SE 3XA3: Test Plan
Sketchy Super Mario Bros.

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

April 12, 2022

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
March 11, 2022	1.0	Revision 0
April 11, 2022	2.0	Revision 1

1 General Information

1.1 Purpose

The purpose of this documentation how the group plans on validating and verifying the Sketchy Super Marios Bros game. The requirements that were outlined in the Software Requirements Specification documentation are ultimately going to be the basis of the how we approach the verification and validation of the game. The majority of the document is going to detail the test that will be performed to ensure functionality as well as validation of the implementation.

1.2 Scope

Overall scope of this documentation is to outline the testing approach for this project as well as to explain the tests that will be performed. Outlining the testing approach involves specifying the types of tests that will be performed and the reasoning behind those choices. How exactly is our testing approach going to verify the requirements put forth in our SRS documentation? In terms of the explanations of the tests that will be performed, this will entail an explanation how we will execute our tests, but also how we plan to conduct automated tests, and the test case that we plan on using for our proof of concept. Lastly, administrative details related to testing will also be outlined in this document. these details include our testing schedule as well as how we will be assessing code coverage in our tests. It should be noted that the original implementation of the project did not contain any test cases so our test cases will be covering the main functionalities of the game (e.g. main controls, changes in state, etc.) as well as our new features.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
SRS	Software Requirement Specification
POC	Proof of Concept
FR	Functional Requirement
NFR	Non-functional Requirement
GUI	Graphical User Interface
UI	User Interface
IF	Internal Functions

Table 3: **Table of Definitions**

Term	**Definition**
Sketchy Super Mario Bros	This is the name of our project, which is a recreation of the 16-bit version of Super Mario Bros.
JUnit	JUnit is a unit testing framework for the Java programming language
Java	Java is a high-level, class-based, object-oriented programming language

1.4 Overview of Document

The remainder of this document will be outlining exactly how we plan on testing the requirements mentioned in the SRS for the Sketchy Super Mario Bros project. Initially is the plan, and then after the plan is the test case description. Each test case will highlight what type of test it is, the initial state, the input/condition, the output, and finally how that test will be performed.

2 Plan

2.1 Software Description

Sketchy Super Mario Bros. is a software application in which the original 16-bit Super Mario Bros. game is implemented as a desktop application. The original Super Mario

Bros game was not designed with the modern laptop in mind, but rather with the consoles that Nintendo sold during its release in mind, so the software provides fans of the game with an opportunity to play on their own computers. As well, the repository from which the original game was taken never fully implemented all of the features and functionalities of the game and the group's approach is to try to restore as many of those features and functionalities as possible given the remaining time available in the course.

2.2 Test Team

The testing team is comprised of all three current members of the development team: Kristine Uchendu, Jason Nam, and Rylan Sykes. Because the group is comprised of three people. We intend on dividing the burden of testing, and who works on what will largely be determined by interest as well as what feature in game they are implementing.

2.3 Automated Testing Approach

~~Part of the testing strategy for Sketchy Super Mario Bros is going to be the use of automatic testing to test specific features of the game (e.g. smaller features like mushrooms, as well as larger features like levels). JUnit is designed for creating automated tests in Java based projects and thus will be the ideal framework for us to implement automated tests.~~ Since the project is a game, many of our tests will need to be manual to ensure that beyond the code doing what was intended, the game itself is working in the way we intended it to. We will use various types of static and manual testing as opposed to automated testing because our testing will focus on improving the gameplay experience.

2.4 Testing Tools

~~The system is almost entirely built with Java, therefore the primary tool that we will be using for testing is JUnit.~~ The game will be run on the most recent version of IntelliJ, and running the game done when manual verification and UI testing are necessary to validate the system's higher level functionality.

2.5 Testing Schedule

See Gantt Chart at the following URL: Gantt Chart

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Testing Core Gameplay Functionality

1. T1

   Type: Dynamic, Manual

   Initial State: N/A

   Input: The user opens the game

   Output: The user is presented with the main menu GUI

   How test will be performed: The game will be started using the IDE (IntelliJ), being launched using the Gradle run command for the desktop configuration.

2. T2

   Type: Functional, Manual

   Initial State: N/A

   Input: The user opens the game

   Output: All graphic and music files should be loaded and processed with libgdx without error

   How test will be performed: The game will be started using the IDE (IntelliJ) and all menu & in-game graphics will be loaded. Additionally, background music and sound effects should be played correctly. **Any exceptions from missing or corrupted files should be handled**.

3. T3

   Type: Dynamic, Manual

   Initial State: The user is at the main menu GUI

   Input: The user clicks the GUI option to start the game

   Output: Main menu is disposed and the initial game state is loaded and rendered

   How test will be performed: The start game menu option will be clicked after intial start

4. T4

   Type: Dynamic, Manual

   Initial State: The user has entered the game and is either grounded or moving left or right

   Input: The user directs Mario left or right using the input keys

   Output: Mario will react by moving in the direction of the specified input

   How test will be performed: Mario should react regardless of state: if Mario is grounded the velocity will be directed in the input's direction. If Mario is already moving (on ground) then Mario's velocity should either be maintained or reverse directions depending on which input in provided.

5. T5

   Type: Dynamic, Manual

   Initial State: Mario is grounded/moving or in the air

   Input: The user directs Mario upwards (jump)

   Output: Mario will react by moving jumping while maintaing the horizontal velocity

   How test will be performed: Mario should react based on his current velocity: if Mario is grounded he will jump with no horizontal velocity. If Mario is moving horizontally he will maintain this velocity when jumping. Regardless, Mario should gain vertical position.

6. T6

   Type: Dynamic, Manual

   Initial State: Game is started and Mario is not on the flagpole

   Input: Actors are moving or grounded

   Output: Mario, and actors, should be subject to normal physics based on his environment

   How test will be performed: Mario should be grounded on a surface, fall when there is no surface, and interact with any physical objects in his environment during movement. The same rules apply for actors that move.

7. T7

   Type: Dynamic, Manual

   Initial State: Mario steps on a Goomba's head (stomp)

   Input: User lands on a Goomba

   Output: Goomba is eliminated and Mario gains small jump boost

   How test will be performed: Mario will jump (or fall off a surface) to gain height that is greater than the Goomba's height, then land directly on the Goomba.

8. T8

   Type: Dynamic, Manual

   Initial State: Mario is full health and not on the flagpole or has recently been attacked by a Goomba

   Input: User interacts with a Goomba (non-stomp)

   Output: Mario will take 1 damage hit

   How test will be performed: Mario's damage will be tested with 2 states: small Mario and big Mario. Mario should die if taken a hit while small, and Mario should reduce his health by 1 (and return to small Mario) if currently big.

9. T9

   Type: Dynamic, Manual

   Initial State: Mario is not on the flagpole

   Input: User will fall off the bounds of the map

   Output: Mario will die and a game over screen will appear

   How test will be performed: Mario will fall within a hole or edge of map. It will then be ensured that Mario is eliminated and the game over screen appears.

10. T10

    Type: Dynamic, Manual

    Initial State: Mario is alive and not on the flagpole

    Input: User makes Mario interact with powerup

Output: Powerup ability should apply to Mario

How test will be performed: Powerup should apply to Mario regardless of what type of interaction occurs. If Mario already has a powerup active, there should be no interaction.

11. T11

    Type: Dynamic, Manual

    Initial State: Mario is alive and not on the flagpole

    Input: User jumps below a brick, causing Mario's head to touch the brick

    Output: If Mario is small, the brick should release a powerup (if applicable). If Mario is big, the brick should break.

    How test will be performed: Both big and small Mario will be tested. The user should position themselves such that Mario's head will interact with the brick at some point during the jump. The user should jump and the correct interaction should happen.

12. ~~T12~~

    ~~Type: Dynamic, Manual~~

    ~~Initial State: Mario is alive and not on the flagpole~~

    ~~Input: User moves Mario to interact with a coin~~

    ~~Output: The coin should disappear and 1 point should be added to Mario's total points for the level~~

    ~~How test will be performed: Mario will be moved into a coin (by any direction) through the inputs and, by tracking the score, ensure that the level points increase by 1 for each coin attained.~~

13. T13

    Type: Dynamic, Manual

    Initial State: Mario is alive

    Input: User moves Mario to interact with the flagpole

    Output: Mario should attach to the flagpole and slide down to the bottom

How test will be performed: The end of the level will be reached by fully playing through the level. The flagpole will be interacted with by jumping from the platform, or walking directly into the pole. After such, Mario should slide down to the bottom of the pole, and then a success/congratulations game over screen should be displayed.

14. T28

    Type: Dynamic, Manual

    Initial State: Mario is alive

    Input: User moves Mario upwards into a brick that has a hidden coin

    Output: Mario's coin counter should go up by 1 and a coin sound effect should play

    How test will be performed: Mario will be positioned beneath a brick that is known to have a coin. Mario will then jump and the changes will be observed.

15. T29

    Type: Dynamic, Manual

    Initial State: Mario is alive

    Input: User directs Mario to interact with actor that triggers an increase in points

    Output: Mario's point counter should increase by the actor's respective point worth and a points sound effect should play

    How test will be performed: Mario will be positioned to interact with the actor such that the points should be given to Mario, the changes will be observed.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel

1. T14

   Type: Dynamic, Manual

   Initial State: The system is cloned in the device of tester.

   Input/Condition: The tester will execute the system.

Output/Result: The GUI is displaying in the screen.

How test will be performed: The test will run the system. The tester will manually check to see if the GUI displays correctly.

2. T15

   Type: Dynamic, Manual

   Initial State: The system is cloned in the device of tester.

   Input/Condition: The tester will execute the system.

   Output/Result: The tester will then determine if the GUI allows tester to select stage and set preferences within the options windows.

   How test will be performed: The test will run the system. The tester will determine if it is possible to access different functionalities within the game from the GUI.

3.2.2 Usability and Humanity

1. T16

   Type: Dynamic, Manual

   Initial State: The system is running on the device of tester.

   Input/Condition: The tester who is new to the system will test the system and interact with system and game stages..

   Output/Result: The new tester will be able to successfully operate and interact with the system. The new tester will also be able to successfully play the first stage in the game.

   How test will be performed: A new user will interact with the system. The new user will be able to traverse the GUI, and pass the first stage of the game in less than 5 runs.

2. T17

   Type: Manual, Static

   Initial State: All texts within the game will be documented in a manual.

   Input: The tester reads said document.

   Output: The document contains simple vocabulary of English language that is easy to read and understand.

   How test will be performed: The tester will determine that all text within the game are easy to read and understand. The tester will determine that a user with simple understand of the English language will be able to interact with the system with little impairment.

3. T18

   Type: Manual, Static

   Initial State: All symbols within the game will be documented in a manual.

   Input: The tester reads said document.

   Output: The document contains universal symbols that everyone can understand.

   How test will be performed: The tester will determine that all symbols within the game are easy to understand. The tester will determine that new users will not need to learn new symbols and will be able to subconsciously identify symbols used in the system.

3.2.3 Performance

1. T19

   Type: Dynamic, Manual

   Initial State: The repository is cloned on the tester's device.

   Input: The tester will run the system. The tester will run all components of the system.

   Output: The system will run successfully for all system components. All test executions will result in stable graphic output.

   How test will be performed: The system will be run on the device with medium spec device. The tester will determine that the system runs without failures, and that all system components run smoothly.

2. T20

   Type: Dynamic, Manual

   Initial State: The system is running on tester's device.

   Input: The tester will input executable commands.

   Output: The system will respond within TIME_RESPONSE with the requested action and appropriate response.

   How test will be performed: The system will be tested by inserting input commands and measuring the response time for the requested action sequence to execute. The tester will determine if the response time is over TIME_RESPONSE, which would equate to test failure.

3.2.4 Operational and Environmental

1. T21

   Type: Dynamic, Manual

   Initial State: The repository is cloned on a Linux and macOS operating system device.

   Input/Condition: The system is run on the device of the tester.

   Output/Result: The system successfully runs on the most recent version of Linux OS and macOS.

   How test will be performed: The test will determine that the system is functional on Linux operating systems and macOS. The test can also be performed on a virtual machine running on the Linux and mac operating system.

2. T22

   Type: Dynamic, Manual

   Initial State: The repository is cloned on a Windows operating system device.

   Input: The system is run on the device of the tester.

   Output: The system successfully runs on the most recent version of Windows OS.

   How test will be performed: The test will determine that the system is functional on Windows operating systems. The test can also be performed on a virtual machine running on the Windows operating system.

3. T23

   Type: Dynamic, Manual

   Initial State: The repository is cloned on tester's device.

   Input: The tester runs the system in a Java 15 environment.

   Output: The tester will check to ensure all system functions work on the version 15 of Java. The system successfully runs on the device of the tester.

   How test will be performed: The test will determine that the system is executable and functional on Java 15.

3.2.5 Maintainability and Support

1. T24

   Type: Dynamic, Manual

Initial State: The repository is cloned on the tester's device.

Input/Condition: The tester runs the Gradle build from the source file.

Output/Result: The tester will count lines of code that expresses error, if there is one.

How test will be performed: The source code will be tested to check if there are errors or faults in the source code. The tester will count, document, and monitor the lines of code that needs modification.

3.2.6 Security

1. T25

   Type: Dynamic, Static, Manual

   Initial State: N/A

   Input/Condition: The tester reads through source code.

   Output/Result: The tester determine if the system has collected user data without permission.

   How test will be performed: The tester will have thorough knowledge of coding practices. The tester will then evaluate the code to find any source code that collects user data without authorization. The tester will make sure no output files are created that contains any user data.

3.2.7 Cultural

1. T26

   Type: Dynamic, Manual

   Initial State: All imagery and text within the game are documented in manual.

   Input/Condition: The tester reads said document.

   Output/Result: The tester determines whether the imagery and text outlined in the documents are considered hostile or offensive.

   How test will be performed: The tester will possess thorough knowledge of cultural and religious differences that can consider some imagery or text to be offensive. The content from the manual will evaluated based on said knowledge.

3.2.8 Health and Safety

1. T27

   Type: Dynamic, Manual

   Initial State: The tester has started the game.

   Input/Condition: The tester will play through the game.

   Output/Result: The tester will determine if epileptic seizures from the game's graphics is plausible.

   How test will be performed: The tester will play through all of the game components to review all the system graphics and determine the feasibility of the system causing epileptic seizures based on medical consensus.

3.3 Traceability Between Test Cases and Requirements

Test Case ID	**Test Case Description**	**Test Type**	**Req. ID(s)**	**Requirement Description**
T1	The system must present the user with the main menu on start	Dynamic, Manual	FR1	The user must be able to start a new game
T2	The system must load all graphics and sounds on start	Functional, Manual	FR1	The user must be able to start a new game
T3	The main menu must allow the user to enter the game	Dynamic, Manual	FR1	The user must be able to start a new game
T4	Mario must be able to move horitonztally	Dynamic, Manual	FR3	The user must be able to move Mario to the right and left
T5	Mario must be able to jump	Dynamic, Manual	FR4	The user must be able to move Mario up (Jump)
T6	All moving actors must be subject to the game's physics	Dynamic, Manual	FR5	Mario, enemies, and power-ups in the game must be subject to gravity
T7	Mario must be able to stomp Goomba's	Dynamic, Manual	FR6, FR12	Mario must be able to attack enemies

T8	Mario must be able to take damage	Dynamic, Manual	FR7, FR8, FR9, FR11	Mario must be able to take damage from the environment and enemies, die, as well as an animation should play after receiving damage
T9	Mario must die when fallen off of map bounds	Dynamic, Manual	FR5, FR8	Mario must be subject to gravity as well as receive damage
T10	Mario must be able to consume power-ups	Dynamic, Manual	FR13	Mario must be able to consume power-ups
T11	Mario must be able to interact with bricks	Dynamic, Manual	FR14	Mario must be able to break bricks
~~T12~~	~~Mario must be able to collect coins~~	~~Dynamic, Manual~~	~~N/A~~	~~N/A~~
T13	Mario must be able to finish the game by interacting with the flagpole	Dynamic, Manual	FR15	Mario must be able to finish the level
T14	GUI should display	Dynamic, Manual	NF1, NF2	The system shall look similar to the original Super Mario Bros 16 bit game
T15	User should easily interact with the menu GUI	Dynamic, Manual	NF1, NF6, NF7, NF4, NF5	System is easily useable assuming the user know basic English
T16	User should be able to play the first stage in the game	Dynamic, Manual	NF4	System shall be used by users with no knowledge about the system mechanics
T17	All texts within the game will be documented in a manual	Manual, Static	NF5, NF6, NF7	System shall use symbols and terms that are easy to understand
T18	All symbols within the game will be documented in a manual	Manual, Static	NF6	System shall use symbols and terms that are easy to understand

T19	The system will have a stable graphic output	Dynamic, Manual	NF8, NF9, NF10, NF11, NF12, NF13	System shall be readily available on a variety of OS
T20	System responds within a certain time respond threshold	Dynamic, Manual	NF9, NF10	System is available and responds aptly
T21	System runs on the most recent version of Linux	Dynamic, Manual	NF9, NF11	System is available and runs on Linux
T22	System runs on the most recent version of Windows	Dynamic, Manual	NF9, NF12	System is available and runs on Windows
T23	System runs in Java 15 environment	Dynamic, Manual	NF13	System shall be programmed in a language that will operate on future operating systems
T24	System runs Gradle build	Dynamic, Manual	NF9	System shall be available
T25	System does not collect user data without consent	Dynamic, Manual	NF15	System shall not collect or upload user data without authorization
T26	Tester confirms that all imagery and text do not appear to be offensive	Dynamic, Manual	NF16	System shall not contain any imagery or text that could be regarded as hostile or offensive
T27	No epileptic seizures from game's graphics are plausible	Dynamic, Manual	NF17	System shall avoid causing epileptic seizures to users
T28	Mario should gain a coin after jumping into a brick (from below) with a hidden coin	Dynamic, Manual	FR#?	FR#? description
T29	Mario should gain points after interacting into an actor that grants points	Dynamic, Manual	FR#?	FR#? description

4 Tests for Proof of Concept

4.1 Graphic and Sound Tests

1. POCT1

   Type: Functional, Manual

   Initial State: N/A

   Input: The user opens the game

   Output: All graphic and music files should be loaded and processed with libgdx without error

   How test will be performed: The game will be started using the IDE (IntelliJ) and all menu & in-game graphics will be loaded. Additionally, background music and sound effects should be played correctly.

   Note: Upgraded graphics and music/sound effects were the main features of our proof of concept demo.

4.2 Refactoring Tests

1. POCT2

   Type: Dynamic, Manual

   Initial State: N/A

   Input: The user performs all basic actions of the game

   Output: All actions should be performed as expected with no errors

   How test will be performed: The game will be launched with the IDE (IntelliJ) and a quick playthrough of the level will be performed. Core functionality that is expected to be checked are: movement, death, attacks, powerups, brick interactions, and level completion. No errors or exceptions are expected.

5 Comparison to Existing Implementation

In the original project, the group did not find any documentation or files relating to how the system was tested, and based on the amount of bugs that were found when the game was first initially run, it almost appeared as though no tests were done (highly unlikely though). Some of the immediately apparent issues was sound during gameplay, as well as warped graphics barely resembling the game released by Nintendo. Due to this lack of documentation from the original creator of the project, we cannot really compare the testing approach outlined in this documentation to the testing approach used by the original creator. WIth regards to some of the new functionality we will be adding, and comparing that to the old implementation, a main menu, end game prompts to reset the game and coin boxes are being added, and the group intends to test these as well.

6 Unit Testing Plan

6.1 Unit testing of internal functions

~~The project will use the Junit testing framework to test out source code. We can test each of our Java classes individually and thoroughly. Junit allows us to write and run repeatable automated tests with varying test inputs. With Junit tests, we expect to achieve test coverage of up to 50 percent of source code.~~ Unit testing will not be utilized for the reasons explained in the Automated Testing Approach section.

6.2 Unit testing of output files

The system does not generate any output files.

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

RESPONSE_TIME: The time it takes for a pixel to change its colour. (5 ms)

7.2 Usability Survey Questions

User Experience Survey

The following survey should be filled out after playing the Sketchy Super Mario Bros for at least 20 minutes.

Time spent using software:

Please provide a ranking between 0 and 10 in each of the categories. Please include any extra comments on what could be done to improve the product.

Colors, Typography, Style	0 1 2 3 4 5 6 7 8 9 10 [0 = very inconsistent, 10 = very consistent]
Ease of Use:	0 1 2 3 4 5 6 7 8 9 10 [0 = very difficult and counter-intuitive, 10 = very easy and intuitive]
User Interface:	0 1 2 3 4 5 6 7 8 9 10 [0 = messy and distracting, 10 = clean and nonperturbing to the eye]
Accessibility:	0 1 2 3 4 5 6 7 8 9 10 [0 = inaccessible, 10 = easy to access]
Overall Enjoyment :	0 1 2 3 4 5 6 7 8 9 10 [0 = playing this game was an extremely unpleasant experience, 10 = playing this game was an extremely pleasant experience]
Notes:	