SE 3XA3: Test Plan
Mari0

Team 9, Ninetendo
David Hobson - hobsondd
Jose Miguel Ballesteros - ballesjm
Jeff Pineda - pinedaj

December 8, 2016

Contents

List of Tables

Table 1: **Revision History**

Date	Version	Notes
2016-10-28	1.0	Created document
2016-10-29	1.1	Added General Information. Added Plan. Added System Test Description. Added Tests for Proof of Concept. Added Comparison to Existing Implementation. Added Unit Testing Plan. Added Appendix
2016-11-28	1.2	Updated Document Description. Updated Table of Definitions. Updated tests for functional requirements. Updated Object Collision Testing.
2016-12-06	1.3	Added Manual Testing Approach. Updated Automated Testing Approach. Updated Gantt chart to reflect testing. Updated Non-Functional Requirements. Updated Survey.

This document will show the different requirements will be testing and the rationale behind some of the tests that we choose. Furthermore, this document will show what will be demonstrating along with how the tests will be conducted.

1 General Information

1.1 Purpose

Testing is a critical part of developing software to both ensure the program meets the project's requirements and to address the errors of the product.

1.2 Scope

In regard to our project, the game Mari0, the tests implemented will need to be carefully chosen in order to ensure that the game is above all else, fun. The chosen tests must also give the stakeholders of the project the assurance and confidence that the game plays and functions accordingly to the requirements outlined in the Software Requirements Specification (SRS) document. Testing will be automated where possible, however it is a necessity to test for certain attributes of the game manually.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
A.I.	Artificial Intelligence
G.U.I. / U.I.	Graphical User Interface / User Interface
SMB	Super Mario Bros.
SRS	Software Requirements Specifications

1.4 Overview of Document

This document will describe our project of reimplementing Mari0, the test team and the tools and frameworks that they will be using, and a list

Table 3: Table of Definitions

Term	Definition
Mario	The character that the player portrays
Portals	Two connected portals that allow characters and projectiles to enter one and exit through the other, whilst maintaining physics properties such as velocity and acceleration
Goomba	Enemy character that is defeated after the player stomps on the top of its head
Lives	The amount of times the player can die before game over
Question Block	Blocks found that when hit give the player coins or power ups
Score	The amount of points a player has obtained by collecting coins or killing enemy characters

and description of the tests we will be using to ensure our game meets the requirements listed in the SRS.

2 Plan

2.1 Software Description

The core game mechanics of Mari0 involve the simple platforming mechanics of moving left or right, and jumping; in addition to portals which allow characters and objects to be teleported between two portals while maintaining their physical characteristics.

2.2 Test Team

All members of Ninetendo, David Hobson, Jose Miguel Ballesteros, and Jeff Pineda, are to actively create and implement tests and test cases, and are also responsible for recording each tests' results.

2.3 Automated Testing Approach

If a method being written can be tested automatically, the test will be concurrently written alongside it or shortly after the method's completion. In addition, in order to ensure that all code has been tested effectively, white box testing will be used.

2.4 Manual Testing Approach

Many of the manual tests will be done by players/beta testers to ensure that the game reacts accordingly. A black box testing approach will be used for beta testing when users play our game and will ensure that outputs make sense for the game.

2.5 Testing Tools

As mentioned in the technology section of our Development Plan document, we will be using the NUnit framework to conduct our unit tests. Manual testing will be done in the Unity play mode interface.

2.6 Testing Schedule

Table 4: **Testing Schedule**

Date	**What is being tested**
Wed Nov 2	Testing player input and character movement
Fri Nov 4	Testing character physics and character rules (death)
Wed Nov 9	Testing enemy A.I. and A.I. rules
Fri Nov 11	Testing portal mechanics
Mon Nov 14	Testing level design

Click here for the Gantt Chart.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Input Testing

1. Run Right (Starting from Idle)

   Type: Dynamic, Manual

   Initial State: In-game state where the character is not moving

Input: RIGHT_ARROW or D_KEY on keyboard are pressed

Output: The character will be moving towards the right at a constant speed

Tester(s): Development Team and Colleagues

Description: The tester will make sure that the character?s movement functions as expected

2. Run Left (Starting from Idle)

   Type: Dynamic, Manual

   Initial State: In-game state where the character is not moving

   Input: LEFT_ARROW or A_KEY on keyboard are pressed

   Output: The character will be moving towards the left at a constant speed

   Tester(s): Development Team and Colleagues

   Description: The tester will make sure that the character?s movement functions as expected

3. Run Right (Starting from moving left)

   Type: Dynamic, Manual

   Initial State: In-game state where the character is currently moving left at a constant speed

   Input: RIGHT_ARROW or D_KEY on keyboard are pressed

   Output: The character will be moving towards the right at a constant speed

   Tester(s): Development Team and Colleagues

   Description: The tester will make sure that the character?s movement functions as expected

4. Run Left (Starting from moving right)

   Type: Dynamic, Manual

Initial State: In-game state where the character is currently moving right at a constant speed

Input: LEFT_ARROW or A_KEY on keyboard are pressed

Output: The character will be moving towards the left at a constant speed

Tester(s): Development Team and Colleagues

Description: The tester will make sure that the character's movement functions as expected

5. Stop (Starting from moving right)

   Type: Dynamic, Manual

   Initial State: In-game state where the character is currently moving right at a constant speed

   Input: RIGHT_ARROW or D_KEY on keyboard are released

   Output: The character will reach a state of rest

   Tester(s): Development Team and Colleagues

   Description: The tester will make sure that the character stops moving accordingly

6. Stop (Starting from moving left)

   Type: Dynamic, Manual

   Initial State: In-game state where the character is currently moving right at a constant speed

   Input: LEFT_ARROW or A_KEY on keyboard are released

   Output: The character will reach a state of rest

   Tester(s): Development Team and Colleagues

   Description: The tester will make sure that the character stops moving accordingly

7. Jump (Starting from Idle)

Type: Dynamic, Manual

Initial State: In-game state where the character is not moving

Input: SPACE_BAR on keyboard is pressed

Output: The character will move upwards with an instantaneous force against a platform object it is on top of

Tester(s): Development Team and Colleagues

Description: The tester will make sure that the motion of jumping stays consistent through the trial. The tester must make sure that no continuous anti-gravitational force is applied.

8. Jump (Starting from running)

Type: Dynamic, Manual

Initial State: In-game state where the character is currently running

Input: SPACE_BAR on keyboard is pressed

Output: The character will move upwards with an instantaneous force against a platform object it is on top of and should contain the current horizontal speed

Tester(s): Development Team and Colleagues

Description: The tester will make sure that the motion of jumping stays consistent through the trial. The tester must make sure that no continuous anti-gravitational force is applied.

9. Jump (While airborne)

Type: Dynamic, Manual

Initial State: In-game state where the character is currently midair

Input: SPACE_BAR on keyboard is pressed

Output: The character?s current movement will continue without alterations

Tester(s): Development Team and Colleagues

Description: The tester must make sure that no continuous anti-gravitational force is applied.

10. Fire Blue Portal

    Type: Dynamic, Manual

    Initial State: Any In-game state

    Input: LEFT_CLICK on supported platform

    Output: Blue portal is formed on platform

    Tester(s): Development Team and Colleagues

    Description: The tester checks if portals are able to be fired on a platform that is at least two units long.

11. Fire Orange Portal

    Type: Dynamic, Manual

    Initial State: Any in-game state

    Input: RIGHT_CLICK on supported platform

    Output: Orange portal is formed on platform

    Tester(s): Development Team and Colleagues

    Description: The tester checks if portals are able to be fired on a platform that is at least two units long.

12. Pause

    Type: Dynamic, Manual

    Initial State: Any In-Game State

    Input: P_KEY on keyboard is pressed

    Output: The Pause menu is brought up

    Tester(s): Development Team and Colleagues

    Description: The tester checks to see if they can pause the current game

13. Play Game

Type: Dynamic, Manual

Initial State: Main Menu

Input: LEFT_CLICK on PLAY option in the main menu

Output: The game begins

Tester(s): Development Team and Colleagues

Description: The tester checks to see if they can start a new game

14. Help

    Type: Dynamic, Manual

    Initial State: Main Menu

    Input: LEFT_CLICK on HELP option in the main menu

    Output: The help menu is brought up

    Tester(s): Development Team and Colleagues

    Description: The tester checks to see if they can view the help menu

3.1.2 Object Collision Testing

1. Collision with Wall

    Type: Dynamic, Manual

    Initial State: Character is moving toward wall

    Input/Condition: Character hits wall object

    Output: Character is stopped by wall

    Tester(s): Development Team and Colleagues

    Description: The tester checks if characters make impact with wall

2. Collision with Floor Platform

    Type: Dynamic, Manual

    Initial State: Character is on/falling towards floor platform

    Input/Condition: Character hits floor object

Output: Character lands and stays on platform

Tester(s): Development Team and Colleagues

Description: The tester checks if floor platforms are working as expected

3. Collision with Castle

   Type: Dynamic, Manual

   Initial State: Character is moving toward a flag

   Input/Condition: Character hits the flag object

   Output: Level is won user stops controlling character

   Tester(s): Development Team and Colleagues

   Description: The tester checks if flag objects work and can finish the level

4. Collision with Blue Portal

   Type: Dynamic, Manual

   Initial State: Game Object is moving toward portal

   Input/Condition: Game Object hits blue portal object

   Output: Game Object teleports to the orange portal?s location conserving previous movement speed

   Tester(s): Development Team and Colleagues

   Description: The tester checks if Game Objects can teleport through portals

5. Collision with Orange Portal

   Type: Dynamic, Manual

   Initial State: Game Object is moving toward portal

   Input/Condition: Game Object hits orange portal object

   Output: Game Object teleports to the blue portal?s location conserving previous movement speed

Tester(s): Development Team and Colleagues

Description: The tester checks if Game Objects can teleport through portals

6. Collision with Goombas (front facing collision)

   Type: Dynamic, Manual

   Initial State: Character is moving toward a Goomba

   Input/Condition: Character hits a Goomba body front first

   Output: Character loses a life

   Tester(s): Development Team and Colleagues

   Description: The tester checks if Goombas can defeat the character

7. Collision with Goombas (foot first collision)

   Type: Dynamic, Manual

   Initial State: Character is moving toward a Goomba

   Input/Condition: Character hits a Goomba body foot first

   Output: Goomba is defeated

   Tester(s): Development Team and Colleagues

   Description: The tester checks if they can defeat the Goomba

3.2 Tests for Nonfunctional Requirements

The different tests listed will demonstrate that the non-functional requirements in the software requirements specification are met.

3.2.1 Look and Feel Requirements

1. Game Environment

   Type: Dynamic, Manual

   Initial State: In-game state

Tester(s): Development Team, Colleagues and/or testing group

Description: The tester will see if the different level environments are consistent and are also similar to the original game.

2. Game Hud/Interface

   Type: Dynamic, Manual

   Initial State: In-game state

   Tester(s): Development Team, Colleagues, and/or testing group

   Description: The tester will make sure that the score, time, lives, and amount of coins is not obstructive in the game's view.

3.2.2 Usability and Humanity Requirements

1. Ease of Learning

   Type: Dynamic, Manual

   Initial State: In-game state

   Tester(s): Development Team, Colleagues, and/or testing group

   Description: The tester will play through the game and will inform the development team of clarifications. Furthermore, beta tester and survey feedback will quantify the game difficulty.

2. Entertainment

   Type: Dynamic, Manual

   Initial State: In-game state

   Tester(s): Development Team, Colleagues, and/or testing group

   Description: The tester will make sure that the game is entertaining and follows similar principles to that of the original game.

3.2.3 Performance Requirements

1. Controls/Commands

   Type: Dynamic, Manual

   Initial State: In-game state

   Tester(s): Development Team, Colleagues, and/or testing group

   Description: The tester will make sure that the game does not have any noticeable delays with controls, as well as any controls that seem odd/difficult to understand.

3.2.4 Operational and Environment Requirements

1. Operating System Support

   Type: Dynamic, Manual

   Initial State: Downloading/Installing

   Tester(s): Development Team, Colleagues, and/or testing group.

   Description: The tester will make sure that the game is able to run on Windows, MacOS, and Ubuntu.

3.2.5 Security Requirements

1. Altering Information

   Type: Dynamic, Manual

   Initial State: In-game state

   Tester(s): Development Team, Colleagues, and/or testing group

   Description: The tester will make sure that the game does not alter any files or processes that are not directly related to the game.

3.2.6 Cultural Requirements

1. Spelling and Grammar

Type: Dynamic, Manual

Initial State: In-game state

Tester(s): Development Team, Colleagues, and/or testing group

Description: The tester will make sure that the game has no spelling/grammar errors and that messages, menus, and overall interface is written in English.

2. Offensive Content

Type: Dynamic, Manual

Initial State: In-game state

Tester(s): Development Team, Colleagues, and/or testing group

Description: The tester will make sure that the game has no offensive content towards culture (religion, politics, ethnics, race, etc...).

3.2.7 Legal Requirements

1. License Adherence

Type: Dynamic, Manual

Tester(s): Development Team, and Colleagues

Description: The tester will make sure that the game is not breaching the license that comes along with the game.

3.2.8 Health and Safety Requirements

1. Epileptic Prevention

Type: Dynamic, Manual

Initial State: In-game state

Tester(s): Development Team, Colleagues, and/or testing group

Description: The tester will make sure that the game does not trigger epileptic seizures as a result from playing.

4 Tests for Proof of Concept

A proof on concept test will be used to show that the development for Mari0 is feasible with the current skills and technology we have available to us. This section describes the proof of concept test and the details associated with it.

4.1 Demonstration Plan

For a proof of concept test we will create a small prototype that will be ran from Unity that can be used on Windows 10, MacOS, and Ubuntu. The prototype will be a small game demo demonstrating collision detection with different in game objects, the main gravity system, and the portal interactions. Many of these different game elements will be implemented using the Unity's collision and physics engines, as this will make our final goal easier to achieve. The main graphics that are used in the actual game will be used for this demo.

The prototype will be a floor that will be similar to the final game, which will be populated with the player character, six portals, two pipes, and platforms which the character can interact with. The player will be able to stand on the main floor and the platforms. There are also no walls to contain the character on either side. The player (which will be represented by Mario) can be controlled in the following ways:

- The player moves left and right with the 'a' and 'd' keys respectively
- The player can jump by using the spacebar

The player will interact with the different objects in the following ways:

- The floor will be the main platform that the user will be able to stand on.
- The pipes will act like walls when approached from the side and not allow the user to pass through, and act like a floor when approached from the top.
- All 6 portals are paired in different ways, when entering a blue portal, the character will exit the orange portal, and vice versa.
- All physics will be maintained when entering through a portal, and portals can be on any surface that is at least two units.

Proof of Concept Demonstration Many of the things that are demonstrated in the proof of concept will be stated in the System Tests section of this document. These tests include. Player Movement, Portal Collision, and Ground and Wall Collision.

1. Proof of Concept

   Type: Manual

   Tester(s): Development Team and Colleagues

   Description: Tests whether significant risks to the completion of the project can be overcome.

5 Comparison to Existing Implementation

Since the functional and non-function requirements that Mari0 must meet are derived from the original implementation, all system tests that test whether or not these requirements are met will be the metric used to determine the likeness of our reimplementation to the original product. We will make sure that we test our product in parallel with the current product that is available. We will ensure that our game meets the same requirements as the original. These tests will include a lot of the non-functional requirements such as Look and Feel, and Performance Requirements. Many of the different functional requirements will also be tested alongside with the original product to ensure accuracy. These requirements will be movement, portal colllision, wall and ground collision, enemy collision, and player physics.

6 Unit Testing Plan

The NUnit framework will be used to conduct all unit tests of Mari0. We will conduct our testing in a manner that satisfies a complete condition coverage criteria. We will ensure code coverage by making sure that all code is being ran effectively and consistently. Furthermore, having the original product for parallel testing will allow us to see what parts of the code are missing and what needs to be implemented further.

6.1 Unit testing of internal functions

Internal functions and methods that return any value type, such as integer, float, boolean, etc. can and will be tested dynamically, with player inputs or input conditions being given and comparing the actual output to the expected output. Due to the development and testing schedule, the use of drivers is not needed, and the use of stubs will be very limited if they are used at all. Since any video game is inherently dependant on player actions and how their interactions affect the game, we will be testing for complete condition coverage.

6.2 Unit testing of output files

Mari0 will have no output files to be tested. The only output file the original implementation had was a saved game file that kept track of the players progress, however, due to limiting our reimplementation of the game to a single level, a save file is unnecessary.

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

Since many functionalities of Mari0 are difficult to test, either due to testing having to be done manually or a bias by the development team during testing, the following usability survey questions have been listed to test some of these difficult functionalities

- Are the portal mechanics intuitive and easy to understand?
- Do the controls feel smooth?
- Do you feel like you have full or good control of the character?
- Are the U.I. elements obtrusive? Easily readable? Aesthetically pleasing?
- Do you think the game is fair?
- Do you think the game is fun?

Each question above will have a score associated with it that beta testers will be able to fill out and reflect on the product. The score will be from 1 - 10, 1 being negative and poor, and 10 being positive and well done. We are looking for a score of about 8 or higher on average to ensure that we have created an excellent product.