SE 3XA3: Test Plan
Mario Level X

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

February 29, 2020

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
Feb. 28, 2020	0.0	Initial revision of this document

1 General Information

1.1 Purpose

The purpose for this test plan is to ensure that the software for the application was implemented correctly and behaves as expected.

1.2 Scope

The test plan will be used as a guideline of how the software should be tested. This includes what sections of the code will be tested, the benchmarks for successful test results, as well as what testing tools will be used.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
POC	Proof of Concept
UI	User Interface
JSON	Javascript Object Notation

Table 3: **Table of Definitions**

Term	**Definition**
unittest	Standard test library for python
Map	The playable game area
Level	An instance of a map

1.4 Overview of Document

This document will consist of test cases based off the requirements derived from the requirements specification, test schedule, and unit test plans.

2 Plan

2.1 Software Description

Mario-Level-X is a re-implementation of Mario-Level-1 that separates the level data rendering from the game functionalities. By rendering the level data from an external source, we can provide a UI for players to edit and save their levels.

2.2 Test Team

The members responsible for testing are Edward Liu, Ahmed Gharib, and Connor Czarnuch.

2.3 Automated Testing Approach

To implement automated unit testing, we will write tests using the unittest python library for our modules, and create scripts that runs every test case. We will run tests after each major commit to ensure that the software remains correct.

2.4 Testing Tools

Testing tools we will use is the unittest standard library for python.

2.5 Testing Schedule

Member	Task	Date
Ahmed	Menu	03/02/2020
Connor	Level Editor	03/15/2020
Edward	Level Player	03/20/2020

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Menu

1. FR-M-1

   Type: Functional, Dynamic, Manual

   Initial State: Main Menu Screen

   Input: User presses enter on "1 player game" menu option

   Output: Game loads into the game-screen and is displayed on screen

   How test will be performed: The game will be run and the "1 player game" option is selected to ensure that it works properly (user loads into the game).

2. FR-M-2

   Type: Functional, Dynamic, Manual

   Initial State: Main Menu Screen

   Input: User presses enter on "Edit Levels" menu option

   Output: Game loads into the level select UI.

   How test will be performed: Tester will launch game, select "Edit Levels", and should be redirected to the level select.

3. FR-M-3

   Type: Functional, Dynamic, Manual

   Initial State: Main Menu Screen

   Input: User presses on the level to edit in the level select screen.

   Output: The selected level loads.

   How test will be performed: Tester will launch game, select "Edit Levels", and should be redirected to the level that is selected.

4. FR-M-4

   Type: Functional, Dynamic, Manual

   Initial State: Main Menu Screen

Input: Arrow keys from the keyboard.

Output: The arrow keys will change the item that is selected in the main menu.

How test will be performed: Tester will log into the main menu of the game and check to see if the arrow keys cause the cursor to change item on the main menu and in the level select.

3.1.2 Level Player

5. FR-LP-1

   Type: Functional, Dynamic, Manual

   Initial State: Level Player

   Input: Mario goes through the map that is loaded

   Output: Each block that is placed on in the JSON file shows up in the correct position on the map.

   How test will be performed: Player loads into any level and checks the map and the JSON file to check if they match.

6. FR-LP-2

   Type: Functional, Dynamic, Manual

   Initial State: Level Player

   Input: Mario collides with all blocks properly.

   Output: Mario is unable to pass through existing blocks.

   How test will be performed: The tester will play the game and check if the block boundaries cause Mario to stop.

7. FR-LP-3

   Type: Functional, Dynamic, Manual

   Initial State: Level Player

   Input: Mario is moving on the screen.

   Output: The camera for the level player, scrolls with Mario, and Mario cannot pass the middle of the screen until the end of the map. The camera cannot move backwards.

How test will be performed: Tester will load into any level, and play through the level, making sure the Mario does not move past the middle of the screen until the end of the level. The camera also must not move backwards.

3.1.3 Level Editor

8. FR-LE-1

   Type: Functional, Dynamic, Manual

   Initial State: Level Editor

   Input: User clicks on a location in the level editor.

   Output: The block is saved in the JSON file when the level editor is exited.

   How test will be performed: User will enter the level editor using any level, and will place a block or enemy in the map. They will then exit the level editor and check if the block is saved in the JSON file.

9. FR-LE-2

   Type: Functional, Dynamic, Manual

   Initial State: Level Editor

   Input: User clicks on a location in the level editor.

   Output: The block type that appears in the location that is clicked is cycled when that location is clicked again.

   How test will be performed: User will enter the level editor using any level, and will click on a location in the map to check if the block cycles properly.

10. FR-LE-3

    Type: Functional, Dynamic, Manual

    Initial State: Level Editor

    Input: Mouse cursor selects the location to place a new block.

    Output: The correct location for the block or enemy is placed.

How test will be performed: The tester will enter the level editor and check if the mouse cursor selects the correct location to place the block or enemy.

11. FR-LE-4

    Type: Functional, Dynamic, Manual

    Initial State: Level Editor

    Input: Loading the level editor.

    Output: The proper level is loaded from the file and the user is able to edit.

    How test will be performed: The tester will load the level from the level select and check if the correct level is loaded by comparing the JSON file to what is displayed.

3.2 Tests for Nonfunctional Requirements

3.2.1 Usability

1. SS-1

   Type: Structural, Static, Manual

   Initial State: the game is installed onto the system and have yet to be launched

   Input/Condition: user is asked to run the program without assistance

   Output/Result: most users are able to successfully launch the program without assistance within 2 minutes

   How test will be performed: a test group of students within the Software Engineering program will be asked to attempt to launch our program without prior instructions. The time taken to run our program is recorded and averaged among all participants of the survey. The majority of these students will have the game running within 2 minutes.

2. SS-2

   Type: Structural, Static, Manual

   Initial State: the game is already launched and running on the main menu screen

Input: user is asked to load into the game from the main menu screen

Output: most users are able to successfully traverse from the main menu into game play

How test will be performed: a test group of students within the Software Engineering program will be asked to attempt to get into the play-screen from the main menu. The time taken to run our program is recorded and averaged among all participants of the survey. The majority of these students will successfully leave the main menu into the game and playing within 2 minutes.

3. SS-3 Type: Structural, Static, Manual

   Initial State: the game is running on any screen that contains text (main menu, in-game, etc)

   Input: user is asked to read what's written on screen, to see if it is identifiable as Canadian English

   Output: most users are able to successfully read the text on screen

   How test will be performed: one of our group members will run the program and cycle through all screens to ensure that the program is being displayed in English.

3.2.2 Performance

1. SS-4

   Type: Structural, Static, Manual

   Initial State: the game is on the menu screen

   Input: user is asked to press on a button on the main menu

   Output: most users experience immediate feedback that their input has been processed

   How test will be performed: a test group of students within the Software Engineering program will be asked to attempt to press on a menu button. The majority of these students will successfully receive feedback that the game processed their input successfully.

2. SS-5

   Type: Structural, Static, Manual

Initial State: the game is running on the gameplay screen

Input: user is asked to move the player around with user inputs

Output: most users experience immediate feedback that their input has been processed, when they see the player has moved

How test will be performed: a test group of students within the Software Engineering program will be asked to attempt to move the player around. The majority of these students will successfully receive feedback that the game processed their input successfully.

3. SS-6

   Type: Structural, Static, Manual

   Initial State: the game is running on any screen (main menu, in-game, etc)

   Input: once the program has been launched, it is left idle for an extended period of time

   Output: the game will not crash and will continue to run, not closing unexpectedly

   How test will be performed: one of our group members will run several instances of the game on different machines, leaving them running and checking periodically to see if the game is still running. If the game has not crashed in 90 percent of tests within 3 hours of running, then it has passed the test.

3.3 Traceability Between Test Cases and Requirements

4 Tests for Proof of Concept

4.1 Area of Testing1

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

Input:

Output:

How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

4.2 Area of Testing2

...

5 Comparison to Existing Implementation

6 Unit Testing Plan

We will be using the python unittest library to do unit testing for this project.

6.1 Unit testing of internal functions

To test the internal functions of our project, a good place to focus most of the testing is map creation methods such that they are fully functional for when the program starts reading and writing from map files. This will include testing of the functions that create the ground, pipes, or any other block that appears in the game. These tests will ensure that the methods operate correctly, meaning that a pipe declared to spawn at coordinates (200, 200) correctly spawns at the expected location. All map functions will be tested in a similar manner to ensure accuracy. An easy way to implement these tests is to return the coordinate values of any components added to the map, and compare these to what was written in the code. The goal is to

achieve as close to 100 percent accuracy with our implementation, with the exception of any game-library restrictions that are beyond our control.

6.2 Unit testing of output files

The output files for our project will be the save files for when users create their own custom maps. In order for the unit testing of output files to begin, the unit testing of internal functions must already have occurred and passed, as our map creation relies on the correctness and accuracy of our internal functions. The testing of output files will be conducted in two steps. One portion of testing is to ensure that a map is successfully saved to a file when the user decides to save their map. The second portion is to test loading the map to see if it successfully loads. There will be a method that tests whether elements that appear on the map are written in the map file, to see if they match. If the element in the game has a corresponding section in the map file, it is confirmed that the reading and writing to file is working as intended. In order for the map test cases to pass, there must be a 100 percent success rate when comparing map elements to the map file outline.

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

7.2 Usability Survey Questions?

This is a section that would be appropriate for some teams.