SE 3XA3: Test Report
Mario Level X

210, Group 210
Edward Liu, liuz150
Ahmad Gharib, ghariba
Connor Czarnuch, czarnucc

April 7, 2020

Contents

List of Tables

1 Functional Requirements Evaluation

1.1 Menu

Test name: FR-M-1
Results: The game is able to load into the level player every time that this test is performed.

Test name: FR-M-2
Results: The game is able to load into the level editor every time that this test is performed.

Test name: FR-M-3
Results: The game is able to load the same level that is selected every time this test is performed.

Test name: FR-M-4
Results: When the up and down arrow keys are pressed, the item that is selected changes accordingly.

Test name: FR-M-5
Results: No exception is generated while this test is performed, no matter how quickly the escape key is pressed.

Test name: FR-M-6
Results: When other keys are pressed, no unexpected output is generated from the game. The same results occur when key combinations are pressed.

1.2 Level Player

Test name: FR-LP-1

Table 1: **Revision History**

Date	**Version**	**Notes**
Apr. 6 2020	1.0	Final Revision

Results: Every block that is placed during the level edit mode is saved in the JSON file when right click is pressed. If the same location is pressed multiple times, there will be multiple entries at the same location in the JSON file.

Test name: FR-LP-2
Results: Mario collides into each block type correctly when coming into contact on screen.

Test name: FR-LP-3
Results: During the course of the level, Mario does not move past the middle of the screen. The screen does not move backwards when Mario approaches the left side of the screen.

Test name: FR-LP-4
Results: When Mario dies off screen or by an enemy, the death animation is shown and Mario does not take any movement inputs and no exception is generated.

Test name: FR-LP-5
Results: During gameplay, when any unmapped or unintentional key is pressed no exception is generated and nothing unexpected happened.

1.3 Level Editor

Test name: FR-LE-1
Results: When right click is pressed during the level editor, the JSON is saved and blocks are in the correct position.

Test name: FR-LE-2
Results: When a block is selected, the correct block is placed when the tester clicks on any location on the map.

Test name: FR-LE-3
Results: The selected block is placed on the map when the tester clicks the location on the map.

Test name: FR-LE-4

Results: The correct level is loaded when the tester selects a JSON level file, and all level files are displayed in the game menu.

Test name: FR-LE-5
Results: Mario is unable to die while in the level editor. Enemies do not kill him and he is unable to fall off the map.

Test name: FR-LE-6
Results: When multiple enemies are placed on top of each other, they collide and spread when they fall.

Test name: FR-LE-7
Results: During gameplay, when any unmapped or unintentional key is pressed no exception is generated and nothing unexpected happened.

2 Nonfunctional Requirements Evaluation

2.1 Usability

Test name: SS-1
Results: Each user that we observed performing the test was able to successfully run the program without assistance within 2 minutes. The game is aimed at users who are familiar with programming concepts and have experience with using the Python programming language.

Test name: SS-2
Results: Once the game is launched, everyone from our test group was able to successfully navigate the menu screen and select a level and start the level player.

Test name: SS-3
Results: All of the users are able to read the text on screen.

2.2 Performance

Test name: SS-4
Results: There were several different outcomes of this test. Though after testing, each tester was able to successfully navigate the menu screen. At first the testers did not know which keys caused which actions so they were all pressing different keys in the beginning. However, once they found out which keys did what by trial and error, they were able to navigate the menu successfully.

Test name: SS-5
Results: Similar to the last test, at first the testers did not know which buttons performed which action. Most of the testers were able to successfully identify the side to side movement keys right away. However the jump key is what they struggled with. Some testers found out that the "a" key was the jump key due to it's correlation with the menu, others took some time performing trial and error. In the end, our users were able to find the correct movement keys within two minutes.

Test name: SS-6
Results: The game is able to run without crashing or raising an exception at least 2 hours after it is started with no input from the user. Longer time periods were out of the testing scope.

Test name: SS-7
Results: During normal testing and use case this test passes without any dip in performance. However, when there are many other programs running or if the CPU is under heavy load, then due to the nature of the Python environment, the game can experience a very large slow down based on the performance of the CPU and the other programs being run simultaneously. Since this is outside of the scope of the test, we do not eed to consider it a test fail.

3 Comparison to Existing Implementation

There are several distinct differences between our modified project, Mario Level X, and the original Mario Level 1 implementation. Throughout our

development process, significant refactors have taken place which differentiate our system design from the original. Our new system is more modular, as a complete overhaul of the map-saving, reading, and writing took place. A rework of the collision classes was also incorporated to use inheritance which did not take place in the existing implementation. We have also added several features, including map editing, and map saving to files, such that the user can select any of the saved maps on the system. All these changes combine to create a new game that contains more features, and has a more modular and better designed code-base as its core. These changes allow for new map components (enemies, collidables) to be easily added in the future with little modifications.

4 Unit Testing

Only modified components classes were tested using the "component_test.py". Since most methods require the game to be running, the unit tests only tests for the data validity of each object. Tests consists of constructing a component class with predefined parameters and testing if the state variables correspond with the given input. This ensures that the game will render these components with the values passed into the constructors. The "serialize" method were also tested for each component as we can assume that the values of the serialized object should match the values provided with the input. This way we can ensure that each component will convert into JSON with fields that correspond to our inputs, allowing us to save the states of these components

5 Changes Due to Testing

During the testing phase, we discovered some issues with our initial version of the code.

5.1 Menu Evaluation

There have been no changes to this category as all requirements were fulfilled in the testing phase.

5.2 Level Player Evaluation

In the initial stages of testing, we found few minor problems with the Level Player gameplay. Since the game's collider objects were chosen based on the map background and not discrete entities, there is no direct relationship between the image and collider. This caused some "invisible wall" effects where the collider did not match with the image. This caused requirement FR-LP-2 to fail. Secondly, the camera was behaving in a different way than what we had expected, moving forward on the map when Mario was moving back and forth or when he was stationar causing FR-LP-3 to fail. We implemented changes to fix this after testing. After the implemented changes, requirements FR-LP-2 and FR-LP-3 passed producing an overall output of 5 out of 5 tests passing.

5.3 Level Editor Evaluation

During the testing phase of the Level Editor, we discovered several tests to fail. The first failed requirement was FR-LE-3. The problem with this requirement was if a block was placed in the same location multiple times, there will be multiple entries in the JSON file. The second requirement to fail was FR-LE-6. This requirement failed due to the enemies stacking on top of each other when the same location is clicked. The problem was solved by the same change for both of these requirements. After the changes, all 7 of 7 tests were passing.

5.4 Usability Evaluation

The third category, usability, did not cause any of the participants to fail any of our tests, showing that the program is very usable and user friendly. No changes were made after the testing phase.

5.5 Performance Evaluation

The last category, performance evaluation did not cause any significant problems. The one special case which is not covered by our tests, but is also outside of a normal use case is if the game is being run while the CPU is also experiencing heavy load. When the game is run during this state, there can be significant slow downs based on the performance of the CPU. There were

no changes made due to the nature of the Python environment, having it's performance closely correlated to the performance/loading of the CPU.

6 Automated Testing

Initially, we had planned to incorporate some level of automated testing into our test plan. But due to the nature of our tests we realized that this would not only be difficult to accomplish but also not necessary. This game requires heavy user interaction with key presses and visual queues from the game view, it would be very difficult to develop an automated testing platform. Due to this, we focused many of our test cases on the fact that a user would be able to visually percieve any exceptions or crashes that may occur, it would be the simplest and most time saving method to perform all of the tests manually, recording the results of each test in a file. Though there were no automated tests, we were still able to carry out all of our white box and black box tests effectively, without impeding our ability do continually develop the code to account for any changes.

7 Trace to Requirements

This section shows the Traceability Matrix between the test cases and the requirements to which the tests apply.

Test Case	**Req.**
FR-M-1	FR2
FR-M-2	FR2, FR3
FR-M-3	FR1
FR-M-4	NF4, NF7
FR-M-5	NF9
FR-LP-1	FR2, NF8, NF10
FR-LP-2	FR2, NF7
FR-LP-3	FR2, NF10
FR-LP-4	NF7, NF8, NF10
FR-LP-5	NF9, NF10
FR-LE-1	FR1, FR3, FR4, FR5, FR6, NF1, NF3, NF7
FR-LE-2	FR1
FR-LE-3	FR1
FR-LE-4	FR1, FR3, FR4, NF7
FR-LE-5	FR1, FR2
FR-LE-6	FR1
FR-LE-7	NF2, NF3, NF7
SS-1	NF4, NF5
SS-2	NF4, NF5
SS-3	NF6
SS-4	NF4
SS-5	NF8
SS-6	NF9
SS-7	NF8, NF10
PoC-S-1	FR2, FR4
PoC-S-2	FR3
PoC-S-3	NFR9
PoC-S-4	NFR9
PoC-S-5	NFR9
PoC-M-1	FR2, NF8
PoC-M-2	FR2, NF8
PoC-ST-1	NF2

Table 2: Trace Between Test Cases and Requirements

8 Trace to Modules

This section shows the Traceability Matrix between the test cases and the modules of the system to which the tests apply. Please refer to the Module Guide for the breakdowns of the modules included below.

Test Case	**Req.**
FR-M-1	M2, M2
FR-M-2	M1, M2, M3
FR-M-3	M2, M3
FR-M-4	M2
FR-M-5	M1, M2, M3
FR-LP-1	M2, M3
FR-LP-2	M2, M3
FR-LP-3	M2, M3
FR-LP-4	M2, M3
FR-LP-5	M1, M2, M3
FR-LE-1	M1, M2, M3
FR-LE-2	M2, M3
FR-LE-3	M2, M3
FR-LE-4	M1, M2, M3
FR-LE-5	M1, M2, M3
FR-LE-6	M2, M3
FR-LE-7	M1, M2, M3
SS-1	M1, M2
SS-2	M1, M2
SS-3	M2
SS-4	M2
SS-5	M1
SS-6	M1, M2, M3
SS-7	M2, M3
PoC-S-1	M2, M3
PoC-S-2	M1
PoC-S-3	M1, M2, M3
PoC-S-4	M1, M2, M3
PoC-S-5	M1, M2, M3
PoC-M-1	M2, M3
PoC-M-2	M2, M3

Table 3: Trace Between Test Cases and Requirements

9 Code Coverage Metrics

Through our unit testing, found in component_tests.py, we obtained 100% code coverage within our system. These results were found by thoroughly testing our serialize methods that are the foundation for map-creation, as well as testing the individual components that are added to each map. As our program does not use branching methods, it is easy to ensure coverage by simply testing each methods individually, and checking to see whether or not it satisfies the correct and expected result. We can also ensure through the traceability matrix found above, that all modules are sufficiently tested.