SE 3XA3: Test Report
Sketchy Super Mario Bros.

Team #1, Wario's Miners
Kristine Uchendu (uchenduc)
Jason Nam (namy2)
Rylan Sykes (sykesr)

April 12, 2022

Contents

1 Functional Requirements Evaluation **1**
- 1.1 Revision 0 Functional Tests . . . 1
- 1.2 Revision 1 Functional Tests . . . 2

2 Nonfunctional Requirements Evaluation **2**
- 2.1 Usability . . . 2
- 2.2 Performance . . . 3
- 2.3 Operational and Environmental . . . 3
- 2.4 Maintainability and Support . . . 4
- 2.5 Health and Safety . . . 4

3 Comparison to Existing Implementation **4**

4 Unit Testing **5**

5 Changes Due to Testing **5**

6 Automated Testing **5**

7 Trace to Requirements **5**
- 7.1 Functional Requirements . . . 5
- 7.2 Non-Functional Requirements . . . 6

8 Trace to Modules **7**

9 Code Coverage Metrics **7**

List of Tables

1 **Revision History** . . . 1

List of Figures

This document outlines the results of our testing plan that we described in our TestPlan.pdf document. Additionally, this document covers the effects that testing had on our application, as well as the approaches we took for automated testing and code coverage.

1 Functional Requirements Evaluation

1.1 Revision 0 Functional Tests

1. Test ID: **T1** (Dynamic, Manual). Result: Success
2. Test ID: **T2** (Functional, Manual). Result: Success
3. Test ID: **T3** (Dynamic, Manual). Result: Success
4. Test ID: **T4** (Dynamic, Manual). Result: Success
5. Test ID: **T5** (Dynamic, Manual). Result: Success
6. Test ID: **T6** (Dynamic, Manual). Result: Success
7. Test ID: **T7** (Dynamic, Manual). Result: Success
8. Test ID: **T8** (Dynamic, Manual). Result: Success
9. Test ID: **T9** (Dynamic, Manual). Result: Success
10. Test ID: **T10** (Dynamic, Manual). Result: Success
11. Test ID: **T11** (Dynamic, Manual). Result: Success
12. ~~Test ID: **T12** (Dynamic, Manual). Result: Success~~
13. Test ID: **T13** (Dynamic, Manual). Result: Success

Table 1: **Revision History**

Date	Version	Notes
April 12, 2022	0.0	Revision 0

1.2 Revision 1 Functional Tests

1. Test ID: **T28** (Dynamic, Manual). Result: Success
2. Test ID: **T29** (Dynamic, Manual). Result: Success

2 Nonfunctional Requirements Evaluation

2.1 Usability

NFR-U-1 (T16)
Type: Manual
Result: Success

The following manual test was performed by introducing the game to 10 new users. The 10 new users were subjected to play the game without prior knowledge or know-how (knowledge and know-how of the original Super Mario Bros 16 bit is non-trivial). All found it easy to access stage, and were able to complete the first stage in less than 5 tries.

NFR-U-2 (T17)
Type: Manual
Result: Success

The following manual test was performed by surveying different people from different age groups. The age groups were as follows: 10-20, 20-30, 30-40. The people were surveyed on all the texts within the game that had been documented. All surveyed understood the simple English vocabulary used in the game.

NFR-U-3 (T18)
Type: Manual
Result: Success

The following manual test was performed by surveying people. The people were surveyed on all the symbols that ware used in the game. They were asked if the symbols are easy to understand and is universal. All surveyed understood the simple symbols used in the game, and most deemed the symbols universal.

2.2 Performance

NFR-P-1 (T19)
Type: Manual
Result: Success

The following manual test was performed by cloning the game repository on different types of devices, and running them to observe that the system runs without failure. All the devices were able to run the game without problems, and the game ran smoothly.

NFR-P-2 (T20)
Type: Manual
Result: Mostly Success

The following manual test was performed by manually testing numerous key bind inputs, and recording the time of the response of each actions. All the responses were less than 1 seconds, which satisfied our requirement. Some inputs had lag. However, this happened to be a lack of optimization for some operating systems.

2.3 Operational and Environmental

NFR-OE-1 (T21)
Type: Manual
Result: Success

The following manual test was performed by manually executing the game on Linux and macOS operating system devices. All devices running on Linux or macOS operating systems were able to run the game perfectly.

NFR-OE-2 (T22)
Type: Manual
Result: Success

The following manual test was performed by manually executing the game on Windows operating system devices. All devices running on Windows operating systems were able to run the game perfectly.

NFR-OE-3 (T23)
Type: Manual
Result: Success

The following manual test was performed by running the game in a Java 15 environment. The devices used to test this had Java 15 installed. All devices were able to run the game perfectly in a Java 15 environment.

2.4 Maintainability and Support

NFR-MS-1 (T24)
Type: Manual
Result: Success

The following manual test was performed by manually running a Gradle build from the source code files. The Gradle build was successful, and there were no lines of code that generated an error.

2.5 Health and Safety

NFR-HS-1 (T27)
Type: Manual
Result: Success

The following manual test was performed by having 10 users to play through all the game components while looking at the screen. All users reviewed the system graphics and determined that the feasibility of the system causing epileptic seizures based on medical consensus was close to none.

3 Comparison to Existing Implementation

The original project did not have any tests implemented. Given that the project was unable to run during initial attempts, it would not have passed any of our test cases. Our project differed as we employed various manual tests (functional, non-functional) to verify functionality. We also created

modules in our code that allows for better traceability throughout our tests and requirements.

4 Unit Testing

Due to the nature of the software system, unit testing was not able to really be achieved in a meaningful way throughout this project. In a game such as Super Mario Brothers, the units of code work very closely with one another so one unit may need to employ another unit in order to be tested, not because the system isn't modular but because testing one unit may require the testing of another unit and that other unit may not make sense to test manually or automatically. For example, one code unit may deal with the physics of a character jumping, which could be tested manually but this would require the tester to see the character moving which would mean running many other units of the code as well. The physics code unit could also be tested automatically with a testing framework like JUnit, but that would require us to have almost exact expected conditions for the system to compare with to evaluate whether the test failed or not. Another part of our project that made unit difficult was the fact that automated testing was also quite difficult (and as we eventually discovered infeasible), and this is further discussed in the **Automated Testing** section.

5 Changes Due to Testing

The results of manual testing were comprised of mainly minor bug fixes and UI/graphical changes. There were no major overhauls or design changes as a result of our testing. Once the unintended behaviours were fixed, all tests were passed.

6 Automated Testing

As alluded to in the **Unit Testing** section of this document, automated testing was deemed infeasible for this project. Initially we had expected that automated testing would not be the highest priority type of testing but that we would still complete some to test the functionality of some important classes. We realized midway through the semester that trying to feed specific

input conditions into JUnit and predict a specific end state for the system to compare with, was quite tedious when the game could just be run and the functionality of the game could be observed. Ultimately, the priority in a game is to achieve the desired functionality, and for that to translate onto the screen in the intended way so that the user is able to play the game. That is why automated tests are omitted from the test plan as well as this document. While we are confident in this approach for the scope of this project and course, this does impact our ability to quantitatively assess our code coverage when testing, which will be discussed in the **Code Coverage Metrics** section.

7 Trace to Requirements

7.1 Functional Requirements

FR1: T1 T2

~~**FR2:**~~

FR3: T4

FR4: T5

FR5: T6

FR6: T7

FR7: T8, T9

FR8: T9

FR9: T8

FR10: T9

~~**FR11:**~~

~~**FR12:**~~

FR13: T10

FR14: T11

FR15: T13

FR16: T3

FR17: T28

FR18: T29

7.2 Non-Functional Requirements

NF4: T16

NF5: T17

NF6: T18

NF8: T19

NF10: T20

NF11: T21

NF12: T22

NF13: T23

NF17: T27

NF18: T24

8 Trace to Modules

M1 Gradle Configuration Module: T1, T2

M2 Launcher Module: T1, T2

M3 Audio Module: T2, T3

M4 Actions Module: T4, T5, T6, T7, T8, T9, T10, T11, T12, T13

M5 Input Module: T4, T5, T7, T9, T11, T13

M6 Models Module: T2, T3, T6, T7, T8, T11, T12, T28, T29

9 Code Coverage Metrics

As previously mentioned in this document, we focused on manual testing and with that comes the challenge of ensuring code coverage when testing. Due to the limited time we had to work on this project we didn't have as much time as we had hoped to look into a more sophisticated way of assessing how much of the code our manual tests were covering. Something we would often do is we would put print statements in specific methods to ensure that as we played the game, the methods that we were expecting to execute would execute, and that gave us an idea of how much of the code was actually being reach by our tests. Something we would have liked to try had we had more time is trying a software that records code coverage during manual tests. One software that was found in our preliminary research was Teamscale, but more specifically the Test Gap analysis feature. This feature is able to record test coverage regardless of whether the test is automated or manual, and has detailed documentation as well as tutorials. Trying this Teamscale feature out, would definitely be a learning curve, but would also be a key next step for the group if given more time.