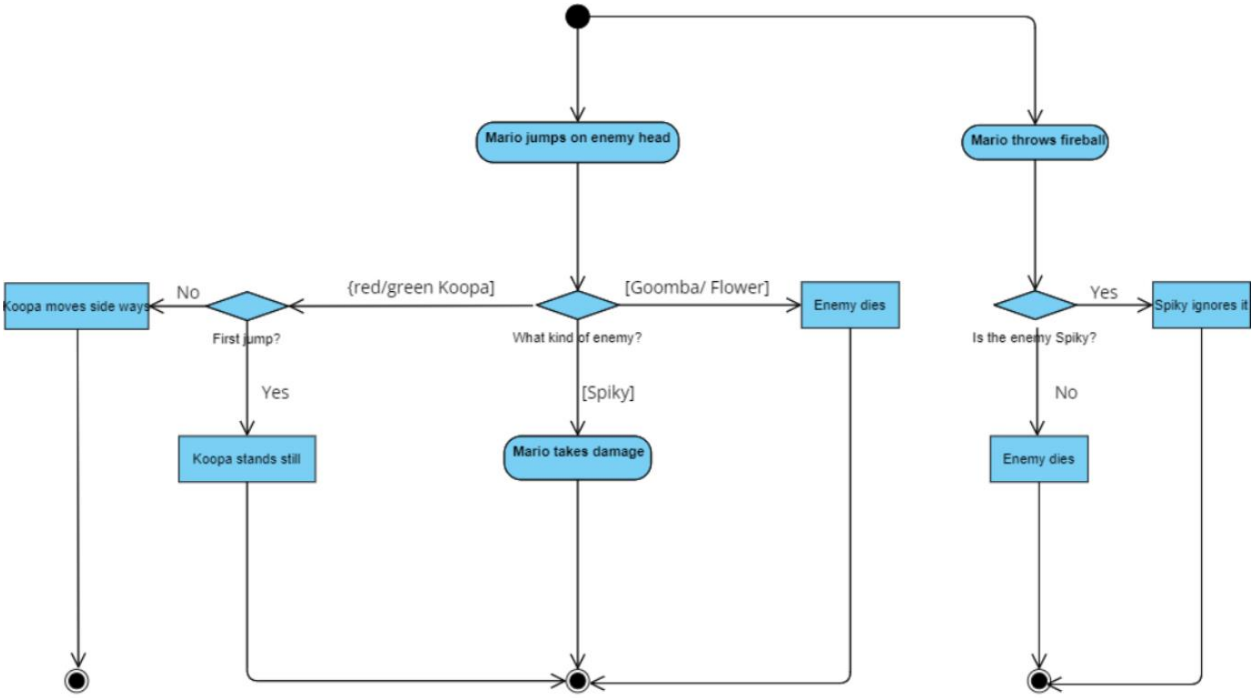
PowerUp Activity Diagram

Figure 8.6.4



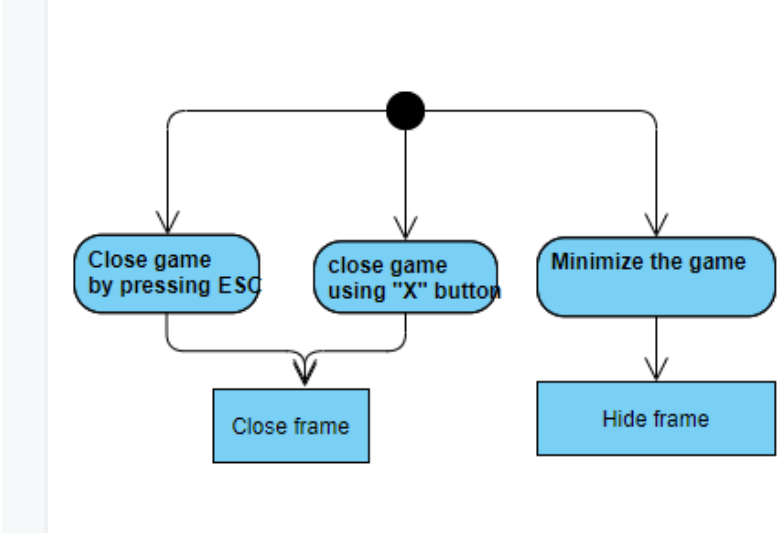
Movement Activity Diagram

Figure 8.6.5



Attack Enemy Activity Diagram

Figure 8.6.6



General-action Activity Diagram

9. Suspension and Exit Criteria:

Conditions to trigger suspension of test and analysis activities (e.g., an excessive failure rate) and conditions for restarting or resuming an activity.

- **9.1 Suspension criteria:**
  In the case that 55% of test cases fail, the testing effort should stop immediately, and a report of the failed test cases should be delivered to the development team to fix the problems.

- **9.2 Resumption criteria:**
  When 90% of failed test cases are fixed, the testing effort should resume. Some regression testing will be needed to see if there are any bugs injected while fixing the failed test cases.

- **9.3 Exit criteria:**
  When 95% of test cases are passed and 0% of test cases result in an error, the testing effort will be completed, and a report of all test cases should be made.

  Note: We have achieved this exit criteria, and a report has already been generated for the testing efforts.

10. Required system and Human Resources:

10.1 Human resources:

- **Test Manager:** Sets up time estimate of how long testing activities (E.g., Phase 1 of the project). Also, Manages the meeting between us and coordinates and distributes testing efforts among the team. Helps in monitoring test results.
- **Tester:** Builds and executes test suites. Also, contribute to time estimates and plan for testing.

10.2 System resources:

- Five workstations for all of the five team members.
- Java JDK downloaded to run .bat files and java code.
- CMD in the same file as .bat file to make it run.
- Testing tools such as Junit,BugSpot and Mockito were installed.
- IDE to help the testers in working with code more efficiently.

10.3. Test Items:

- Infinite tux game.
- Documentation of the game
  - **Link: https://libregamewiki.org/Infinite_Tux#Origin.**
- Source code
  - **GitHub Link: https://github.com/qbancoffee/infinite-tux.**

11. Test Schedule and Task Distribution:

	Phase one	Phase two	Phase three	Phase four
	Test Plan	Test Cases design	Test case Execution	Presentation
Expected Time	Monday 28-March	Monday 18-April	Monday 9-May	Sunday 15-May
Approximate time needed	10 hours	10 hours	2 hours	3 hours
Description	• Logical list requirement • Introduction • Description • Objectives • Scheduling	• Unit Test cases. • Integration test cases. • System (functional) test cases. • Techniques used	• Unit testing and integration correct and consistent • Final evaluation • Major functionality tested	• Demonstrating • Engaging choice of content • Contribution • Engaging style and lessons

Phase 1:

Task	Members	Estimate effort	Actual effort	Status
Logical List of Requiremnet	Hisham, Abdulaziz	5 Hours	3 Hours	Done
Introduction about the system	Hisham	1 Hour	1 Hours	Done
Descrption about the system	Hisham	1 Hour	1 Hours	Done
Clear Objectives of the test plan	Hisham	1 Hour	30 Min	Done
Schdualing and effort estimation	Serry	1 Hour	2 Hour	Done
Activity Diagrams	Abdullah	3 Hours	4 Hours	Done
Class Diagrams	Fouad, Serry	3 Hours	4 Hours	Done
Risks and mitigations	Fouad	3 Hours	2.5 Hours	Done
Test logistics	Hisham	1 Hour	2 Hours	Done
Scope pf testing	Hisham	2 Hours	4 Hours	Done
Types of testing that will be used	Abdulaziz	2 Hours	3 Hours	Done
Testing environment	Abdulaziz	2 Hours	2 Hours	Done
Suspension and Exit criteria	Abdullah	3 Hours	1.5 Hours	Done
Human and system resources	Abdullah	2 Hours	1 Hours	Done

Phase 2:

Task	Members	Estimate effort	actual effort	Status
Unit test cases 27→30	Hisham	4 Hours	8 Hours	Done
Integration test case 20	Hisham	1 Hour	30 Min	Done
System test cases 1→2	Hisham	2 Hours	1 Hours	Done
Modify contribution table	Hisham	2 Hours	30 Hours	Done
Unit test cases 23→26	Abdulaziz	2 Hours	5 Hours	Done
System test cases 3→4	Abdulaziz	2 Hours	3 Hours	Done
Modify test objective	Abdullah	2 Hours	2 Hours	Done
Modify features to be tested	Abdullah	2 Hours	3 Hours	Done
Unit test cases 1→14	Abdullah	4 Hours	16 Hours	Done
Integration test cases 1→15	Abdullah	6 Hours	12 Hours	Done
Documentation	Fouad	3 Hours	4 Hours	Done
Unit test cases 15→20	Fouad	2 Hours	15 Hours	Done
Integration test cases 16→19	Fouad	2 Hours	10 Hours	Done
System test case	Fouad	1 Hour	1 Hour	Done
Modify scheduling table	Serry	2 Hours	2 Hours	Done
CFG graph	Serry	2 Hours	2 Hours	Done
Unit test cases 21→22	Serry	1 Hour	1 Hour	Done

Phase 3:

Task	Members	Estimate effort	actual effort	Status
Implemented Unit test cases 1→ 4, 27→ 30.	Hisham	4 Hours	4 Hours	Done
Implemented Integration test cases 12, 20.	Hisham	6 Hours	15 Hours	Done
Implemented Sytsem test case 3	Hisham	1 Hour	1 Hour	Done
Modified scheduling and estimation table	Hisham	2 Hours	1 Hour	Done
Modified contribution table for Phase 3	Hisham	2 Hours	1 Hour	Done
Implemented Unit test cases 23 26.	Abdulaziz	4 Hours	6 Hours	Done
Implemented Integration test cases 1→ 5, 11	Abdulaziz	3 Hours	10 Hours	Done

Implemented Sytsem test case 5	Abdulaziz	2 Hours	1 Hour	Done
Modified test cases	Abdullah	2 Hours	4 Hours	Done
Implemented Unit test cases 5□14	Abdullah	6 Hours	11 Hours	Done
Implemented Sytsem test case 4	Abdullah	2 Hours	1 Hour	Done
documentation and formation	Fouad	3 Hours	4 Hours	Done
Implemented Unit test cases 15□20	Fouad	4 Hours	12 Hours	Done
Implemented Integration test cases 16□19	Fouad	9 Hours	10 Hours	Done
Implemented Sytsem test case 2	Fouad	2 Hours	1 Hour	Done
Modified testing techniques and Risks table	Fouad	2 Hours	1 Hour	Done
Implemented Unit test cases 21□22	Serry	2 Hours	1 Hour	Done
Implemented Integration test cases 6□10, 13→ 1	Serry	4 Hours	12 Hours	Done
Implemented Sytsem test case 1	Serry	2 Hours	1 Hour	Done
Modified class diagram	Serry	3 Hours	2 Hours	Done

12. Test Types

In our project, we will first perform Black-box testing by playing the game. We will check if the features to be tested are working correctly. If a feature is not working correctly then we will perform White-box testing on the parts of the code that we suspect are causing this failure. We will keep testing the code (White-box) until we find the cause of the failure.

13. Test Environment

Our test environment will be a single computer with the following features:

- Runs windows 10 (checking compatibility with windows is a feature to be tested).
- Has at least 4GB RAM (to run the game smoothly).
- No internet connection is required (the game is offline).
- Using Eclipse IDE that's an open platform for developers and mostly used. And we'll use the extensions that are integrated with Eclipse for the following:
    - Junit.
    - Spotbugs.

14. Risk and Rational Mitigations

Risk is the futures for an uncertain event with a probability of occurrence and potential for loss.
Note: Whenever the risk happens, it becomes an "issue".

Risk ID	Risk	Probability	Influence	Mitigation

R.1	The project timeline for testing the game is not enough to finish it up.	**High**	**Medium**	• Set a schedule time plan with the consideration of the tester who will participate and add expanded time for any concern of task. • Set test priority for each of the test activities.
R.2	Lack of skill with the team members that are dealing with test cases.	**Medium**	**Low**	• Plan for training the testers who lack skills with the goal of increasing the efficiency, and performance of the testers. • Make an Invitation for an experienced tester that's capable of supporting the team with any issues or concerns.
R.3	Lack of communication between the team members might affect either productivity, creativity, or workflow of the task	**High**	**High**	• Encourage the tester in his required task and inspire them for a better effort. • Plan the tasks and assign them to work as a team, so that makes it impossible to work independently.
R.4	The testers that have designed test cases, tasks, and test activities are not well-defined or understood.	**Medium**	**High**	• Plan for meeting every week to make an overview about what they did in the testing project or the tasks that are assigned to the tester, so everyone will catch up on what's is happening. And to fix any inappropriate task that has been done through the meeting.
R.5	The test cases that are used are frequently used.	**Medium**	**High**	• Make independent testing, so that will help us to provide unique test cases that'll be used.
R.6	The requirement is not traced well with our test cases.	**Low**	**High**	• Trace each requirement that will satisfy our test cases.
R.7	Some of the functional requirement can not be applied (not realistic)	**Medium**	**Low**	• Modify the functional requirement that can not be applied. • Remove the functional requirement.

				• Change the test scope that the function requirement will be not tested.

15. Test Logistics:

The main aspects of testing will be covered in this STLC by the following testers:

- Hisham Adnan.
- Abdulaziz Alowain.
- Abdullah Abdul Mohimen.
- Fouad Alkadri.
- Serry Sibaee.

16. Test Deliverables:

Our test deliverables before testing:

- Test Plan which will provide the overall plan for the testing work to be done.

Our test deliverables during testing:

- Test Scripts if a failure is found during Black-box testing and a test script was written to find the failure.

Our test deliverables after testing:

- Test report which will give a summary of our testing process.
- Defect report which will give a summary of the defects we found.

17. Task Distribution:

Phase 1

Name	Tasks
Hisham Adnan	• list of important functional and non-functional requirements • Introduction that describes the aim of the document and the topics that will be covered in the document • Identify features to be tested and features not to be tested • Test logistics are clearly and logically identified

	• Test Objective that clearly describes the aim of the testing process
Abdulaziz Alowin	• list of important functional and non-functional requirements • Identify features to be tested and features not to be tested • Identify test types with consideration to system type and requirements • Test Objective that clearly describes the aim of the testing process • The test environment required to execute the planned tests is clearly described • Realistic test deliverables are listed
Abdullah Abdul Mohimen	• Activity diagrams that reflect the classes and activities of the system • Suspension and Exit criteria are rationally specified • Specify required system and human resources
Fouad Alkadri	• Complete Class diagrams • Identify and describe risks and rational mitigations • Took care of documentation and formation
Serry Sibaee	• Complete Class diagrams. • Rational scheduling estimation is provided for all testing tasks

Phase 2

Name	Tasks
Hisham Adnan	• Unit test cases 27→30. • Integration test cases 20. • System test case 1, 2. • Modified contribution table for Phase 2.
Abdulaziz Alowin	• Unit test cases 23□26.

	• System test case 3, 4.
Abdullah Abdul Mohimen	• Modified test objective of phase 1. • Modified features to be tested and features not to be tested. • Unit test cases 1□14. • Integration test cases 1□15.
Fouad Alkadri	• Took care of documentation and formation. • Unit test cases 15□20. • Integration test cases 16□19. • System test case 5.
Serry Sibaee	• Modified scheduling estimation of phase 1. • Made control flow graph. • Unit test cases 21□22.

Phase 3

Name	Tasks
Hisham Adnan	• Implemented Unit test cases 1→ 4, 27→ 30. • Implemented Integration test cases 12, 20. • Implemented System test case 3.

	• Modified scheduling and estimation table. • Modified contribution table for Phase 3.
Abdulaziz Alowin	• Implemented Unit test cases 23 26. • Implemented Integration test cases 1→ 5, 11. • Implemented System test case 5
Abdullah Abdul Mohimen	• Modified test cases. • Implemented Unit test cases 5 14. • Implemented System test case 4
Fouad Alkadri	• Took care of documentation and formation. • Implemented Unit test cases 15 20. • Implemented Integration test cases 16 19. • Implemented System test case 2 • Modified testing techniques and Risks table.
Serry Sibaee	• Implemented Unit test cases 21 22. • Implemented Integration test cases 6 10, 13→ 15. • Implemented System test case 1 • Modified class diagram

Phase 2:

18. Test Cases

18.1 Unit test cases:

ID	UTC1

Functional Req	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it.
TC Description	verify that Chili powerup increases Mario Size.
Technique Used	**Decision Table Testing. (Black-box technique)**
Item to be tested	Class: Mario, Method:setLarge()
Pre-condition	1- Launch the game 2- Player starts a level 3- Mario is not dead 4- Mario is not Large 5- Mario is not Red Mario 6- Chili powerup spawns
Steps	1- Mario walks into Mushroom powerup 2- Mario's increased size animation should start.
Input	1- Large = False; 2- Fire = True;
Expected Result	Mario.large = True; Mario.fire=True;

ID	**UTC2**
Functional Req	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it.
Technique Used	**Decision Table Testing. (Black-box technique)**
TC Description	verify that Chili powerup increases Mario Size.
Item to be tested	Class: Mario, Method:setLarge()
Pre-condition	1- Launch the game 2- Player starts a level 3- Mario is not dead 4- Mario is not Large 5- Mario is not Red Mario 6- Chili powerup spawns
Steps	1- Mario walks into Mushroom powerup 2- Mario's increased size animation should start.
Input	1- isLarge = true; 2- isFire = true;
Expected Result	Mario.large = true; Mario.fire=true;

ID	**UTC3**
Functional Req	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.

TC Description	Verify that Cake powerup increases Mario size.
Technique Used	**Decision Table Testing. (Black-box technique)**
Item to be tested	Class: Mario, Method:setLarge()
Pre-condition	1- Launch the game 2- Player starts a level 3- Mario is not dead 4- Mario is not large 5- Mario is not Red Mario 6- Cake powerup spawns
Steps	1- Mario consumes the Cake powerup 2- Mario goes through an increase in animation.
Input	1- isLarge = True; 2- isFire = False;
Expected Result	Mario.large = True; Mario.fire = False;

ID	**UTC4**
Functional Req	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
TC Description	Verify that Cake powerup increases Mario size.
Technique Used	**Decision Table Testing. (Black-box technique)**
Item to be tested	Class: Mario, Method:setLarge()
Pre-condition	1- Launch the game 2- Player starts a level 3- Mario is not dead 4- Mario is not large 5- Mario is not Red Mario 6- Cake powerup spawns
Steps	1- Mario consumes the Cake powerup 2- Mario goes through an increase in animation.
Input	1- isLarge = False; 2- isFire = False;
Expected Result	1- isLarge = false; 2- isFire = false;

ID	**UTC5**
Functional Req	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
TC Description	verifying that Large Mario decreases in size after getting damaged by an enemy.

Technique Used	Decision Table Testing. (Black-box technique)
Item to be tested	Class: Mario, Method:getHurt()
Pre-condition	1- launch the game 2- Player starts a level 3- enemy spawns 4- Mario is Large Mario 5- Mario is not Red Mario
Steps	1- Mario walks into the enemy 2- Mario collides with an enemy 3- Mario becomes temporarily invulnerable 4- Mario decreasing size animation starts
Input	1- large = True; 2- InvulnerableTime =0; 3- word.pause = False 4- deathTime=0; 5- fire = false;
Expected Result	Mario.large =False; Mario.fire=False;

ID	UTC6
Functional Req	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
TC Description	verifying that Large Mario decreases in size after getting damaged by an enemy.

Technique Used	**Decision Table Testing. (Black-box technique)**
Item to be tested	Class: Mario, Method:getHurt()
Pre-condition	1- launch the game 2- Player starts a level 3- enemy spawns 4- Mario is Large Mario / Mario is Red Mario
Steps	1- Mario walks into the enemy 2- Mario collides with an enemy 3- Mario becomes temporarily invulnerable 4- Mario decreasing size animation starts
Input	1- large = False; 2- InvulnerableTime =0; 3- word.pause = False 4- deathTime=0; 5- fire = false;
Expected Result	Mario.large = False; Mario.fire = False; deathTime=1;

ID	**UTC7**
Functional Req	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
TC Description	verifying that Large Mario decreases in size after getting damaged by an enemy.

Technique Used	**Decision Table Testing. (Black-box technique)**
Item to be tested	Class: Mario, Method:getHurt()
Pre-condition	1- launch the game 2- Player starts a level 3- enemy spawns 4- Mario is Large Mario / Mario is Red Mario
Steps	1- Mario walks into the enemy 2- Mario collides with an enemy 3- Mario becomes temporarily invulnerable 4- Mario decreasing size animation starts
Input	1- large = True; 2- InvulnerableTime =0; 3- word.pause =True; 4- deathTime=0; 5- fire = false;
Expected Result	Mario.large = True; Mario.fire = False; deathTime=0;

ID	**UTC8**
Functional Req	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
TC Description	verifying that Large Mario decreases in size after getting damaged by an enemy.

Technique Used	Decision Table Testing. (Black-box technique)
Item to be tested	Class: Mario, Method:getHurt()
Pre-condition	1- launch the game 2- Player starts a level 3- enemy spawns 4- Mario is Large Mario /Mario is Red Mario
Steps	1- Mario walks into the enemy 2- Mario collides with an enemy 3- Mario becomes temporarily invulnerable 4- Mario decreasing size animation starts
Input	1- large = false; 2- InvulnerableTime =0; 3- word.pause =False; 4- deathtime=1; 5- fire = false;
Expected Result	Mario.large = False; Mario.fire = False; deathTime=1;

ID	UCT9
Function Req	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
TC Description	verifying that Large Mario decreases in size after getting damaged by an enemy.

Technique Used	**Decision Table Testing. (Black-box technique)**
Item to be tested	Class: Mario, Method:getHurt()
Pre-condition	1- launch the game 2- Player starts a level 3- enemy spawns 4- Mario is Large Mario / Mario is Red Mario
Steps	1- Mario walks into the enemy 2- Mario collides with an enemy 3- Mario becomes temporarily invulnerable 4- Mario decreasing size animation starts
Input	1- large = true; 2- InvulnerableTime =0; 3- word.pause =True; 4- deathTime=0; 5- fire = True;
Expected Result	Mario.large = True; Mario.fire=true;

ID	**UTC10**
Functional Req	4.3.21 The system shall make enemies who get hit by fireballs die.
TC Description	Verifying that fireballs kill enemies
Technique used	**BVA, and Equivalence partitioning. (Black-box techniques)**
Item to be tested	Class: Enemy, Method:fireBallCollideCheck();
Pre-condition	1- launch the game 2- Player starts a level 3- Enemy Spawn 4- Mario is Red Mario 5- Enemy is not Spiky (spiky doesn't take Fireball damage)
Steps	1- Mario fires the fireball 2- Fireball hits the enemy 3- The Fireball collides with the enemy.
Input	xD in {-4,4} note: width of both fireball and Enemy has width 4 yD in {-24,8} enemy.type =1 (Green Koopa)
Expected Result	True.
Post-condition	Enemy killed

ID	**UTC11**
Functional Req	4.3.21 The system shall make enemies who get hit by fireballs die.
TC Description	Verifying that fireball kill enemies
Technique used	**BVA, and Equivalence partitioning. (Black-box techniques)**

Item to be tested	Class: Enemy, Method:fireBallCollideCheck();
Pre-condition	1- launch the game 2- Player starts a level 3- Enemy Spawn 4- Mario is Red Mario 5- Enemy is not Spiky (spiky doesn't take Fireball damage)
Steps	1- Mario fires the fireball 2- Fireball hits the enemy 3- The Fireball collides with the enemy.
Input	xD = 4 yD = 8 enemy.type =1 (Green Koopa)
Expected Result	True
Post-condition	Enemy killed

ID	**UTC12**
Functional Req	4.3.21 The system shall make enemies who get hit by fireballs die.
TC Description	Verifying that fireball kill enemies
Technique used	**BVA, and Equivalence partitioning. (Black-box techniques)**
Item to be tested	Class: Enemy, Method:fireBallCollideCheck();
Pre-condition	1- launch the game 2- Player starts a level 3- Enemy Spawn 4- Mario is Red Mario 5- Enemy is not Spiky (spiky doesn't take Fireball damage)
Steps	1- Mario fires the fireball 2- Fireball hits the enemy 3- The Fireball collides with the enemy.
Input	xD = 5 yD = 9 enemy.type =1 (Green Koopa)
Expected Result	False
Post-condition	Enemy Alive

ID	**UTC13**
Function Req	4.3.21 The system shall make enemies who get hit by fireballs die.

TC Description	Verifying that fireball kill enemies
Technique used	**BVA, and Equivalence partitioning. (Black-box techniques)**
Item to be tested	Class: Enemy, Method:fireBallCollideCheck();
Pre-condition	1- launch the game 2- Player starts a level 3- Enemy Spawn 4- Mario is Red Mario 5- Enemy is not Spiky (spiky doesn't take Fireball damage)
Steps	1- Mario fires the fireball 2- Fireball hits the enemy 3- The Fireball collides with the enemy.
Input	xD = -4 yD = - 24 enemy.type =1 (Green Koopa)
Expected Result	true
Post-condition	Enemy killed

ID	**UTC14**
Functional Req	4.3.21 The system shall make enemies who get hit by fireballs die.
TC Description	Verifying that fireball kill enemies
Technique used	**BVA, and Equivalence partitioning. (Black-box techniques)**
Item to be tested	Class: Enemy, Method: fireBallCollideCheck();
Pre-condition	1- launch the game 2- Player starts a level 3- Enemy Spawn 4- Mario is Red Mario 5- Enemy is not Spiky (spiky doesn't take Fireball damage)
Steps	1- Mario fires the fireball 2- Fireball hits the enemy 3- The Fireball collides with the enemy.
Input	xD= -5 yD = -25 enemy.type =1 (Green Koopa)
Expected Result	false
Post-condition	Enemy alive

ID	**UTC15**
Functional Req	4.3.24 If large/fire Mario gets damaged, he should become invulnerable.
TC Description	Verifying that Mario if he was in a large or fire state when Mario gets damaged, he will be able to absorb the shock after the he got

	damaged.
Technique used	**Statement coverage (structural testing: control flow based).**
Item to be tested	Class: Fireball, Method:isBlocking();
Pre-condition	1. Launch the game 2. Player starts a level 3. Enemy spawn 4. Mario eats red pepper or cake 5. Mario become large/fire state
Steps	1. Mario runs through the enemy. 2. Mario gets damaged by the enemy.
Input	_x, _y, xa, ya
Expected Result	return blocking;
Post-condition	Mario Become small

ID	**UTC16**
Function Req	4.3.27 The system shall allow the user to add 1 life after collecting 100 coins.
TC Description	Verifying that Mario while he's collecting coins and while he's collecting coins if he reaches 100 coins, he'll get 1 extra life.
Technique used	**EP- Equivalence Partition (black-box testing)**
Item to be tested	Class: Mario, Method: getCoin();
Pre-condition	1. Launch the game. 2. Player starts a level. 3. Enemy spawn. 4. Mario starts collecting coins and hitting block.
Steps	1. Mario goes through coins. 2. jumps and hit block to get coins. 3. Mario collects 100 coins.

Input	coins=0;
	coins=10;
	coins=19;

Expected Result	coins++; , coins=1;
	coins++; , coins=11;
	coins++; , coins=20;

Post-condition	Mario gets extra life.

ID	**UTC17**
Function Req	4.3.27 The system shall allow the user to add 1 life after collecting 100 coins.
TC Description	Verifying that Mario while he's collecting coins and while he's

	collecting coins if he reaches 100 coins, he'll get 1 extra life.
Technique used	**EP- Equivalence Partition (black-box testing)**
Item to be tested	Class: Mario, Method:getCoin();
Pre-condition	1. Launch the game. 2. Player starts a level. 3. Enemy spawn. 4. Mario starts collecting coins and hitting block.
Steps	1. Mario goes through coins. 2. jumps and hit block to get coins. 3. Mario collects 100 coins.

Input	coins=20;
	coins=29;
	coins=39;

Expected Result	coins++; , coins=21;
	coins++; , coins=30;
	coins++; , coins=40;

Post-condition	Mario gets extra life.

ID	**UTC18**
Functional Req	4.3.27 The system shall allow the user to add 1 life after collecting 100 coins.
TC Description	Verifying that Mario while he's collecting coins and while he's collecting coins if he reaches 100 coins, he'll get 1 extra life.
Technique used	**EP- Equivalence Partition (black-box testing)**
Item to be tested	Class: Mario, Method:getCoin();
Pre-condition	1. Launch the game. 2. Player starts a level. 3. Enemy spawn. 4. Mario starts collecting coins and hitting block.
Steps	1. Mario goes through coins. 2. jumps and hit block to get coins. 3. Mario collects 100 coins.

Input	coins=40;
	coins=50;
	coins=59;

Expected Result	coins++; , coins=41;
	coins++; , coins=51;
	coins++; , coins=60;

Post-condition	Mario gets extra life.

ID	**UTC19**
Functional Req	4.3.27 The system shall allow the user to add 1 life after collecting 100 coins.
TC Description	Verifying that Mario while he's collecting coins and while he's collecting coins if he reaches 100 coins, he'll get 1 extra life.

Technique used	**EP- Equivalence Partition (black-box testing)**
Item to be tested	Class: Mario, Method:getCoin();
Pre-condition	1. Launch the game. 2. Player starts a level. 3. Enemy spawn. 4. Mario starts collecting coins and hitting block.
Steps	1. Mario goes through coins. 2. jumps and hits block to get coins. 3. Mario collects 100 coins.

Input	coins=60;
	coins=70;
	coins=79;

Expected Result	coins++; , coins=61;
	coins++; , coins=71;
	coins++; , coins=80;

Post-condition	Mario gets extra life.

ID	**UTC20**
Functional Req	4.3.27 The system shall allow the user to add 1 life after collecting 100 coins.
TC Description	Verifying that Mario while he's collecting coins and while he's collecting coins if he reaches 100 coins, he'll get 1 extra life.
Technique used	**EP- Equivalence Partition (black-box testing)**
Item to be tested	Class: Mario, Method:getCoin();
Pre-condition	1. Launch the game. 2. Player starts a level. 3. Enemy spawn. 4. Mario starts collecting coins and hitting block.
Steps	1. Mario goes through coins. 2. jumps and hit block to get coins. 3. Mario collects 100 coins.

Input	coins=80;
	coins=90;
	coins=99;

Expected Result	coins++; , coins=81;
	coins++; , coins=91;
	coins++; , coins=0; , invokes get1Up();

Post-condition	Mario gets extra life.

ID	**UTC21**
Functional Req	4.1.4 The system shall allow the user to exit by pressing ESC in any screen/view.
TC Description	press ESC key the program closes immediately

Technique Used	**statement coverage**
Item to be tested	Mario_Component class toggleKey() method
Pre-condition	1. launch the game 2. start the game
Steps	3. press ESC any time after the game launched
Input	ESC key
Expected Result	The window closes immediately

ID	**UTC22**
Functional Req	4.2.1 The system shall allow the user to navigate through the map using arrow keys.
TC Description	Verifying that Mario can move in the map using arrow keys.
Technique Used	**Equivalence Partitioning (Black-box Technique)**
Item to be tested	Class: MapScene; Method: tryWalking()
Pre-condition	1. The game is launched. 2. The player is in the map 3. The target tile is a road, and the Mario is not on a level tile.
Steps	1. Player enters into the map. 2. Mario moves to the current tile. 3. Player attempts to move the Mario to target tile.
Input	Arrow key is pressed.
Expected Result	The Mario moves to the target road tile
Post-condition	N/A

ID	**UTC23**
Functional Req	4.2.1 The system shall allow the user to navigate through the map using arrow keys.
TC Description	Verifying that Mario can move in the map using arrow keys.
Technique Used	**Equivalence Partitioning (Black-box Technique)**

Item to be tested	Class: MapScene; Method: tryWalking()
Pre-condition	1.The game is launched. 2. The player is in the map 3. The target tile is not a road or level.
Steps	1.Player enters into the map. 2.Mario moves to the current tile. 3.Player attempts to move the Mario to target tile.
Input	Arrow key is pressed.
Expected Result	The Mario stays in its current tile
Post-condition	N/A

ID	**UTC24**
Functional Req	4.2.1 The system shall allow the user to navigate through the map using arrow keys.
TC Description	Verifying that Mario can move in the map using arrow keys.
Technique Used	**Equivalence Partitioning (Black-box Technique)**
Item to be tested	Class: MapScene; Method: tryWalking()
Pre-condition	1. The game is launched. 2. The player is in the map 3. The target tile is a level.
Steps	1. Player enters into the map. 2. Mario moves to the current tile. 3. Player attempts to move the Mario to target tile.
Input	Arrow key is pressed.
Expected Result	The Mario moves to the target level tile
Post-condition	N/A

ID	**UTC25**
Functional Req	4.2.1 The system shall allow the user to navigate through the map using arrow keys.
TC Description	Verifying that Mario can move in the map using arrow keys.
Technique Used	**Equivalence Partitioning (Black-box Technique)**
Item to be tested	Class: MapScene; Method: tryWalking()

Pre-condition	1. The game is launched. 2. The player is in the map 3. The target tile is a road, and the current tile is a level tile. 4. The Mario has not unlocked this part of the map.
Steps	1. Player enters into the map. 2. Mario moves to the current tile. 3. Player attempts to move the Mario to target tile.
Input	Arrow key is pressed.
Expected Result	The Mario stays in its current tile
Post-condition	N/A

ID	**UTC26**
Functional Req	4.2.1 The system shall allow the user to navigate through the map using arrow keys.
TC Description	Verifying that Mario can move in the map using arrow keys.
Technique Used	**Equivalence Partitioning (Black-box Technique)**
Item to be tested	Class: MapScene; Method: tryWalking()
Pre-condition	1.The game is launched. 2.The player is in the map 3.The target tile is a road, and the current tile is a level tile. 4.The Mario has unlocked this part of the map.
Steps	1. Player enters into the map. 2. Mario moves to the current tile. 3. Player attempts to move the Mario to target tile.
Input	Arrow key is pressed.
Expected Result	The Mario moves to the target road tile
Post-condition	N/A

ID	**UTC27**
Function Req	4.3.8 The system shall spawn Mario with 3 lives
TC Description	When Mario dies - loses all his lives- his lives should be reseted to 3
Technique used	**Black box → Cause and effect**
Item to be tested	Class: Mario Method: resetStatic()

Pre-conditions	1. Game should be running 2. Mario should be on a level 3. Mario should have 1 life left 4. Mario shouldn't have or take any power up 5. Mario shouldn't collect more than 99 coins (He will not gain extra life)
steps	1. Move towards an enemy 2. Hit the enemy and die to him
Input	Method invoked, which is done by hitting an enemy with 1 life left only
Expected Result	Mario's number of lives should be changed to 3
Post-condition	a new random level map should be generated

ID	**UTC28**
Functional Req	4.3.9 The system shall spawn Mario without power up
TC Description	When Mario dies - loses all his lives- he should spawn without a cake power up
Technique used	**Black box → Cause and effect**
Item to be tested	Class: Mario Method: resetStatic()
Pre-conditions	1. Game should be running 2. Mario should be on a level 3. Mario should have 1 life left 4. Mario shouldn't have or take any power up 5. Mario shouldn't collect more than 99 coins (He will not gain extra life)
steps	1. Move towards an enemy 2. Hit the enemy and die to him
Input	Method invoked, which is done by hitting an enemy with 1 life left only
Expected Result	Spawn without cake power up (No change should occur)
Post-condition	a new random level map should be generated

ID	**UTC29**
Function Req	4.3.10 The system shall spawn Mario without fireball
TC Description	When the player loses (died three times), Mario should spawn again without a fireball power up
Technique used	**Black box → Cause and effect**
Item to be tested	Class: Mario Method: resetStatic()

ID	UTC29
Function Req	4.3.10 The system shall spawn Mario without fireball
TC Description	When the player loses (died three times), Mario should spawn again without a fireball power up
Pre-conditions	1. Game should be running 2. Mario should be on a level 3. Mario should have 1 life left 4. Mario shouldn't have or take any fireball power up 5. Mario shouldn't have or take cake power up 6. Mario shouldn't collect more than 99 coins (He will not gain extra life)
steps	1. Move towards an enemy 2. Hit the enemy and die to him
Input	Method invoked, which is done by hitting an enemy with 1 life left only
Expected Result	Spawn without fireball power up (No change should occur)
Post-condition	a new random level map should be generated

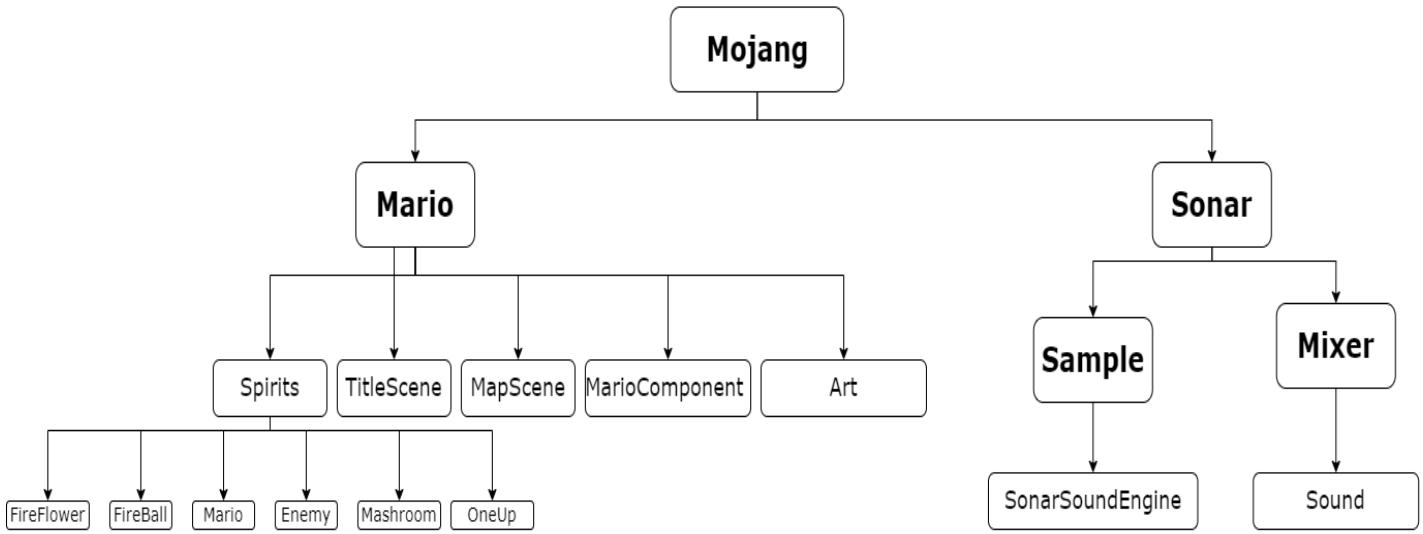
ID	UTC30
Function Req	4.3.11 The system shall spawn Mario without coins
TC description	When the player loses (died three times), Mario should spawn again without any coins with him
Technique used	**Black box → Cause and effect**
Item to be tested	Class: Mario Method: resetStatic()
Pre-conditions	1. Game should be running. 2. Mario should be on a level. 3. Mario should have 1 life left. 4. Mario shouldn't have to take cake power up. 5. Mario shouldn't collect more than 99 coins (He will not gain extra life)
steps	1. Move towards an enemy. 2. Hit the enemy and die to him.
Input	Method invoked, which is done by hitting an enemy with 1 life left only
Expected Result	Spawn without coins
Post-condition	a new random level map should be generated

➢ 18.2 Integration test cases:

Integration testing are components that are combined to form a complete system. And integration testing focuses on as the following:

- Interfaces between modules (or components).
- Integrated functional features.

Control Flow Diagram (CFG)



➢ Traceability Matrix:

Requirement (4. before each from the report)	Test Cases
Main menu and UI requirements	
4.1.1	ITC20 STC5
4.1.2	
4.1.3	
4.1.4	UTC21

	STC5
4.1.5	
Level selection map	
4.2.1	UTC22,UTC23,UTC24,UTC25,UTC26 STC4
4.2.2	STC4
In-game	
4.3.1	
4.3.2	STC1,STC2
4.3.3	
4.3.4	STC1
4.3.5	
4.3.6	STC2
4.3.7	
4.3.8	UTC27
4.3.9	UTC28
4.3.10	UTC29
4.3.11	UTC30
4.3.12	STC1,STC2
4.3.13	UTC1,UTC2 ITC1,ITC2,ITC3,ITC4,ITC5 STC3

4.3.14	UTC1,UTC2 ITC1,ITC2,ITC3,ITC4,ITC5 STC3
4.3.15	UTC3,UTC4 ITC6,ITC7,ITC8,ITC9,ITC10

4.3.16	UTC3,UTC4 ITC6,ITC7,ITC8,ITC9,ITC10
4.3.17	ITC11,ITC12,ITC13,ITC14,ITC15 STC1,STC2
4.3.18	UTC5,UTC6,UTC7,UTC8,UTC9
4.3.19	STC3
4.3.20	
4.3.21	,UTC10,UTC11,UTC12,UTC13,UTC14 STC3
4.3.22	ITC16 STC1
4.3.23	ITC17 STC1
4.3.24	UTC15
4.3.25	ITC18
4.3.26	ITC19
4.3.27	UTC16,UTC17,UTC18,UTC19,UTC20

ID	**ITC1**
Functional Req	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
TC Description	Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards.
Technique Used	**Decision Table Testing. (Black-box technique)**
Item to be tested	Class: FireFlower, Method:collideCheck()
Pre-condition	1- Launch the game 2- Player enters a level. 3- Chili powerup has spawned 4- Mario is not Large Mario 5- Mario is not Red Mario
Steps	1- Mario walks into the Chili powerup. 2- Mario consumes the Chili powerup.
Input	For xMarioD {-16,16} yMarioD in {-12,12}
Expected Result	Mario.large = True;

	mario.fire= True powerUpTime = 18;

ID	ITC2
Functional Req	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it.
TC Description	Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards.
Technique Used	**Decision Table Testing. (Black-box technique)**
Item to be tested	Class: FireFlower, Method:collideCheck()
Pre-condition	1- Launch the game 2- Player enters a level. 3- Chili powerup has spawned 4- Mario is not Large Mario 5- Mario is not Red Mario
Steps	1- Mario walks into the Chili powerup. 2- Mario consumes the Chili powerup.
Input	For xMarioD =16 yMarioD = 12

Expected Result	Mario.large = True;
	mario.fire= True;
	powerUpTime = 18;

ID	ITC3
Functional Req	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it.
TC Description	Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards.
Technique Used	**Decision Table Testing. (Black-box technique)**
Item to be tested	Class: FireFlower, Method:collideCheck()
Pre-condition	1- Launch the game 2- Player enters a level. 3- Chili powerup has spawned 4- Mario is not Large Mario 5- Mario is not Red Mario
Steps	1- Mario walks into the Chili powerup. 2- Mario consumes the Chili powerup.
Input	For xMarioD= 17 yMarioD = 13
Expected Result	Mario.large = False;

	mario.fire= False; powerUpTime = 0;

ID	ITC4
Functional Req	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size. If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it.
TC Description	Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards.
Technique Used	**Decision Table Testing. (Black-box technique)**
Item to be tested	Class: FireFlower, Method:collideCheck()
Pre-condition	1- Launch the game 2- Player enters a level. 3- Chili powerup has spawned 4- Mario is not Large Mario 5- Mario is not Red Mario
Steps	1- Mario walks into the Chili powerup. 2- Mario consumes the Chili powerup.
Input	For xMarioD= -16 yMarioD = -12

Expected Result	Mario.large = True;
	mario.fire= True;
	powerUpTime = 18;

ID	ITC5
Functional Req	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it.
TC Description	Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards.
Technique Used	**Decision Table Testing. (Black-box technique)**
Item to be tested	Class: FireFlower, Method:collideCheck()
Pre-condition	1- Launch the game 2- Player enters a level. 3- Chili powerup has spawned 4- Mario is not Large Mario 5- Mario is not Red Mario
Steps	1- Mario walks into the Chili powerup. 2- Mario consumes the Chili powerup.
Input	For xMarioD = -17 yMarioD = -13

Expected Result	Mario.large = False;
	mario.fire= False;
	powerUpTime = 0;

ID	ITC6
Functional Req	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
TC Description	Checking if Mario consumes Cake powerup up on impact, and if he becomes Red Mario afterwards
Technique Used	**BVA, Equivalence Partitioning**
Item to be tested	Class: Mashroom, Method:collideCheck()
Pre-condition	1- Launch the game 2- Player enters a level. 3- Cake powerup has spawned 4- Mario is not Large Mario 5- Mario is not Red Mario
Steps	1- Mario walks into Cake powerup 2- Mario consumes the Cake powerup
Input	For xMarioD {-16,16} yMarioD in {-12,12}
Expected Result	mario.Large = True; powerupTime = 18;

ID	ITC7
Functional Req	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
TC Description	Checking if Mario consumes Cake powerup up on impact, and if he becomes Red Mario afterwards
Technique Used	**BVA, Equivalence Partitioning**
Item to be tested	Class: Mashroom, Method:collideCheck()
Pre-condition	1- Launch the game 2- Player enters a level. 3- Cake powerup has spawned 4- Mario is not Large Mario 5- Mario is not Red Mario
Steps	1- Mario walks into Cake powerup 2- Mario consumes the Cake powerup
Input	For xMarioD =16 yMarioD = 12

Expected Result	mario.Large = True; powerupTime = 18;

ID	ITC8
Functional Req	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
TC Description	Checking if Mario consumes Cake powerup up on impact, and if he becomes Red Mario afterwards
Technique Used	**BVA, Equivalence Partitioning**
Item to be tested	Class: Mashroom, Method:collideCheck()
Pre-condition	1- Launch the game 2- Player enters a level. 3- Cake powerup has spawned 4- Mario is not Large Mario 5- Mario is not Red Mario
Steps	1- Mario walks into Cake powerup 2- Mario consumes the Cake powerup
Input	For xMarioD= 17 yMarioD = 13
Expected Result	mario.Large = False; powerupTime = 0;

ID	ITC9
Functional Req	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
TC Description	Checking if Mario consumes Cake powerup up on impact, and if he becomes Red Mario afterwards
Technique Used	**BVA, Equivalence Partitioning**
Item to be tested	Class: Mashroom, Method:collideCheck()
Pre-condition	1- Launch the game 2- Player enters a level. 3- Cake powerup has spawned 4- Mario is not Large Mario 5- Mario is not Red Mario
Steps	1- Mario walks into Cake powerup 2- Mario consumes the Cake powerup
Input	For xMarioD= -16 yMarioD = -12

Expected Result	mario.Large = True; powerupTime = 18;

ID	ITC10
Functional Req	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
TC Description	Checking if Mario consumes Cake powerup up on impact, and if he becomes Red Mario afterwards
Technique Used	**BVA, Equivalence Partitioning**
Item to be tested	Class: Mashroom, Method:collideCheck()
Pre-condition	1- Launch the game 2- Player enters a level. 3- Cake powerup has spawned 4- Mario is not Large Mario 5- Mario is not Red Mario
Steps	1- Mario walks into Cake powerup 2- Mario consumes the Cake powerup
Input	For xMarioD = -17 yMarioD = -13
Expected Result	mario.Large = False; powerupTime = 0;

ID	ITC11
Functional Req	4.3.17 If Mario collides with an enemy, then he should get hurt.
TC Description	Checking if Mario gets hurt when colliding with an enemy.
Technique Used	**Equivalence Partitioning (Black-Box).**
Item to be tested	Class: Enemy, Method:collideCheck()
Pre-condition	1-Launch the game. 2- Player enters a level. 3- Enemy spawns. 4- Enemy has spawned 5- Mario is not invulnerable 6- Mario is not Large Mario 7- Mario is not Red Mario 8- Mario have 1 life only
Steps	1- Mario walks into an enemy. 2- Mario collides with an enemy.
Input	For xMarioD{-12,12} yMarioD in {-12,12} enemy type = 2 (Goomba)

Expected Result	deathTime=1;

ID	ITC12
Functional Req	4.3.17 If Mario collides with an enemy, then he should get hurt.
TC Description	Checking if Mario gets hurt when colliding with an enemy.
Technique Used	**Equivalence Partitioning (Black-Box).**
Item to be tested	Class: Enemy, Method:collideCheck()
Pre-condition	1-Launch the game. 2- Player enters a level. 3- Enemy spawns. 4- Enemy has spawned 5- Mario is not invulnerable 6- Mario is not Large Mario 7- Mario is not Red Mario 8- Mario have 1 life only
Steps	1- Mario walks into an enemy. 2- Mario collides with an enemy.
Input	For xMarioD{12} yMarioD in {12} enemy type = 2 (Goomba)
Expected Result	deathTime=1;

ID	ITC13
Functional Req	4.3.17 If Mario collides with an enemy, then he should get hurt.
TC Description	Checking if Mario gets hurt when colliding with an enemy.
Technique Used	**Equivalence Partitioning (Black-Box).**
Item to be tested	Class: Enemy, Method:collideCheck()
Pre-condition	1-Launch the game. 2- Player enters a level. 3- Enemy spawns. 4- Enemy has spawned 5- Mario is not invulnerable 6- Mario is not Large Mario 7- Mario is not Red Mario 8- Mario have 1 life only
Steps	1- Mario walks into an enemy. 2- Mario collides with an enemy.
Input	For xMarioD{13} yMarioD in {13}

	enemy type = 2 (Goomba)
Expected Result	deathTime=0;

ID	**ITC14**
Functional Req	4.3.17 If Mario collides with an enemy, then he should get hurt.
TC Description	Checking if Mario gets hurt when colliding with an enemy.
Technique Used	**Equivalence Partitioning (Black-Box).**
Item to be tested	Class: Enemy, Method:collideCheck()
Pre-condition	1-Launch the game. 2- Player enters a level. 3- Enemy spawns. 4- Enemy has spawned 5- Mario is not invulnerable 6- Mario is not Large Mario 7- Mario is not Red Mario 8- Mario have 1 life only
Steps	1- Mario walks into an enemy. 2- Mario collides with an enemy.
Input	For xMarioD{-12} yMarioD in {-12} enemy type = 2 (Goomba)
Expected Result	deathTime=1;

ID	**ITC15**
Functional Req	4.3.17 If Mario collides with an enemy, then he should get hurt.
TC Description	Checking if Mario gets hurt when colliding with an enemy.
Technique Used	**Equivalence Partitioning (Black-Box).**
Item to be tested	Class: Enemy, Method:collideCheck()
Pre-condition	1-Launch the game. 2- Player enters a level. 3- Enemy spawns. 4- Enemy has spawned 5- Mario is not invulnerable 6- Mario is not Large Mario 7- Mario is not Red Mario 8- Mario have 1 life only
Steps	1- Mario walks into an enemy. 2- Mario collides with an enemy.
Input	For xMarioD{-13}

	yMarioD in {-13} enemy type = 2 (Goomba)
Expected Result	deathTime=0;

ID	ITC16
Functional Req	4.3.22 If Mario jumps on Red Kubba then it should become a Shell.
TC Description	Mario collides with Red Kubba which leads to a new shell appearing.
Technique Used	**Black box → Cause and effect**
Item to be tested	Class: Enemy, Method: collideCheck()
Pre-condition	1. Launch the game 2. Player starts a level 3. Enemy spawn. 4. Mario jump. 5. Mario hits the Red Kubba once.
Steps	1. Mario jumps. 2. Mario hits the Red Kubba once. 3. Red Kubba transfers into shell.
Input	Mario Coordinates & Enemy Coordinates
Expected Result	spriteContext.addSprite(new Shell(world, x, y, 0));

ID	ITC17
Functional Req	4.3.23 If Mario jumps on Green Kubba then it should become a Shell.
TC Description	Mario collides with Green Kubba which leads to a new shell appearing.
Technique Used	**Black box → Cause and effect**
Item to be tested	Class: Enemy, Method: collideCheck()
Pre-condition	1. Launch the game 2. Player starts a level 3. Enemy spawn. 4. Mario jump. 5. Mario hits the Green Kubba once.
Steps	1. Mario jumps. 2. Mario hits the Green Kubba once. 3. Green Kubba transfers into shell.

Input	Mario Coordinates
Expected Result	spriteContext.addSprite(new Shell(world, x, y, 1));

ID	**ITC18**
Functional Req	4.3.25 If Mario collides with an enemy during vulnerability, he should not get damaged.
TC Description	When Mario gets hit alive for a specific amount of time
Technique Used	**Black box → Cause and effect**
Item to be tested	Class: Mario, getHurt()
Pre-condition	1. Launch the game. 2. Player starts a level. 3. Enemy spawn. 4. Mario eats red pepper or cake. 5. Mario becomes a large/fire state.
Steps	1. Mario gets hurt by an enemy. 2. Mario absorbs the damage. 3. Mario becomes unhittable for a moment when he got hit.
Input	get hurt by an enemy
Expected Result	paused2 = true;

ID	**ITC19**
Functional Req	4.3.26 The system shall allow the user start spawn in the LevelScene after clicking S in MapScene.
TC Description	**Verify when the user start playing in the level when when click in the map scene.**
Technique Used	**Branch coverage (White-Box).**
Item to be tested	Class: MapScene, tick()
Pre-condition	1. Launch the game. 2. The user starts selecting the level he wants to play. 3. The user clicks S for the selected level in map scene. 4. The level starts to be generated.
Steps	1. Run the game. 2. Select the level and click S in map scene.
Input	ValueCheck= False;
Expected Result	ValueCheck= Ture;

Post-condition	the level is generated and ready to be played.

ID	ITC20
TC Description	game starts after pressing S key by the user
Functional Req	4.1.1 The system shall allow the user to start the game by pressing S.
Technique Used	**Black box → Cause and effect**
Item to be tested	TitleScene class tick(): method
Pre-condition	1-Launch the game.
Steps	1- launch the game 2- start screen shows
Input	Press S key
Expected Result	screen of level map will display
Post Condition	game should generate a random map for the player

18.3 System test cases:

The system test cases are used to test a feature of the game. Each system test case consists of several related unit and integration test cases. Each feature covers several functional requirements.

ID	STC1
Feature to be tested	Mario should get hurt if he collides with an enemy and if the enemy is a red or green Kubba the enemy becomes a shell. If Mario touches the shell, it should start moving and if it collides with Mario again, he will die ---------- Mario should change the Kubba (red/green) to a shell if collide with it from top and make it move if touch it from the sides
Functional Requirements covered by feature	• 4.3.2 • 4.3.4 • 4.3.12

	• 4.3.17 • 4.3.22 • 4.3.23
Related unit and integration test cases.	• ITC 11, 12, 13, 14, 15, 16, 17
Post-condition	If the shell collides with mario dies

ID	**STC2**
Feature to be tested	When Mario dies or the timer of the game finishes, "Game over screen" should be displayed
Functional Requirements covered by feature	• 4.3.2 • 4.3.6 • 4.3.12 • 4.3.17
Related unit and integration test cases.	• ITC 11, **12**, 13, 14, 15
Post-condition	The GUI should display the "Game over screen"

ID	**STC3**
Feature to be tested	The Chili powerup should make the Mario red and large and the Mario should be able to throw fireballs that kill enemies.
Functional Requirements covered by feature	• 4.3.13 • 4.3.14 • 4.3.19 • 4.3.21
Related unit and integration test cases.	• UTC 10, 11, 12, 13, 14 • ITC 1, 2, 3, 4, 5

Post-condition	Enemy dies.

ID	**STC4**
Feature to be tested	The player should be able to use the map to start playing levels.
Functional Requirements covered by feature	● 4.2.1 ● 4.2.2
Related unit and integration test cases.	● UTC 22, 23, 24, 25, 26 ● ITC 19
Post-condition	The player enters the level.

ID	**STC5**
Feature to be tested	This feature is related to UI functions, such as closing the game with ESC and starting the game at the beginning by pressing S.
Functional Requirements covered by feature	● 4.1.1 ● 4.1.4
Related unit and integration test cases.	● UTC 22, 23, 24, 25, 26 ● ITC 20
Post-condition	● Game closed if ESC pressed. ● Game Started if S pressed.

19. Techniques Used:

19.1 Black-Box Techniques:

19.1.1 Cause and effect.

We've used cause and effect technique that will consider our output that is caused by our input and make sure if our output is correct and satisfy our test cases.

19.1.2 Equivalence Partitioning (EP):

We will divide the input domain into equivalence classes. We used technique only when we had a range of inputs, to create equivalence classes. It can't be used with methods that take binary/Boolean inputs (true/false).

19.1.3 Boundary Value Analysis:

We will look at the edges of input ranges. We used this technique with (EP) to check boundaries of Equivalence classes.

19.1.4 Decision Table Testing:

We used this technique when we had a binary/Boolean input, and we had a small number of inputs. We used this technique to consider all possible situations for a specific method input and corresponding output.

19.2 White-Box Techniques:

19.2.1 Branch Coverage:

We've used branch coverage to make sure that every edge in our control graph that we've made covers the statement (if, while, else if, else, switch) and executed at least once.

19.2.2 Statement Coverage:

We've used statement coverage to make sure that our test cases cover every statement and executed at least once.

Cause and effect	Equivalence Partitioning (EP)	Boundary Value Analysis	Branch Coverage	Statement Coverage	Decision Table Testing
UTC's: 27,28,29,30. **ITC's:** 16,17,18,20.	**UTC's:** 10,11,12,13,14,22,23,24,25,26. **ITC's:** 6,7,8,9,10,11,12,13,14,15.	**UTC's:** 10,11,12,13,14. **ITC's:** 6,7,8,9,10.	**ITC:** 19.	**UTC's:** 15,21.	**UTC's:** 1,2,3,4,5,6,7,8,9. **ITC's:** 1,2,3,4,5.

Phase 3:

20. Unit Testing

Note: when a test cases number is noted in the following format → UTC_X_C, we mean here that for unit test case number X is Splitted in different parts; such that we avoid making complex test cases.

Unit Test Cases	Passed Test cases	Failed Test cases
55 test case	Number of TC's: 55	Number of TC's: 0

Test Case Number	UTC_1_1
Test Case	verify that when Mario eats a Chili powerup, it increases Mario Size.

Code	`@Test`
	`void UTC_1_1() {`
	`LevelScene level=Mockito.mock(LevelScene.class);`
	`Mario mar=new Mario(level);`
	`boolean large=false;`
	`boolean fire=true;`
	`mar.setLarge(large, fire);`
	`Mario is supposed to be large after setLarge is called`
	`boolean expectedLarge=true;`
	`boolean resultLarge=mar.large;`
	`assertEquals(expectedLarge,resultLarge);`
	`}`
	`@Test`

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it.
Expected Result	Mario becomes large
Actual Result	Mario became large
Output	**PASS**

Test Case Number	UTC_1_2
Test Case	verify that when Mario eats a Chili powerup, it gives him fireball ability

Code	`@Test`
	`void UTC_1_2() {`
	`LevelScene level=Mockito.mock(LevelScene.class);`
	`Mario mar=new Mario(level);`
	`boolean large=false;`
	`boolean fire=true;`
	`mar.setLarge(large, fire);`
	`Mario is supposed to have fire ability after setLarge is called`
	`boolean expectedFire=true;`
	`boolean resultFire=mar.fire;`
	`assertEquals(expectedFire,resultFire);`
	`}`

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it.
Expected Result	Mario has fireball ability
Actual Result	Mario had the fireball ability
Output	**PASS**

Test Case Number	**UTC_2_1**
Test Case	verify that when Mario a Cake then Chili powerup, he will become large

Code	`@Test`
	`void UTC_2_1() {`
	`LevelScene level=Mockito.mock(LevelScene.class);`
	`Mario mar=new Mario(level);`
	`boolean large=true;`
	`boolean fire=true;`
	`mar.setLarge(large, fire);`
	`Mario is supposed to be large after setLarge is called`
	`boolean expectedLarge=true;`
	`boolean resultLarge=mar.large;`
	`assertEquals(expectedLarge,resultLarge);`
	`}`

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it.
Expected Result	Mario will become Large
Actual Result	Mario becomes large
Output	**PASS**

Test Case Number	UTC_2_2
Test Case	verify that when Mario does eat Cake and Chili powerups, he will have fireball ability
Code	

```
@Test
void UTC_2_2() {
    LevelScene level=Mockito.mock(LevelScene.class);
    Mario mar=new Mario(level);
    boolean large=true;
    boolean fire=true;
    mar.setLarge(large, fire);
    Mario is supposed to have fire ability after setLarge is called
    boolean expectedFire=true;
    boolean resultFire=mar.fire;
    assertEquals(expectedFire,resultFire);
}
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it. 4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	Mario will have fireball ability
Actual Result	Mario has fireball ability
Output	PASS

Test Case Number	UTC_3_1
Test Case	verify that when Mario eats a cake powerup, his size will increase and become large
Code	

```
@Test
void UTC_3_1() {
    LevelScene level=Mockito.mock(LevelScene.class);
    Mario mar=new Mario(level);
    boolean large=true;
    boolean fire=false;
    mar.setLarge(large, fire);
    Mario is supposed to be large after setLarge is called
    boolean expectedLarge=true;
    boolean resultLarge=mar.large;
    assertEquals(expectedLarge,resultLarge);
}
@Test
```

REQ	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	Mario will become Large

Actual Result	Mario did become large
Output	**PASS**

Test Case Number	**UTC_3_2**
Test Case	verify that when Mario does eat a Cake powerup, he will not have fireball ability
Code	

```
@Test
void UTC_3_2() {
    LevelScene level=Mockito.mock(LevelScene.class);
    Mario mar=new Mario(level);
    boolean large=true;
    boolean fire=false;
    mar.setLarge(large, fire);
    Mario is not supposed to have fire ability after setLarge is called
    boolean expectedFire=false;
    boolean resultFire=mar.fire;
    assertEquals(expectedFire,resultFire);
}
```

REQ	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	Mario will not have fireball ability
Actual Result	Mario didn't have fireball ability
Output	**PASS**

Test Case Number	**UTC_4_1**
Test Case	verify that when Mario doesn't eat a cake powerup, he will not become large.
Code	

```
@Test
void UTC_4_1() {
    LevelScene level=Mockito.mock(LevelScene.class);
    Mario mar=new Mario(level);
    boolean large=false;
    boolean fire=false;
    mar.setLarge(large, fire);
    Mario is supposed to large after setLarge is called
    boolean expectedLarge=false;
    boolean resultLarge=mar.large;
    assertEquals(expectedLarge,resultLarge);
}
```

REQ	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	Mario will not become Large

Actual Result	Mario did not become large
Output	**PASS**

Test Case Number	**UTC_4_2**
Test Case	verify that when mario doesn't eat a cake powerup, he will not have fireball ability
Code	

```
@Test
void UTC_4_2() {
    LevelScene level=Mockito.mock(LevelScene.class);
    Mario mar=new Mario(level);
    boolean large=false;
    boolean fire=false;
    mar.setLarge(large, fire);
    Mario is supposed to have fire ability after setLarge is called
    boolean expectedFire=false;
    boolean resultFire=mar.fire;
    assertEquals(expectedFire,resultFire);
}
@Test
```

REQ	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	Mario will not have fireball ability
Actual Result	Mario did not have fireball ability
Output	**PASS**

Test Case Number	**UTC_5_1**
Test Case	verifying that Large Mario decreases in size after getting damaged by an enemy.

Code	

```
@Test
public  void UTC_5_1(){
    LevelScene world = Mockito.mock(LevelScene.class);
    Mario test = new Mario (world);
    test.large= true;
    test.deathTime=0;
    test.fire=false;
    test.getHurt2_Abdullah();
    boolean expectedValueLarge  =false;
    boolean actualValueLarge= test.large;
    assertEquals(expectedValueLarge,actualValueLarge);
}
```

REQ	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
Expected Result	Mario.large =false;
Actual Result	Mario.large=False;
Output	**PASS**

Test Case Number	**UTC_5_2**
Test Case	Verifying that changing Mario from large to small doesn't make Mario into a Fire Mario
Code	

```
@Test
public void UTC_5_2() throws LineUnavailableException{

    LevelScene world = Mockito.mock(LevelScene.class);
    Mario test = new Mario (world);
    test.setInvulnerableTime_Abdullah(0);
    test.paused2_Abdullah = false;
    test.large= true;
    test.deathTime=0;
    test.fire=false;
    test.getHurt2_Abdullah();
    boolean expectedValueFire=false;
    boolean actualVlueFire = test.fire;
    assertEquals(expectedValueFire,actualVlueFire);
}
```

REQ	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
Expected Result	Mario.fire =False;

Actual Result	Mario.fire=False;
Output	**PASS**

Test Case Number	**UTC_6_1**
Test Case	Making sure that if normal Mario gets hurt, he doesn't turn into Large Mario
Code	

```
}
@Test
public  void UTC_6_1(){
    LevelScene world = Mockito.mock(LevelScene.class);
    Mario test = new Mario (world);
    test.large= false;
    test.setInvulnerableTime_Abdullah(0);
    test.paused2_Abdullah = false;
    test.deathTime=0;
    test.fire=false;
    test.getHurt2_Abdullah();
    boolean expectedValueLarge  =false;
    boolean actualValueLarge= test.large;
    assertEquals(expectedValueLarge,actualValueLarge);
}
```

REQ	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
Expected Result	Mario.large = False;
Actual Result	Mario.large = False;
Output	**PASS**

Test Case Number	**UTC_6_2**
Test Case	Making sure that if normal Mario gets hurt, he doesn't turn into Fire Mario
Code	

```
}
@Test
public  void UTC_6_2(){
    LevelScene world = Mockito.mock(LevelScene.class);
    Mario test = new Mario (world);
    test.large= false;
    test.setInvulnerableTime_Abdullah(0);
    test.paused2_Abdullah = false;
    test.deathTime=0;
    test.fire=false;
    test.getHurt2_Abdullah();
    boolean expectedValueFire   =false;
    boolean actualValueFire= test.fire;
    assertEquals(expectedValueFire,actualValueFire);
}
```

REQ	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
Expected Result	Mario.fire = False;
Actual Result	Mario.fire = False;
Output	**PASS**

Test Case Number	**UTC_6_3**
Test Case	Making sure that if normal Mario gets hurt, he dies
Code	

```
}
@Test
public  void UTC_6_3(){
    LevelScene world = Mockito.mock(LevelScene.class);
    Mario test = new Mario (world);
    test.large= false;
    test.setInvulnerableTime_Abdullah(0);
    test.paused2_Abdullah = false;
    test.deathTime=0;
    test.fire=false;
    test.getHurt2_Abdullah();
    int expectedValueDeathTime  =1;
    int actualValueDeathTime= test.deathTime;
    assertEquals(expectedValueDeathTime,actualValueDeathTime);
}
```

REQ	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
Expected Result	deathTime=1; (dead)
Actual Result	deathTime=1; (dead)
Output	**PASS**

Test Case Number	**UTC_7_1**
Test Case	Making sure if the game is pause, Large Mario doesn't become normal Mario

Code	

```
}
@Test
public  void UTC_7_1(){
    LevelScene world = Mockito.mock(LevelScene.class);
    Mario test = new Mario (world);
    test.large= true;
    test.setInvulnerableTime_Abdullah(0);
    test.paused2_Abdullah = true;
    test.deathTime=0;
    test.fire=false;
    test.getHurt2_Abdullah();
    boolean expectedValueLarge = true;
    boolean actualValueLarge= test.large;
    assertEquals(expectedValueLarge,actualValueLarge);
}
```

REQ	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
Expected Result	Mario.large = True;
Actual Result	Mario.large = True;;
Output	**PASS**

Test Case Number	**UTC_7_2**
Test Case	Making sure if the game is pause, Fire Mario doesn't become normal Mario
Code	

```
}
@Test
public  void UTC_7_2(){
    LevelScene world = Mockito.mock(LevelScene.class);
    Mario test = new Mario (world);
    test.large= true;
    test.setInvulnerableTime_Abdullah(0);
    test.paused2_Abdullah = true;
    test.deathTime=0;
    test.fire=false;
    test.getHurt2_Abdullah();
    boolean expectedValueFire   =false;
    boolean actualValueFire= test.fire;
    assertEquals(expectedValueFire,actualValueFire);
}
```

REQ	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
Expected Result	Mario.fire = False;
Actual Result	Mario.fire = False;
Output	**PASS**

Test Case Number	UTC_7_3
Test Case	Making sure if the game is pause, Mario doesn't die
Code	

```
@Test
public  void UTC_7_3(){
    LevelScene world = Mockito.mock(LevelScene.class);
    Mario test = new Mario (world);
    test.large= true;
    test.setInvulnerableTime_Abdullah(0);
    test.paused2_Abdullah = true;
    test.deathTime=0;
    test.fire=false;
    test.getHurt2_Abdullah();
    int expectedValueDeathTime  =0;
    int actualValueDeathTime= test.deathTime;
    assertEquals(expectedValueDeathTime,actualValueDeathTime);
}
```

REQ	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
Expected Result	deathTime=0;
Actual Result	deathTime=0;
Output	PASS

Test Case Number	UTC_8_1
Test Case	Verifying that if Mario is dead, the value of death time will be set to 1
Code	

```
@Test
public  void UTC_8_1(){
    LevelScene world = Mockito.mock(LevelScene.class);
    Mario test = new Mario (world);
    test.large= false;
    test.setInvulnerableTime_Abdullah(0);
    test.paused2_Abdullah = false;
    test.deathTime=1;
    test.fire=false;
    test.getHurt2_Abdullah();
    int expectedValueDeathTime = 1;
    int actualValueDeathTime= test.deathTime;
    assertEquals(expectedValueDeathTime,actualValueDeathTime);
}
```

REQ	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
Expected Result	deathTime=1;
Actual Result	deathTime=1;
Output	PASS

Test Case Number	UTC_9_1
Test Case	Verifying that if Mario is large and fire, he will not lose his buffs if the game is paused.
Code	

```
@Test
public  void UTC_9_1(){
    LevelScene world = Mockito.mock(LevelScene.class);
    Mario test = new Mario (world);
    test.large= true;
    test.setInvulnerableTime_Abdullah(0);
    test.paused2_Abdullah = true;
    test.deathTime=0;
    test.fire=true;
    test.getHurt2_Abdullah();
    boolean expectedValueLarge  = true;
    boolean actualValueLarge= test.large;
    assertEquals(expectedValueLarge,actualValueLarge);
}
```

REQ	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
Expected Result	Mario.large = True;
Actual Result	Mario.large = True;
Output	PASS

Test Case Number	UTC_9_2
Test Case	Verifying that if Mario is large and fire, he will not lose his buffs if the game is paused.

Code	

```
@Test
public  void UTC_9_2(){
    LevelScene world = Mockito.mock(LevelScene.class);
    Mario test = new Mario (world);
    test.large= true;
    test.setInvulnerableTime_Abdullah(0);
    test.paused2_Abdullah =true;
    test.deathTime=0;
    test.fire=true;
    test.getHurt2_Abdullah();
    boolean expectedValueFire   = true;
    boolean actualValueFire= test.fire;
    assertEquals(expectedValueFire,actualValueFire);
}
```

REQ	4.3.18 The system shall decrease the size of the large Mario after hitting an enemy.
Expected Result	Mario.fire = True;
Actual Result	Mario.fire = True;
Output	**PASS**

Test Case Number	**UTC_10_1**
Test Case	Testing whether FireballCallidCheck will return true when the fireball hits an enemy from the middle of its hitbox.
Code	

```
@Test
public void UTC_10_1(){

    LevelScene test = Mockito.mock( LevelScene.class);
    Enemy enemy = new Enemy(test, 1, 1, 1, 1, false);
    Fireball fireBall = new Fireball(test, 4,4,1);
    // now lets get xD, yD in {-4,4}... the intilaization above
    // will give us, xD=3, yD=3;
    enemy.deadTime=0;
    enemy.fireballCollideCheck2_Abdullah(fireBall);

    assertTrue(enemy.isCollidFireBall_Abdullah);

}
```

REQ	4.3.21 The system shall make enemies who get hit by fireballs die.
Expected Result	isCollidFireBall=true;
Actual Result	isCollidFireBall=false;
Output	**PASS**

Test Case Number	UTC_10_2
Test Case	Testing whether FireballCallidCheck will turn the enemy hit by the fireball to a dead enemy (deadTime=100).
Code	

```
@Test
public void UTC_10_2(){

    LevelScene test = Mockito.mock( LevelScene.class);
    Enemy enemy = new Enemy(test, 1, 1, 1, 1, false);
    Fireball fireBall = new Fireball(test, 4,4,1);
    // now lets get xD, yD in {-4,4}... the intilaization above
    // will give us, xD=3, yD=3;
    enemy.deadTime=0;
    enemy.fireballCollideCheck2_Abdullah(fireBall);

    assertEquals(enemy.deadTime,100);

}
```

REQ	4.3.21 The system shall make enemies who get hit by fireballs die
Expected Result	deadTime=100;
Actual Result	deadTime=0;
Output	PASS

Test Case Number	UTC_11_1
Test Case	Testing whether FireballCallidCheck will return true when a Fireball hits the edge of an enemy's hitbox..
Code	

```
@Test
public void UTC_11_1(){

    LevelScene test = Mockito.mock( LevelScene.class);
    Enemy enemy = new Enemy(test, 0, 0, 1, 1, false);
    Fireball fireBall = new Fireball(test, 4,8,1);
    // the intilaization above
    // will give us, xD=4, yD=8;
    enemy.deadTime=0;
    enemy.fireballCollideCheck2_Abdullah(fireBall);

    assertTrue(enemy.isCollidFireBall_Abdullah);

}
```

REQ	4.3.21 The system shall make enemies who get hit by fireballs die
Expected Result	isCollidFireBall=true;

Actual Result	isCollidFireBall=false;
Output	**FAIL**

Test Case Number	**UTC_11_2**
Test Case	Testing whether FireballCallidCheck will turn an enemy hit by a Fireball to dead enemy (deathTime =100). When the fireball hits the edge of the enemy's hitbox.
Code	

```
@Test
public void UTC_11_2(){

    LevelScene test = Mockito.mock( LevelScene.class);
    Enemy enemy = new Enemy(test, 0, 0, 1, 1, false);
    Fireball fireBall = new Fireball(test, 4,8,1);
    // the intilaization above
    // will give us, xD=4, yD=8;
    enemy.deadTime=0;
    enemy.fireballCollideCheck2_Abdullah(fireBall);

    assertEquals(enemy.deadTime,100);

}
```

REQ	4.3.21 The system shall make enemies who get hit by fireballs die
Expected Result	deadTime=100;
Actual Result	deadTime=0;
Output	**FAIL**

Test Case Number	**UTC_12_1**
Test Case	Testing whether FireballCollidCheck returns true when the Fireball touches the a "pixel" after its hitbox (x=33,y=33).

Code	

```
@Test
public void UTC_12_1(){

    LevelScene test = Mockito.mock( LevelScene.class);
    Enemy enemy = new Enemy(test, 0, 0, 1, 1, false);
    Fireball fireBall = new Fireball(test, 5,9,1);
    // the intilaization above
    // will give us, xD=5, yD=9;
    enemy.deadTime=0;
    enemy.fireballCollideCheck2_Abdullah(fireBall);

    assertFalse(enemy.isCollidFireBall_Abdullah);

}
```

REQ	4.3.21 The system shall make enemies who get hit by fireballs die
Expected Result	isCollidFireball=false;
Actual Result	isCollidFireball=false;
Output	**PASS**

Test Case Number	**UTC_12_2**
Test Case	Testing whether an enemy dies when the Fireball touches the “pixel” after its hitbox (x=33,y=33).
Code	

```
@Test
public void UTC_12_2(){

    LevelScene test = Mockito.mock( LevelScene.class);
    Enemy enemy = new Enemy(test, 0, 0, 1, 1, false);
    Fireball fireBall = new Fireball(test, 5,9,1);
    // the intilaization above
    // will give us, xD=5, yD=9;
    enemy.deadTime=0;
    enemy.fireballCollideCheck2_Abdullah(fireBall);

    assertEquals(enemy.deadTime,0);

}
```

REQ	4.3.21 The system shall make enemies who get hit by fireballs die
Expected Result	deathTime=0; (not dead)
Actual Result	deathTime=0; (not dead)
Output	**PASS**

Test Case Number	UTC_13_1
Test Case	Testing whether FireBallCollidCheck returns true when the Fireball hit the back edge of an enemy's hitbox.
Code	

```
@Test
public void UTC_13_1(){

    LevelScene test = Mockito.mock( LevelScene.class);
    Enemy enemy = new Enemy(test, 4, 24, 1, 1, false);
    Fireball fireBall = new Fireball(test, 0,0,1);
    // the intilaization above
    // will give us, xD=-4, yD=-24;
    enemy.deadTime=0;
    enemy.fireballCollideCheck2_Abdullah(fireBall);

    assertTrue(enemy.isCollidFireBall_Abdullah);

}
```

REQ	4.3.21 The system shall make enemies who get hit by fireballs die
Expected Result	isCollidFireBall = true;
Actual Result	isCollidFireBall = false;
Output	FAIL

Test Case Number	UTC_13_2
Test Case	Testing whether an enemy die when the Fireball hit the back edge of an enemy's hitbox.
Code	

```
@Test
public void UTC_13_2(){

    LevelScene test = Mockito.mock( LevelScene.class);
    Enemy enemy = new Enemy(test, 4, 24, 1, 1, false);
    Fireball fireBall = new Fireball(test, 0,0,1);
    // the intilaization above
    // will give us, xD=-4, yD=-24;
    enemy.deadTime=0;
    enemy.fireballCollideCheck2_Abdullah(fireBall);

    assertEquals(enemy.deadTime,100);

}
```

REQ	4.3.21 The system shall make enemies who get hit by fireballs die
Expected Result	deathTime=100; (dead)
Actual Result	deathTime=-0; (not dead)

Output	**FAIL**

Test Case Number	**UTC_14_1**
Test Case	Testing whether FireballCollidCheck return false when the fireball touches one “pixel” after the enemy.
Code	

```
@Test
public void UTC_14_1(){

    LevelScene test = Mockito.mock( LevelScene.class);
    Enemy enemy = new Enemy(test, 5, 25, 1, 1, false);
    Fireball fireBall = new Fireball(test, 0,0,1);
    // the intilaization above
    // will give us, xD=-5, yD=-25;
    enemy.deadTime=0;
    enemy.fireballCollideCheck2_Abdullah(fireBall);

    assertFalse(enemy.isCollidFireBall_Abdullah);

}
```

REQ	4.3.21 The system shall make enemies who get hit by fireballs die
Expected Result	isCollidFireBall = false;
Actual Result	isCollidFireBall = false;
Output	**PASS**

Test Case Number	**UTC_14_2**
Test Case	Testing whether FireballCollidCheck will make enemy die (deathTime =100) when the fireball touches one “pixel” after the enemy.
Code	

```
@Test
public void UTC_14_2(){

    LevelScene test = Mockito.mock( LevelScene.class);
    Enemy enemy = new Enemy(test, 5, 25, 1, 1, false);
    Fireball fireBall = new Fireball(test, 0,0,1);
    // the intilaization above
    // will give us, xD=-5, yD=-25;
    enemy.deadTime=0;
    enemy.fireballCollideCheck2_Abdullah(fireBall);

    assertEquals(enemy.deadTime,0);

}
```

REQ	4.3.21 The system shall make enemies who get hit by fireballs die
Expected Result	deathTime=0; (not dead)
Actual Result	deathTime=0; (not dead)
Output	**PASS**

Test Case Number	**UTC_15**
Test Case	If large/fire Mario gets damaged, he should become invulnerable.
Code	

```
    @Test
    //Functional req: If large/fire Mario gets damaged, he should become invulnerable.
    // delete here the calling method after test this method (Scaffolding)
    //Statement Coverage
    void UTC_15() {
        LevelScene level=Mockito.mock(LevelScene.class,Mockito.CALLS_REAL_METHODS);
        Fireball fire=new Fireball(level,32,32,1);

           // assuming that is going out of branch!!!!!!!!!!!

        assertTrue(Fireball.isBlocking_FouadAlkadri(32, 32, 16, 16));
        // above the inputs _x,_y,xa,ya
        //levelo.isBlocking(32, 32, 32, 32);
    //  if(fire.x==fire.x/16 && fire.y==fire.y/16 ) {
    //      assertFalse(fire.callisblocking(33, 33, 16, 16));
    //}

}
```

REQ	4.3.24 If large/fire Mario gets damaged, he should become invulnerable.
Expected Result	return blocking
Actual Result	return blocking
Output	**PASS**

Test Case Number	**UTC_16**
Test Case	The system shall allow us to add 1 life after collecting 100 coins

Code	

```
//Technique Used : Equivalence Partition


// functional req : The system shall allow us to add 1 life after collecting 100 coins.

//Range: (0 --> 20)
@Test
void UTC_16_1(){
    int expected=1;
    int result= Mario.getCoin_FouadAlkadri(0);
    assertSame(expected,result);
    }
@Test
void UTC_16_2(){
    int expected=11;
    int result= Mario.getCoin_FouadAlkadri(10);
    assertSame(expected,result);
    }
@Test
void UTC_16_3(){
    int expected=20;
    int result= Mario.getCoin_FouadAlkadri(19);
    assertSame(expected,result);
    }
```

REQ	4.3.27 The system shall allow the user to add 1 life after collecting 100 coins.

Expected Result	coins=1;
	coins=11;
	coins=20;

Actual Result	coins=1;
	coins=11;
	coins=20;

Output	**PASS**

Test Case Number	**UTC_17**
Test Case	The system shall allow us to add 1 life after collecting 100 coins

Code	

```
@Test
//Range: (20 --> 40)
void UTC_17_1() {
    int expected=21;
    int result= Mario.getCoin_FouadAlkadri(20);
    assertSame(expected,result);
}
@Test
void UTC_17_2(){
    int expected=30;
    int result= Mario.getCoin_FouadAlkadri(29);
    assertSame(expected,result);
    }
@Test
void UTC_17_3(){
    int expected=40;
    int result= Mario.getCoin_FouadAlkadri(39);
    assertSame(expected,result);
    }
```

REQ	4.3.27 The system shall allow the user to add 1 life after collecting 100 coins.

Expected Result	coins=21;
	coins=30;
	coins=40;

Actual Result	coins=21;
	coins=30;
	coins=40;

Output	**PASS**

Test Case Number	UTC_18
Test Case	The system shall allow us to add 1 life after collecting 100 coins
Code	

```
@Test
//Range: (40 --> 60 )
void UTC_18_1(){
    int expected=41;
    int result= Mario.getCoin_FouadAlkadri(40);
    assertSame(expected,result);
    }
@Test
void UTC_18_2(){
    int expected=51;
    int result= Mario.getCoin_FouadAlkadri(50);
    assertSame(expected,result);
    }
@Test
void UTC_18_3(){
    int expected=60;
    int result= Mario.getCoin_FouadAlkadri(59);
    assertSame(expected,result);
    }
```

REQ	4.3.27 The system shall allow the user to add 1 life after collecting 100 coins.

Expected Result	coins=41;
	coins=51;
	coins=60;

Actual Result	coins=41;
	coins=51;
	coins=60;

Output	PASS

Test Case Number	UTC_19
Test Case	The system shall allow us to add 1 life after collecting 100 coins

Code	<pre>@Test
	//Range: (60 --> 80)
	void UTC_19_1(){
	int expected=61;
	int result= Mario.getCoin_FouadAlkadri(60);
	assertSame(expected,result);
	}
	@Test
	void UTC_19_2(){
	int expected=71;
	int result= Mario.getCoin_FouadAlkadri(70);
	assertSame(expected,result);
	}
	@Test
	void UTC_19_3(){
	int expected=80;
	int result= Mario.getCoin_FouadAlkadri(79);
	assertSame(expected,result);
	}</pre>

REQ	4.3.27 The system shall allow the user to add 1 life after collecting 100 coins.

Expected Result	coins=61;
	coins=71;
	coins=80;

Actual Result	coins=61;
	coins=71;
	coins=80;

Output	**PASS**

Test Case Number	UTC_20
Test Case	The system shall allow us to add 1 life after collecting 100 coins
Code	

```
//Range: (80--> 100)
@Test
void UTC_20_1(){
    int expected=81;
    int result= Mario.getCoin_FouadAlkadri(80);
    assertSame(expected,result);
    }
@Test
void UTC_20_2(){
    int expected=91;
    int result= Mario.getCoin_FouadAlkadri(90);
    assertSame(expected,result);
    }
@Test
void UTC_20_3(){
    int expected=0; //!!!!!
    int result= Mario.getCoin_FouadAlkadri(99);
    assertSame(expected,result);
    }
```

REQ	4.3.27 The system shall allow the user to add 1 life after collecting 100 coins.

Expected Result	coins=81;
	coins=91;
	coins=0; invokes get1Up();

Actual Result	coins=81;
	coins=91;
	coins=0; invokes get1Up();

Output	**PASS**

Test Case Number	UTC_21
Test Case	**The system shall allow the user to exit by pressing ESC in any screen/view.**
Code	

```
public void UTC_21 () {
    boolean start = true;
    MarioComponent marioC = new MarioComponent(10,10);
   marioC.toggleKey2(0x1B, true);
   assertEquals(start,marioC.after);
```

REQ	4.1.4 The system shall allow the user to exit by pressing ESC in any screen/view.
Expected Result	True
Actual Result	True
Output	PASS

Test Case Number	UTC_22
Test Case	**The system shall allow the user to navigate through the map using arrow keys.**
Code	

```
@Test
public void UTC_22(){
    MapScene map= Mockito.mock(MapScene.class,Mockito.CALLS_REAL_METHODS);
    map.setSeed(2111111);
    map.start_generate();
    map.tryWalking(0, -1);
    map.tryWalking(0, 0);
    int exp = 1;
    int act = map.getmoveTime();
    assertEquals(exp,act);
    }
```

REQ	4.2.1 The system shall allow the user to navigate through the map using arrow keys.
Expected Result	moveTime = 1
Actual Result	moveTime = 1
Output	PASS

Test Case Number	UTC_23
Test Case	**Verifying that Mario can move in the map using arrow keys.**
Code	

```
@Test
void UTC_23() {
    // create MapScene Object
    MapScene map= Mockito.mock(MapScene.class,Mockito.CALLS_REAL_METHODS);
    // create a 3x3 level that looks like this:
    // M R R
    // T W R
    // W W L
    // {R:road=3,M:mario,W:water=1,L:level=2,T:target tile=water}
    int[][] level = {{1,1,3},{1,1,3},{2,3,3}};
    map.setlevel(level);
    map.setxMario(0); // mario at x=0 so 0 = 16 * 0
    map.setyMario(32); // mario at y=2 so 32 = 16 * 2
    // call tryWalking with xd = 0 and yd = -1 (from figure above we need to move y 1 down and x shouldnt move)
    map.tryWalking(0, -1);
    // check if xmarioA, ymarioA = 0 (shouldnt move) and moveTime = 0 (shouldnt move)
    assertEquals(map.getxMarioA(),0 );
    assertEquals(map.getyMarioA(),0 );
    assertEquals(map.getmoveTime(),0 );
}
```

REQ	4.2.1 The system shall allow the user to navigate through the map using arrow keys.

Expected Result	xMarioA = 0;
	yMarioA=0;
	moveTime=0;

Actual Result	xMarioA = 0;
	yMarioA=0;
	moveTime=0;

Output	**PASS**

Test Case Number	UTC_24
Test Case	**Verifying that Mario can move in the map using arrow keys.**

Code	

```
@Test
void UTC_24() {
    // create MapScene Object
    MapScene map= Mockito.mock(MapScene.class,Mockito.CALLS_REAL_METHODS);
    // create a 3x3 level that looks like this:
    // R R R
    // W W M
    // W W T
    // {R:road=3,M:mario=road,W:water=1,L:level=2,T:target tile=level}
    int[][] level = {{1,1,3},{1,1,3},{2,3,3}};
    map.setlevel(level);
    map.setxMario(32); // mario at x=2 so 32 = 16 * 2
    map.setyMario(16); // mario at y=1 so 16 = 16 * 1
    // call tryWalking with xd = 0 and yd = -1 (from figure above we need to move y 1 down and x shouldnt move)
    map.tryWalking(0, -1);
    // check if xmarioA=0 (shouldnt move in x), ymarioA = -8 (should move down in y) and moveTime = 1 (should move)
    assertEquals(map.getxMarioA(),0 );
    assertEquals(map.getyMarioA(),-8 );
    assertEquals(map.getmoveTime(),1 );
}
```

REQ	4.2.1 The system shall allow the user to navigate through the map using arrow keys.

Expected Result	xMarioA = 0;
	yMarioA=-8;
	moveTime=1;

Actual Result	xMarioA = 0;
	yMarioA=-8;
	moveTime=1;

Output	**PASS**

Test Case Number	**UTC_25**
Test Case	**Verifying that Mario can move in the map using arrow keys.**
Code	

```
@Test
void UTC_25() {
    // create MapScene Object
    MapScene map= Mockito.mock(MapScene.class,Mockito.CALLS_REAL_METHODS);
    // create a 3x3 level that looks like this:
    // R R R
    // W W M
    // W W T
    // {R:road=3,M:mario=level,W:water=1,L:level=2,T:target tile=road}
    int[][] level = {{1,1,3},{1,1,3},{3,2,3}};
    map.setlevel(level);
    map.setxMario(32); // mario at x=2 so 32 = 16 * 2
    map.setyMario(16); // mario at y=1 so 16 = 16 * 1
    int[][] data = {{0,0,0},{0,0,0},{1,1,0}};
    map.setdata(data);
    // call tryWalking with xd = 0 and yd = -1 (from figure above we need to move y 1 down and x shouldnt move)
    map.tryWalking(0, -1);
    // check if xmarioA, ymarioA = 0 (shouldnt move) and moveTime = 0 (shouldnt move)
    assertEquals(map.getxMarioA(),0 );
    assertEquals(map.getyMarioA(),0 );
    assertEquals(map.getmoveTime(),0 );
}
```

REQ	4.2.1 The system shall allow the user to navigate through the map using arrow keys.

Expected Result	xMarioA = 0;
	yMarioA=0;
	moveTime=0;

Actual Result	xMarioA = 0;
	yMarioA=0;
	moveTime=0;

Output	**PASS**

Test Case Number	UTC_26
Test Case	**Verifying that Mario can move in the map using arrow keys.**
Code	

```
@Test
void UTC_26() {
    // create MapScene Object
    MapScene map= Mockito.mock(MapScene.class,Mockito.CALLS_REAL_METHODS);
    // create a 3x3 level that looks like this:
    // R R T
    // W W M
    // W W R
    // {R:road=3,M:mario=level,W:water=1,L:level=2,T:target tile=road}
    int[][] level = {{1,1,3},{1,1,3},{3,2,3}};
    map.setlevel(level);
    map.setxMario(32); // mario at x=2 so 32 = 16 * 2
    map.setyMario(16); // mario at y=1 so 16 = 16 * 1
    int[][] data = {{0,0,0},{0,0,0},{1,1,0}};
    map.setdata(data);
    // call tryWalking with xd = 0 and yd = -1 (from figure above we need to move y 1 up and x shouldnt move)
    map.tryWalking(0, 1);
    // check if xmarioA=0 (shouldnt move in x), ymarioA = 8 (should move up in y) and moveTime = 1 (should move)
    assertEquals(map.getxMarioA(),0 );
    assertEquals(map.getyMarioA(),8 );
    assertEquals(map.getmoveTime(),1 );
}
```

REQ	4.2.1 The system shall allow the user to navigate through the map using arrow keys.

Expected Result	xMarioA = 0;
	yMarioA=8;
	moveTime=1;

Actual Result	xMarioA = 0;
	yMarioA=8;
	moveTime=1;

Output	**PASS**

Test Case Number	UTC_27
Test Case	**When Mario losses all of his lives, his lives should be reseted to 3 .**
Code	

```
    @Test
    void UTC_27() {
        LevelScene level=Mockito.mock(LevelScene.class);
        Mario mar=new Mario(level);
        mar.lives=2;
//      System.out.println(mar.lives);
        mar.resetStatic();
//      Mario is supposed to have three lives after resetStatic is called
        int result= mar.lives;
        int expected=3;
        assertEquals(expected,result);
    }
```

REQ	4.3.8 The system shall spawn Mario with 3 lives
Expected Result	Mario's number of lives should be changed to 3
Actual Result	Marios' lives are reseted to 3
Output	**PASS**

Test Case Number	UTC_28
Test Case	When Mario losses all of his lives, he should spawn without a power up - he is not “Large”-.
Code	

```
    @Test
    public void UTC_28() {
        LevelScene level=Mockito.mock(LevelScene.class);
        Mario mar=new Mario(level);
        mar.large=true;
//      System.out.println(mar.large);
        mar.resetStatic();
//      Mario is not supposed to be large after resetStatic is called
        boolean result= mar.large;
        boolean expected=false;
        assertEquals(expected,result);
    }
```

```
    @Test
    void UTC_28() {
        LevelScene level=Mockito.mock(LevelScene.class);
        Mario mar=new Mario(level);
        mar.large=true;
//      System.out.println(mar.large);
        mar.resetStatic();
//      Mario is not supposed to be large after resetStatic is called
        boolean result= mar.large;
        boolean expected=false;
        assertEquals(expected,result);
    }
```

REQ	4.3.9 The system shall spawn Mario without power up
Expected Result	Mario spawn without a power, so he isn’t large
Actual Result	Mario didn’t have power up
Output	PASS

Test Case Number	UTC_29
Test Case	When Mario losses all of his lives, he should spawn without a fireball.
Code	

```
@Test
public void UTC_29() {
    LevelScene level=Mockito.mock(LevelScene.class);
    Mario mar=new Mario(level);
    mar.fire=true;
    System.out.println(mar.fire);
    mar.resetStatic();
    Mario is not supposed to have fire ability affect after resetStatic is called
    boolean result= mar.fire;
    boolean expected=false;
    assertEquals(expected,result);
}
```

	`@Test`
	`void UTC_29() {`
	`LevelScene level=Mockito.mock(LevelScene.class);`
	`Mario mar=new Mario(level);`
	`mar.fire=true;`
	`// System.out.println(mar.fire);`
	`mar.resetStatic();`
	`// Mario is not supposed to have fire ability affect after resetStatic is called`
	`boolean result= mar.fire;`
	`boolean expected=false;`
	`assertEquals(expected,result);`
	`}`

REQ	4.3.10 The system shall spawn Mario without fireball
Expected Result	Mario spawn without a fireball
Actual Result	Mario didn't have fireball
Output	**PASS**

Test Case Number	**UTC_30**
Test Case	**When Mario losses all of his lives, he should spawn with 0 coins**

Code	`@Test`
	`void UTC_30() {`
	`LevelScene level=Mockito.mock(LevelScene.class);`
	`Mario mar=new Mario(level);`
	`mar.coins=55;`
	`// System.out.println(mar.coins);`
	`mar.resetStatic();`
	`// The coins is supposed to be zero`
	`int result= mar.coins;`
	`int expected=0;`
	`assertEquals(expected,result);`
	`}`

REQ	4.3.11 The system shall spawn Mario without coins
Expected Result	Mario spawn with coins
Actual Result	Mario had zero coins
Output	**PASS**

21. Integration testing

Integration Test Cases	Passed Test cases	Failed Test cases

Test case: 36	Number of TC's: 32	Number of TC's: 4

Test Case Number	ITC1_1
Test Case	**Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards**
Code	

```
@Test
void ITC_1_1() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),2111111111,1,0);
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = 15;
    mario.y = 11;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if large
    assertEquals(mario.large,true);
}
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
Expected Result	Large = true;
Actual Result	Large = true;
Output	**PASS**

Test Case Number	ITC1_2

Test Case	**Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards**
Code	

```
}
@Test
void ITC_1_2() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),2111111111,1,0)
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = 15;
    mario.y = 11;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if fire
    assertEquals(mario.fire,true);
}
@Test
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
Expected Result	fire= true;
Actual Result	fire = false;
Output	**FAIL**

Test Case Number	**ITC1_3**
Test Case	**Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards**
Code	

```
@Test
void ITC_1_3() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),2111111111,1,0);
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = 15;
    mario.y = 11;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if power up time is 18
    assertEquals(mario.getPowerUpTime(),18);
}
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
Expected Result	powerUpTime= 18;
Actual Result	powerUpTime = true;
Output	**PASS**

Test Case Number	ITC2_1
Test Case	**Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards**
Code	

```
@Test
void ITC_2_1() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),2111111111,1,0);
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = 16;
    mario.y = 12;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if large
    assertEquals(mario.large,true);
}
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
Expected Result	Large = true;
Actual Result	Large = fail;
Output	**FAIL**

Test Case Number	ITC2_2
Test Case	**Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards**
Code	

```
}
@Test
void ITC_2_2() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),2111111111,1,0);
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = 16;
    mario.y = 12;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if fire
    assertEquals(mario.fire,true);
}
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
Expected Result	fire= true;
Actual Result	fire = false;
Output	**FAIL**

Test Case Number	ITC2_3
Test Case	**Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards**
Code	

```
}
@Test
void ITC_2_3() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),2111111111,1,0);
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = 16;
    mario.y = 12;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if power up time is 18
    assertEquals(mario.getPowerUpTime(),18);
}
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
Expected Result	powerUpTime= 18;
Actual Result	powerUpTime = 0;
Output	**FAIL**

Test Case Number	ITC3_1
Test Case	**Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards**
Code	

```
@Test
void ITC_3_1() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),2111111111,1,0);
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = 17;
    mario.y = 13;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if large
    assertEquals(mario.large,false);
}
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
Expected Result	Large = false;
Actual Result	Large = false;
Output	**PASS**

Test Case Number	ITC3_2
Test Case	**Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards**
Code	

```
@Test
void ITC_3_2() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),2111111111,1,0);
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = 17;
    mario.y = 13;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if fire
    assertEquals(mario.fire,false);
}
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
Expected Result	fire= false;
Actual Result	fire = false;
Output	**PASS**

Test Case Number	ITC3_3
Test Case	**Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards**
Code	

```
@Test
void ITC_3_3() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),2111111111,1,0);
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = 17;
    mario.y = 13;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if power up time is 18
    assertEquals(mario.getPowerUpTime(),0);
}
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
Expected Result	powerUpTime= 0;
Actual Result	powerUpTime = 0;

Output	PASS

Test Case Number	ITC4_1
Test Case	**Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards**
Code	

```
@Test
void ITC_4_1() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),2111111111,1,0);
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = -16;
    mario.y = -12;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if large
    assertEquals(mario.large,true);
}
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
Expected Result	Large = true;
Actual Result	Large = false;
Output	FAIL

Test Case Number	ITC4_2
Test Case	**Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards**
Code	

```
@Test
void ITC_4_2() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),2111111111,1,0);
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = -16;
    mario.y = -12;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if fire
    assertEquals(mario.fire,true);
}
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**

Expected Result	fire= true;
Actual Result	fire = false;
Output	**FAIL**

Test Case Number	**ITC4_3**
Test Case	**Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards**
Code	

```
@Test
void ITC_4_3() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),211111111
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = -16;
    mario.y = -12;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if power up time is 18
    assertEquals(mario.getPowerUpTime(),18);
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
Expected Result	powerUpTime= 18;
Actual Result	powerUpTime = 0;
Output	**FAIL**

Test Case Number	**ITC5_1**
Test Case	**Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards**

Code	

```
@Test
void ITC_5_1() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),21111
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = -17;
    mario.y = -13;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if large
    assertEquals(mario.large,false);
}
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
Expected Result	Large = false;
Actual Result	Large = false;
Output	**PASS**

Test Case Number	**ITC5_2**
Test Case	**Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards**
Code	

```
@Test
void ITC_5_2() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),2111111111,1,0);
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = -17;
    mario.y = -13;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if fire
    assertEquals(mario.fire,false);
}
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
Expected Result	fire= false;
Actual Result	fire = false;
Output	**PASS**

Test Case Number	**ITC5_3**

Test Case	Checking if Mario consumes Chili powerup up on impact, and if he becomes Red Mario afterwards
Code	

```
@Test
void ITC_5_3() {
    // create a level scene
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),2111111111,1,0);
    // create a mario
    Mario mario = new Mario(world);
    // link mario and world
    world.mario = mario;
    // create a fireflower
    FireFlower fire = new FireFlower(world,0,0);
    // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
    mario.large = false;
    mario.fire = false;
    mario.x = -17;
    mario.y = -13;
    // now call collidecheck
    fire.collideCheckAziz();
    // check if power up time is 18
    assertEquals(mario.getPowerUpTime(),0);
}
```

REQ	4.3.13 If Mario takes a Chili powerup, then Mario should increase in size, and become Red Mario. 4.3.14 If Mario collides with a Chili powerup, he should consume it**.**
Expected Result	powerUpTime= 0;
Actual Result	powerUpTime = 0;
Output	**PASS**

Test Case Number	ITC_6_1
Test Case	**If Mario collides with a Cake powerup, he should consume it. (Check if Mario got large)**
Code	

```
@Test
public void ITC_6_1 () {
    LevelScene ls = Mockito.mock(LevelScene.class);
    Mushroom msh = new Mushroom(ls,15,15);
    Mario mario = new Mario(ls);
    ls.mario = mario;
    ls.mario.x = 4;
    ls.mario.y = 4;
    msh.collideCheck2();
    boolean exp = true;
    boolean act = ls.mario.large;
    assertEquals(exp,act);
}
```

REQ	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	mario.large= True
Actual Result	mario.large= True
Output	**PASS**

Test Case Number	ITC_6_2
Test Case	**If Mario collides with a Cake powerup, he should consume it. (Check if the power Time increased)**
Code	

```
@Test
public void ITC_6_2 () {
        // power up works

        LevelScene ls = Mockito.mock(LevelScene.class);
        Mushroom msh = new Mushroom(ls,0,0);
        Mario mario = new Mario(ls);
        ls.mario = mario;
        ls.mario.large = false;
        ls.mario.x = 4;
        ls.mario.y = 4;
        msh.collideCheck2();
        int exp = 18;
        int pg = ls.mario.getPowerUpTime();
        assertEquals(exp,pg);
    }
```

REQ	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	power= 18
Actual Result	get_power() return =18
Output	**PASS**

Test Case Number	ITC_7_1
Test Case	**If Mario collides with a Cake powerup, he should consume it. The range (x=16,y=12)**
Code	

```
    @Test
public void ITC_7_1 () {
        LevelScene ls = Mockito.mock(LevelScene.class);
        Mushroom msh =  new Mushroom(ls,0,0);
        Mario mario = new Mario(ls);
        ls.mario = mario;
        ls.mario.large = false;
        ls.mario.x = 16;
        ls.mario.y = 12;
        msh.collideCheck2();
        boolean exp = true;
        boolean act = ls.mario.large;
        assertEquals(exp,act);
    }
```

REQ	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	Mario.large = True
Actual Result	mario.large = True
Output	**FAIL**

Test Case Number	**ITC_7_2**
Test Case	**If Mario collides with a Cake powerup, he should consume it. The range (x=16,y=12)**
Code	

```
    @Test
public void ITC_7_2 () {
        LevelScene ls = Mockito.mock(LevelScene.class);
        Mushroom msh =  new Mushroom(ls,0,0);
        Mario mario = new Mario(ls);
        ls.mario = mario;
        ls.mario.large = false;
        ls.mario.x = 16;
        ls.mario.y = 12;
        msh.collideCheck2();
        int exp = 18;
        int pg = ls.mario.getPowerUpTime();
        assertEquals(exp,pg);
    }
```

REQ	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	powerTime = 18
Actual Result	powerTime = 18
Output	**FAIL**

Test Case Number	**ITC_8_1**
Test Case	**If Mario collides with a Cake powerup, he should consume it. (the range is out of {x=17,y=13} check the large**

Code	

```java
@Test
public void ITC_8_1 () {
        LevelScene ls = Mockito.mock(LevelScene.class);
        Mushroom msh = new Mushroom(ls,0,0);
        Mario mario = new Mario(ls);
        ls.mario = mario;
        ls.mario.x = 17;
        ls.mario.y = 13;
        ls.mario.large = false;
        msh.collideCheck2();
        boolean exp = false;
        boolean act = ls.mario.large;
        assertEquals(exp,act);
    }
```

REQ	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	Mario.large = false
Actual Result	Mario.large = false
Output	**PASS**

Test Case Number	**ITC_8_2**
Test Case	**If Mario collides with a Cake powerup, he should consume it. (the range is out of {x=17,y=13} check the power up**
Code	

```java
@Test
public void ITC_8_2 () {
    LevelScene ls = Mockito.mock(LevelScene.class);
    Mushroom msh = new Mushroom(ls,0,0);
    Mario mario = new Mario(ls);
    ls.mario = mario;
    ls.mario.large = false;
    ls.mario.x = 17;
    ls.mario.y = 13;
    msh.collideCheck2();
    int exp = 0;
    int pg = ls.mario.getPowerUpTime();
    assertEquals(exp,pg);
}
```

REQ	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	Power up = 0
Actual Result	Power up = 0
Output	**PASS**

Test Case Number	**ITC_9_1**

Test Case	**If Mario collides with a Cake powerup, he should consume it. The range (x=-16,y=-12) Check the large**
Code	

```
@Test
public void ITC_9_1 () {
        LevelScene ls = Mockito.mock(LevelScene.class);
        Mushroom msh = new Mushroom(ls,-32,-32);
        Mario mario = new Mario(ls);
        ls.mario = mario;
        ls.mario.large = false;
        ls.mario.x = -16;
        ls.mario.y = -12;
        msh.collideCheck2();
        boolean exp = true;
        boolean act = ls.mario.large;
        assertEquals(exp,act);
    }
```

REQ	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	True
Actual Result	True
Output	**FAIL**

Test Case Number	**ITC_9_2**
Test Case	**If Mario collides with a Cake powerup, he should consume it. The range (x=-16,y=-12) Check the power up**
Code	

```
public void ITC_9_2() {
    LevelScene ls = Mockito.mock(LevelScene.class);
    Mushroom msh = new Mushroom(ls,-32,-32);
    Mario mario = new Mario(ls);
    ls.mario = mario;
    ls.mario.large = false;
    ls.mario.x = -16;
    ls.mario.y = -12;
    msh.collideCheck2();
    int exp = 18;
    int pg = ls.mario.getPowerUpTime();
    assertEquals(exp,pg);
}
```

REQ	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	powerTime = 18
Actual Result	powerTime = 18
Output	**FAIL**

Test Case Number	ITC_10_1
Test Case	**If Mario collides with a Cake powerup, he should consume it. Out of the range {x=-17,y=-13}**
Code	

```
    @Test
public void ITC_10_1 () {
        LevelScene ls = Mockito.mock(LevelScene.class);
        Mushroom msh = new Mushroom(ls,0,0);
        Mario mario = new Mario(ls);
        ls.mario = mario;
        mario.large = false;
        ls.mario.x = -17;
        ls.mario.y = -13;
        ls.mario.large = false;
        msh.collideCheck2();
        boolean exp = false;
        boolean act = ls.mario.large;
        assertEquals(exp,act);
    }
```

REQ	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	False
Actual Result	False
Output	**PASS**

Test Case Number	ITC_10_2
Test Case	**If Mario collides with a Cake powerup, he should consume it. (the range is out of {x=-16,y=-13} check the power up**
Code	

```
    @Test

public void ITC_10_2() {
        LevelScene ls = Mockito.mock(LevelScene.class);
        Mushroom msh = new Mushroom(ls,0,0);
        Mario mario = new Mario(ls);
        ls.mario = mario;
        ls.mario.large = false;
        ls.mario.x = -17;
        ls.mario.y = -13;
        msh.collideCheck2();
        int exp = 0;
        int pg = ls.mario.getPowerUpTime();
        assertEquals(exp,pg);
    }
```

REQ	4.3.15 If Mario collides with a Cake powerup, he should consume it. 4.3.16 If Mario consumes a Cake powerup, then Mario should increase in size.
Expected Result	Power up = 0

Actual Result	Power up = 0
Output	**PASS**

Test Case Number	**ITC_11**
Test Case	**Checking if Mario gets hurt when colliding with an enemy.**
Code	

```
@Test
    void ITC_11() {
        // create a level scene
        LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),Mockito.mock(MarioComponent.class),2111111111,1,0);
        // create a mario
        Mario mario = new Mario(world);
        // link mario and world
        world.mario = mario;
        // create a spiky enemy
        Enemy enemy = new Enemy(world,0,0,0,2,false);
        // set large and fire to false + set {x,y} position of mario to 0 making xMarioD and yMarioD in {-32,32}
        mario.large = false;
        mario.fire = false;
        mario.x = 4;
        mario.y = 4;
        // now call collidecheck
        enemy.collideCheckAziz();
        // check if deathTime is 1 meaning mario died from the spiky enemy
        assertEquals(mario.deathTime,1);
    }
```

REQ	4.3.17 If Mario collides with an enemy, then he should get hurt.
Expected Result	deathTime = 1;
Actual Result	deathTime =1;
Output	**PASS**

Test Case Number	**ITC12**
Test Case	Checking if Mario gets hurt when colliding with an enemy
Code	

```
    }
    @Test
    void ITC_12(){
        LevelScene level=Mockito.mock(LevelScene.class);
        Enemy enemy= new Enemy(level,0,0,1,2,false);
        Mario mar=new Mario(level);
        level.mario=mar;
        mar.large=false;
        mar.setInvulnerableTime(0);
        mar.pausedHisham=false;
        mar.deathTime=0;
        mar.fire=false;
        mar.x=12;
        mar.y=12;
        enemy.collideCheckHisham();
//      Mario here is supposed to be touching the enemy (no gap between them) and he is supposed to
//      die
        int expected=1;
        int result=mar.deathTime;
        assertEquals(expected,result);
    }
```

REQ	4.3.17 If Mario collides with an enemy, then he should get hurt.
Expected Result	deathTime=1;
Actual Result	deathTime=0;

Output	FAIL

Test Case Number	ITC_13
Test Case	If Mario collides with an enemy, then he should get hurt. (This is for out of the Collison range {x=13, y=13}
Code	

```
@Test
public void ITC_13() {
LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),
                        Mockito.mock(MarioComponent.class),2111111111,1,0);
    Mario mario = new Mario(world);
    world.mario = mario;
    Enemy enemy = new Enemy(world,0,0,0,2,false);
    mario.large = false;
    mario.fire = false;
    mario.x = 13;
    mario.y = 13;
    enemy.collideCheck();
    assertEquals(mario.deathTime,0);
    }
```

REQ	4.3.17 If Mario collides with an enemy, then he should get hurt.
Expected Result	deathTime = 0
Actual Result	deathTime = 0
Output	PASS

Test Case Number	ITC_14
Test Case	If Mario collides with an enemy, then he should get hurt. (this is in the collison range {x=-12,y=-12}
Code	

```
@Test
public void ITC_14() {
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),
                        Mockito.mock(MarioComponent.class),2111111111,1,0);
    Mario mario = new Mario(world);
    world.mario = mario;
    Enemy enemy = new Enemy(world,0,0,0,2,false);
    mario.large = false;
    mario.fire = false;
    mario.x = -12;
    mario.y = -12;
    enemy.collideCheckSerry();
    int exp = 1;
    int act = mario.deathTime;
    assertEquals(act,act);
    }
```

REQ	4.3.17 If Mario collides with an enemy, then he should get hurt.

Expected Result	deatTime = 1
Actual Result	deatTime = 1
Output	**PASS**

Test Case Number	**ITC_15**
Test Case	**If Mario collides with an enemy, then he should get hurt. (This is for out of the Collison range {x=-33, y=-33}**
Code	

```
@Test
public void ITC_15() {
    LevelScene world = new LevelScene(Mockito.mock(GraphicsConfiguration.class),
                                      Mockito.mock(MarioComponent.class),2111111111,1,0);
    Mario mario = new Mario(world);
    world.mario = mario;
    Enemy enemy = new Enemy(world,0,0,0,2,false);
    mario.large = false;
    mario.fire = false;
    mario.x = -13;
    mario.y = -13;
    enemy.collideCheck();
    assertEquals(mario.deathTime,0);
}
```

REQ	4.3.17 If Mario collides with an enemy, then he should get hurt.
Expected Result	deatTime = 0
Actual Result	deatTime = 0
Output	**PASS**

Test Case Number	ITC_16
Test Case	**If Mario jumps on Red Kubba then it should become a Shell.**
Code	

```java
    // Functional Req If Mario jumps on Red Kubba then it should become a Shell.
    //Technique Used Cause and effect.

//Red
    @Test
    void ITC_16() {

        LevelScene level=Mockito.mock(LevelScene.class,Mockito.CALLS_REAL_METHODS);
        level.setSpritesToAdd(new ArrayList<Sprite>());
        Enemy enemy= new Enemy(level,32,32,1,0,false);
        Mario mar=new Mario(level);
        level.mario=mar;
        mar.large=false;
        mar.setInvulnerableTime(0);
        mar.paused_FouadAlkadri=false;
        mar.deathTime=0;
        mar.fire=false;
        mar.x=33;
        mar.y=32;
        mar.ya=1;
        enemy.spriteContext=level;
        //System.out.println(level.spritesToAdd.size());
        enemy.collideCheck_FouadAlkadri();
        //System.out.println(level.spritesToAdd.size());

        boolean expected=true;
        boolean result=enemy.Valuecheck_FouadAlkadri;

        assertEquals(expected,result);
    }
```

REQ	4.3.22 If Mario jumps on Red Kubba then it should become a Shell.
Expected Result	True
Actual Result	Ture
Output	**PASS**

Test Case Number	ITC_17
Test Case	If Mario jumps on Green Kubba then it should become a Shell.
Code	

```
@Test
void ITC_17() {

    LevelScene level=Mockito.mock(LevelScene.class,Mockito.CALLS_REAL_METHODS);
    level.setSpritesToAdd(new ArrayList<Sprite>());
    Enemy enemy= new Enemy(level,32,32,1,1,false);
    Mario mar=new Mario(level);
    level.mario=mar;
    mar.large=false;
    mar.setInvulnerableTime(0);
    mar.paused_FouadAlkadri=false;
    mar.deathTime=0;
    mar.fire=false;
    mar.x=33;
    mar.y=32;
    mar.ya=1;
    enemy.spriteContext=level;
    //System.out.println(level.spritesToAdd.size());
    enemy.collideCheck_FouadAlkadri();
    //System.out.println(level.spritesToAdd.size());

    boolean expected=true;
    boolean result=enemy.Valuecheck_FouadAlkadri;

    assertEquals(expected,result);

}
```

REQ	4.3.23 If Mario jumps on Green Kubba then it should become a Shell.
Expected Result	True
Actual Result	Ture
Output	PASS

Test Case Number	ITC_18
Test Case	**If Mario collides with an enemy during Invulnerability, he should not get damaged.**
Code	

```
//functional req:  If Mario collides with an enemy during vulnerability, he should not get damaged.
// Technique used: Cause and effect
@Test
void ITC_18_1() {
    LevelScene level=Mockito.mock(LevelScene.class);
    Mario mar=new Mario(level);
    mar.large=true;
    mar.fire=true;
    mar.getHurt_FouadAlkadri();
    boolean expected=true;

    assertSame(expected,mar.paused_FouadAlkadri);

}
@Test
void ITC_18_2() {
    LevelScene level=Mockito.mock(LevelScene.class);
    Mario mar=new Mario(level);
    mar.large=true;
    mar.fire=true;
    mar.getHurt_FouadAlkadri();

    int expected= -18 ;
    int result= mar.powerUpTime_FouadAlkadri;
    assertSame(expected,result);
}
```

REQ	4.3.25 If Mario collides with an enemy during vulnerability, he should not get damaged
Expected Result	**paused=true;** **powerUpTime= -18 ;**
Actual Result	**paused=true;** **powerUpTime= -18 ;**
Output	**PASS**

Test Case Number	ITC_19
Test Case	The user shall start spawn in the LevelScene after clicking S in MapScene.
Code	

```
//functional req: The user shall start spawn in the LevelScene after clicking S in MapScene.
//Technique use: Branch Coverage.
@Test
void ITC_19() {
    MarioComponent marc=new MarioComponent(21,16);
    MapScene level=Mockito.mock(MapScene.class,Mockito.CALLS_REAL_METHODS);
    LevelScene level2=Mockito.mock(LevelScene.class);
    Mario mar=new Mario(level2);

    level.setMarioComponent(marc);
    level.setMoveTime(-1);
    level.setCanEnterLevel(true);
    level.keys[4]=true;
    level.setxMarioA(1);
    level.setyMarioA(1);

    int[][] Data1=new int[21][16];
    Data1[0][0]=1;
    level.setData(Data1);

    int[][] level1=new int[21][16];
    level1[0][0]=2;
    level.setLevel(level1);

    level.TILE_ROAD_FouadAlakdri=3;


    level.tick_FouadAlakdri();

    boolean expected=true;
    boolean result=level.Valuecheck_FouadAlakdri;

    assertEquals(expected,result);


}
```

REQ	4.3.26 The system shall allow the user start spawn in the LevelScene after clicking S in MapScene.
Expected Result	True
Actual Result	True
Output	PASS

Test Case Number	ITC_20
Test Case	game starts after pressing S key by the user, such that a new map will be generated
Code	

```java
@Test
void ITC_20(){
    MarioComponent mar=new MarioComponent(32,32);
    TitleScene title=Mockito.mock(TitleScene.class,Mockito.CALLS_REAL_METHODS);
    MapScene map=Mockito.mock(MapScene.class,Mockito.CALLS_REAL_METHODS);
    title.setComponent(mar);
    map.setMarioComponent(mar);
    mar.setMapScene(map);
    title.setWasDown(false);
    title.keys[4]=true;
    boolean result=mar.getScene()==mar.getMapScene();
    boolean expectedBeforeChanging=false;
    //Checking if scene = map before running title.tick2()--> They should not be equal
    assertEquals(expectedBeforeChanging,result);
    title.tickHisham();
    boolean expected=true;
    result=mar.getScene()==mar.getMapScene();
    //Checking if scene = map after running title.tick2()--> They should be equal
    assertEquals(expected,result);
}
```

REQ	4.1.1 The system shall allow the user to start the game by pressing S.
Expected Result	A new map will be generated,
Actual Result	A new map is generated
Output	PASS

22. System Testing:

22.1 Method Used:

22.1.1 Using Functional Testing:

Functional testing finds differences between functional requirements and the implemented system.

22.1.2 Alpha testing:

UAT is conducted by the testers to ensure that system satisfies the contractual acceptance criteria before being signed-off as meeting user needs.

Note: We'll use a functional testing method that will make sure our system test case meets the requirements, which is proved and ensured by passing all the related unit test cases and integration test cases, as well as the post condition -where it is applicable-. After that, we'll do alpha testing of the post condition of functional testing and applying the scenario that's provided to ensure that it will pass our system test cases.

Functional Testing:

ID	STC1
Use case Name	**Getting hurt**
Description	**Mario should change the Kubba (red/green) to a shell if collide with it from top and make it move if touch it from the sides**
Functional Requirements covered by feature	• 4.3.2 • 4.3.4 • 4.3.12 • 4.3.17 • 4.3.22 • 4.3.23
Related unit and integration test cases.	• ITC 11, 12, 13, 14, 15, 16, 17

Pre-Conditions	Use a computer with windows as an operating system.
Event Sequence	
Input events	**Output events**
Mario Goes up the enemy (red/green Kubba)	Kubba change to shell
Mario touches the side	Shell starts moving
Post Conditions	If the shell collides with Mario dies
Test Result, Run by	**Pass | Serry**

Alpha testing:

Output	Input
	**1. Go to the Kubba**

	**2. Hit the Kubba on the head (change to shell)**
	**3.go to hit the shell**
	**4.hit the shell**
	**5.the shell starts moving to right**
Result:	**The system test case passed the Alpha testing.**

Functional Testing:

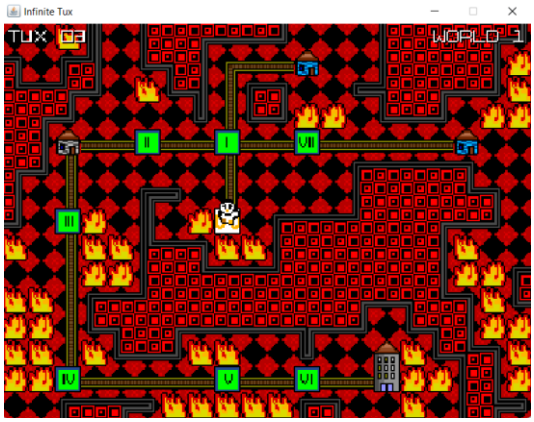
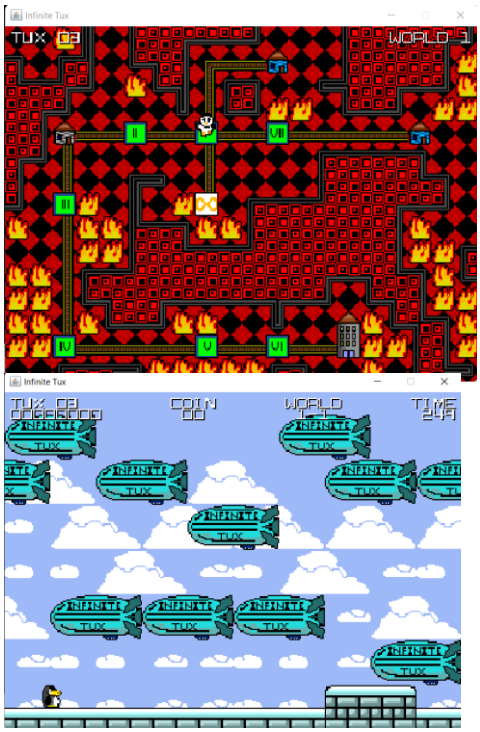
ID	STC2
Use case Name	**Game over**
Description	**When Mario dies or the timer of the game finishes, "Game over screen" should be displayed**
Functional Requirements covered	● 4.3.2 ● 4.3.6 ● 4.3.12 ● 4.3.17
Related unit and integration test cases.	● ITC 11, **12**, 13, 14, 15
Pre-Conditions	**Use a computer with windows as an operating system.**
Event Sequence	
Input events	**Output events**
Start the game	"Main menu" shows up
Press s	"Level Selection Map" show up
Select a level then press S	"In game" screen show up

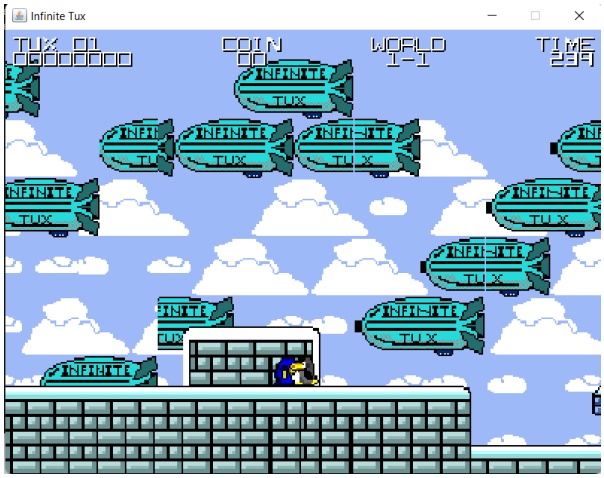
Go through enemy	Get damage
Mario die	Game over
Post Conditions	Game over screen
Test Result, Run by	**Pass| Fouad**

Alpha testing:

Implementation:

Output	**Input**
	**1-** Start the game:

	**2-** Press S:
	**3-** Select a level then press S.

	**4-** Go through enemy.
	5- Mario die
Result:	**The system test case passed the Alpha testing, and Mario dies immediately when an Enemy touches Mario's "Hurt Box".**

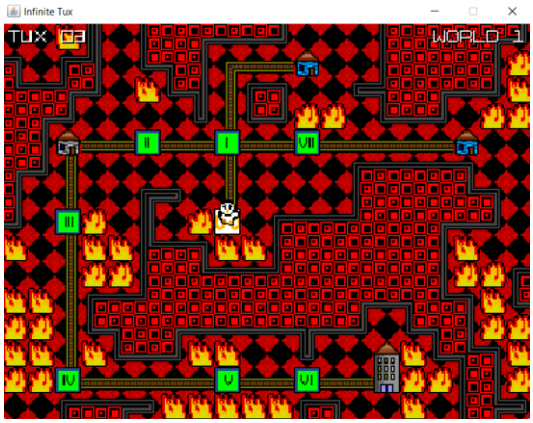
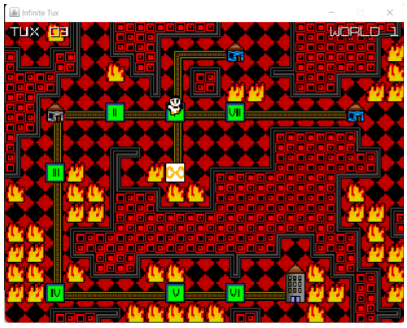
Functional Testing:

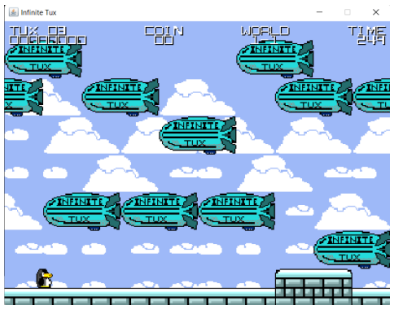
ID	STC3
Use Case Name	**Chili powerup**
Description	Chili powerup should make the Mario red and large and the Mario should be able to throw fireballs that kill enemies.
Functional Requirements covered	• 4.3.13 • 4.3.14 • 4.3.19 • 4.3.21
Related unit and integration test cases.	• UTC 10, 11, 12, 13, 14 • ITC 1, 2, 3, 4, 5
Pre-Conditions	**Use computer with windows as operating system**

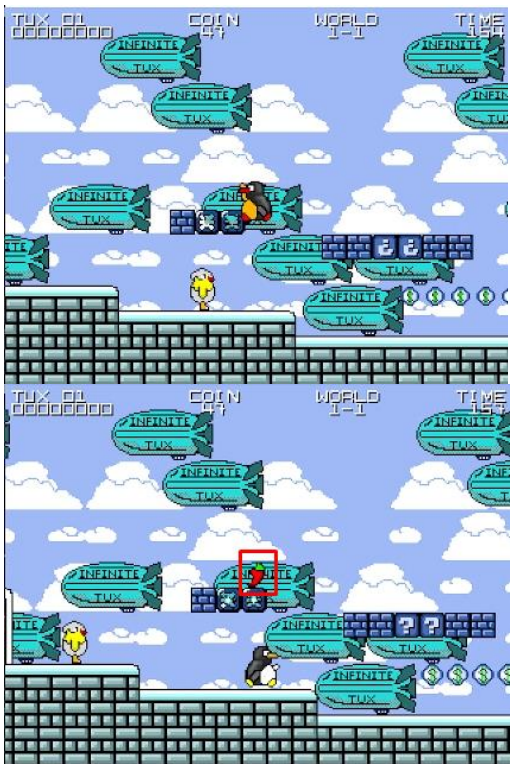
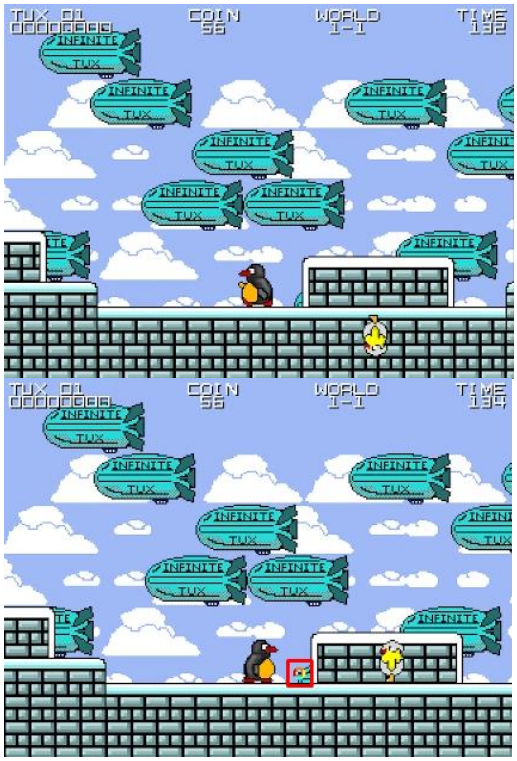
Event Sequence	
Input events	**Output events**
Start the game	“Main menu” shows up
Press s	“Level Selection Map” show up
Select a level then press S	“In game” screen show up
Take “Cake” powerup	Mario become “Large”
Take “Chili” powerup	Mario become red and have fireball ability
Shoot fireball into enemy	Enemy get damage
Post Conditions	Enemy dies
Test Result, Run by	Pass | Hisham

Alpha testing:

Implementation:

Output	Input
	**1-** Start the game:
	**2-** Press S:
	**3-** Select a level then press S.

  	**4-** Take “Cake” powerup

	**5-** Take “Chili” powerup
	**6-** Shoot fireball into enemy

Result:	**The system test case passed the Alpha testing, and enemies died after the Fireball hits the Enemy's "Hurt Box".**

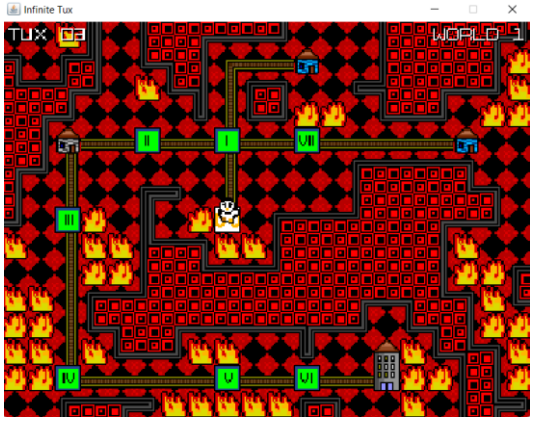
Functional Testing:

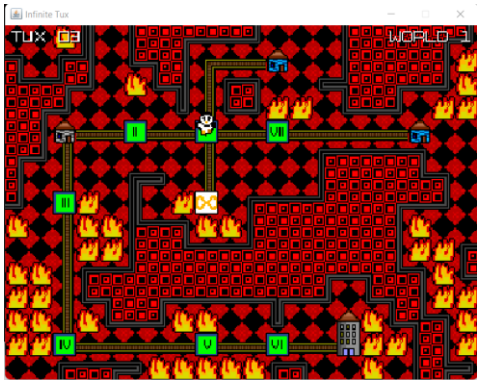
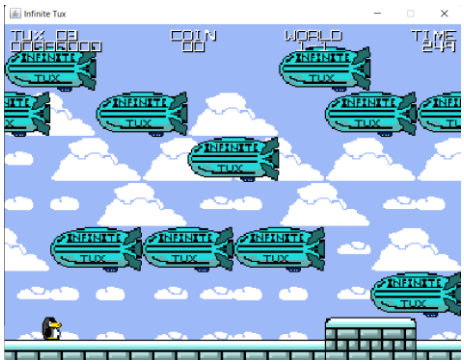
ID	**STC4**
Use Case Name	**Selecting a level**
Description	Player selects a level from the "level selection map" and starts the level.
Functional Requirements covered	• 4.2.1 • 4.2.2
Related unit and integration test cases.	• UTC 22, 23, 24, 25, 26 • ITC 19
Pre-Conditions	**Use computer with windows as operating system**
Event Sequence	
Input events	**Output events**
Start the game	"Main menu" shows up
Press s	"Level Selection Map" show up

Select a level using "arrow keys"	"Level navigation Mario" should be on the specified level
Press s	"In game" screen show up
Post Conditions	Game starts
Test Result, Run by	Pass | Abdullah

Alpha Testing:

Implementation:

Output	**Input**
	**1-** Start the game:
	2- Press S:

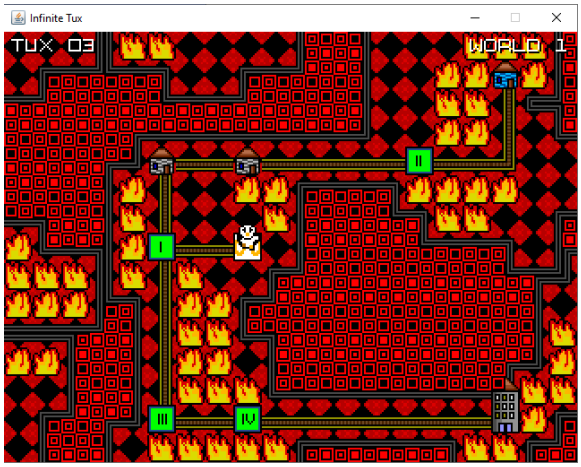
	3- Select a level
	4- Press s
Result:	The system test case passed the Alpha testing since the game loaded as expected after navigating to the desired level.

Functional Testing:

ID	STC5
Use Case Name	**UI functions**
Description	This test case is related to UI functions, such as closing the game with ESC and starting the game at the beginning by pressing S.
Functional Requirements covered	• 4.1.1 • 4.1.4
Related unit and integration test cases.	• UTC 22, 23, 24, 25, 26 • ITC 20
Pre-Conditions	**Use computer with windows as operating system**
	Event Sequence
Input events	**Output events**
Start the game	"Main menu" shows up
Press s	"Level Selection Map" show up
Press ESC	Game should close
Post Conditions	Game is closed
Test Result, Run by	Pass | Abdulaziz

Alpha testing:

Implementation:

Output	Input
	**1-** Start the game:
	2- Press S:

	3- Press ESC
Result:	The system test case passed the Alpha testing since the game loaded, the map appeared after pressing S, and the game closed when ESC was pressed.