SE 3XA3: Test Plan
Legend of Python

Team # 1, Lava Boys Inc
Bilal Jaffry, jaffryb
Giacomo Loparco, loparcog
Lucas Zacharewicz, zacharel

October 26, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
10/26/2018	1.0	Initial Test Plan Created

1 General Information

1.1 Purpose

The purpose of testing this project is to establish confidence and reliability that the software for the project was implemented in a correct and verifiable manner.

1.2 Scope

The test plan allows us to form a basis for the testing the functionality and reliability of our open-source version of the The Legend Of Zelda. Our test plans objective is to prove that a user is able to play a level of our re-creation of The Legend of Zelda effectively and consistently.

The testing plan establishes an outline for how the arrange out testing for all aspects of the software product. The following documentation will outline the various testing methods and the tools that will be used in the process.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Terminology	**Definition**
PoC	Proof of Concept
NES	Nintendo Entertainment System
UI	User Interface
SRS	Software Requirement Specification

Table 3: **Table of Definitions**

Terminology	**Definition**
Manual Testing	Testing conducted by human interaction.
Dynamic Testing	Testing is conducted by asserting and executing test cases at run-time.
Static Testing	Testing that is not conducted by executing any program.
Structural Testing	Testing conducted regarding the internal software structure.
Functional Testing	Testing conducted from a the functional requirements, how the program should function when executed.
Stress Test Mode	Testing mode where the performance of the software is tested under heavy amounts of load.
User	Person who is playing the game.
Player	Represent the controllable in-game character.
Sprite	Images representing environment, UI elements, enemy, and player characters.
Pygame	Cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language.
Pytest	Testing framework for the Python programming language, supporting complex functional testing for applications and libraries.
The Legend of Zelda	A game released by Nintendo in 1986 for the Nintendo Entertainment System.
Nintendo Switch	Game console released by Nintendo in 2017.
Link	This the original name of the user controlled, player character.
Enemy (AI)	This is the Artificial Intelligence that governs the enemies that are harmful to the player-character.
Dragon	The Dragon enemny that the user interacts with in the game.
Keese	This is the Bat enemy that the user interacts with in the game.
Stalfos	This is the Skeleton enemy that user interacts with in the game.
Level/Dungeon	This represents the entire playable space that the user is able to explore.
Rupy	This item found in game is collected by the user to increase total points score.

1.4 Overview of Document

This project will be a re-implementation of the open source project for the NES original game, The Legend of Zelda. The game allows users to control a playable character by the name of Link, allowing them to interact with enemy AI's. The user will solve puzzles and defeat foes on their way to complete the entire level. Software requirements regarding this project can be found in the SRS revision 0 document.

2 Plan

2.1 Software Description

The software will allow users to interact with and explore a custom designed level/dungeon as the playable character Link. This implementation will be completed in Python.

Table 4: **Testing Assignments**

Date	**Task**	**Member**
10/5/2018	Requirements Document Revision 0	ALL
10/16/2018	Proof of Concept Demonstration	ALL
10/26/2018	Test Plan Revision 0	ALL

2.2 Test Team

The individuals involved for the testing are Giacomo Loparco (GL), Lucas Zacharewicz (LZ), and Bilal Jaffry (BJ).

2.3 Automated Testing Approach and Testing Tools

The testing framework that will be used will be the Pytest libary. It will be used to automate the unit testing.

2.4 Testing Schedule

Table 5: **Task Assignments**

Date	Member	Task
10/31/2018	BJ and GL	User and Player Interactions
11/8/2018	GL and LZ	Environment and Interactions
11/14/2018	BJ and GL	Look and Feel
11/17/2018	ALL	Usability
11/20/2018	LZ	Performance

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 User and Player Interactions

User Inputs

1. FR-USR-01

   Type: Functional, Dynamic, Manual

   Initial State: Player is in a room with valid movement options (not blocked in any direction)

   Input: User presses a single movement key (W, A, S, or D) on the keyboard

   Output: Player moves in the direction specified by the key pressed, until the key is let go

   How test will be performed: Within an empty room with no barriers, the player will be loaded into the game, and each movement key will be pressed, held, and let go, to see if the player moves in the intended direction or if there was an error

2. FR-USR-02

Type: Functional, Dynamic, Manual

Initial State: Player is in a room with valid movement options

Input: User presses and holds down a movement key, and then presses and holds another movement key concurrently with the first

Output: Player keeps moving in the direction specified by the first key held down

How test will be performed: Within an empty room with no barriers, the player will be loaded into the game, and the specified input will be put in. If the player keeps moving in the initial direction, ignoring the second direction key pressed, the test is a success. If the player slows, stops, or changes direction, an error has occurred.

3. FR-USR-03

Type: Functional, Dynamic, Manual

Initial State: Player is in a room

Input: User presses the attack key

Output: Player moves into "attack state" (movement not allowed, sword spawned in direction), and then goes back to their initial position

How test will be performed: The player will be spawned into a room, and the user will press the attack key. When the key is pressed, the user will observe whether or not the player has changed positions and a sword has been spawned, and whether the player goes back to their initial position after some time, and the sword has been deleted. If any abnormalities occur from this result, an error has occurred.

4. FR-USR-04

Type: Functional, Dynamic, Manual

Initial State: Player is in a room

Input: User presses the attack key, and holds a movement key while player is in "attack state"

Output: Player stays in current attack state, and does not move

How test will be performed: The player will be spawned into a room, and the user will press the attack key. When the key is pressed, the user will immediately press a movement key. If the player does not move or change states while in the attack state, the test is a success. If not, then an error has occurred.

5. FR-USR-05

   Type: Functional, Dynamic, Manual

   Initial State: Player is in a room

   Input: User repeatedly presses the attack key

   Output: Player pauses between attacks

   How test will be performed: The player will be spawned into a room, and the user will rapidly press the attack key. The user will then observe whether the player consistently attacks whenever the key is pressed, or if there is a brief pause, specified by the code, in between attacks, ignoring the attack input. If this brief pause is observed, the test has passed. Otherwise, an error has occurred.

6. FR-USR-06

   Type: Functional, Dynamic, Manual

   Initial State: Player is in a room, with boomerang item in inventory

   Input: User presses the use item key

   Output: Player uses their boomerang

   How test will be performed: The player will be spawned into a room with a boomerang in their inventory, and the user will press the use item key. When the key is pressed, the user will observe whether a boomerang is spawned in front of the player. If this does not occur, an error has occurred.

Player Interactions

7. FR-PLYR-01

Type: Functional, Dynamic, Manual

Initial State: Player and wall in a room

Input: Player moved towards wall

Output: Player stopped whenever the player sprite touches the wall sprite

How test will be performed: The player will be spawned into a room with a wall. The user will move the player towards the wall, and observe what happens when the two objects, player and wall, collide, and if the specified output is achieved.

8. FR-PLYR-02

   Type: Functional, Dynamic, Manual

   Initial State: Player and enemy in a room

   Input: Player moved towards enemy

   Output: Player pushed backwards (in reference to walking direction), and player health decreased by set amount (depending on enemy type)

   How test will be performed: The player will be spawned into a room with a basic enemy (ex. a keese). The user will move the player towards the enemy, and observe what happens when the two objects, player and enemy, collide, and if the specified output is achieved.

9. FR-PLYR-03

   Type: Functional, Dynamic, Manual

   Initial State: Player and consumable item (rupy) in a room

   Input: Player moved towards item

   Output: Item deleted, player rupy count incremented by 1

   How test will be performed: The player will be spawned into a room with a consumable item (rupy). The user will move the player towards the item, and observe what happens when the two objects, player and item, collide, and if the specified output is achieved.

10. FR-PLYR-04

    Type: Functional, Dynamic, Manual

    Initial State: Player and consumable item (key) in a room

    Input: Player moved towards item

    Output: Item deleted, player key count incremented by 1

    How test will be performed: The player will be spawned into a room with a consumable item (key). The user will move the player towards the item, and observe what happens when the two objects, player and item, collide, and if the specified output is achieved.

11. FR-PLYR-05

    Type: Functional, Dynamic, Manual

    Initial State: Player and consumable item (heart) in a room

    Input: Player moved towards item

    Output: If player health is less than max, the heart is consumed and the player health is brought up by one, or visually, one heart. If the player's health is full, the item is not consumed and no collision event occurs

    How test will be performed: The player will be spawned into a room with a consumable item (heart). The user will move the player towards the item, once with full health and once with the player's health less than the player's max health, and observe what happens when the two objects collide.

12. FR-PLYR-06

    Type: Functional, Dynamic, Manual

    Initial State: Player and boomerang item in a room

    Input: Player moved towards item

    Output: The boomerang item is added to the user's inventory

    How test will be performed: The player will be spawned into a room with a collectible item (boomerang). The user will move the player

towards the item, and observe what happens when the two objects, player and boomerang, collide, and if the specified output is achieved.

13. FR-PLYR-07

    Type: Functional, Dynamic, Manual

    Initial State: Player and keese enemy in room

    Input: Player is in attack state, and the player's sword collides with an enemy

    Output: The enemy's health decrements itself by 1

    How test will be performed: The player will be spawned into a room with a keese enemy, who only have a max health of 1. The player will attack when near and facing an enemy, so the sword and the enemy collide. If the keese is killed (deleted from the screen), the test was a success, and if not, an error has occurred.

14. FR-PLYR-08

    Type: Functional, Dynamic, Manual

    Initial State: Player and keese enemy in room

    Input: Player collides with keese enemy repeatedly

    Output: Player health reduced to 0, and game over screen appears

    How test will be performed: The player will be spawned into a room with a stalfos enemy. The player will repeatedly collide with the keese until the health displayed on screen is reduced to 0, and then observe if the game state moves to the "Game over" screen.

15. FR-PLYR-09

    Type: Functional, Dynamic, Manual

    Initial State: Player with boomerang in room

    Input: Player uses boomerang

    Output: Boomerang object is spawned, moves a certain distance away from player, and then moves back, deleting itself when colliding with the player

How test will be performed: The player will be spawned into a room with a boomerang in their inventory. The user will press the use item key and observe if the specified output is achieved.

16. FR-PLYR-10

    Type: Functional, Dynamic, Manual

    Initial State: Player with boomerang and enemy in room

    Input: Player uses boomerang and boomerang collides with enemy

    Output: Enemy stops moving for a set amount of time

    How test will be performed: The player will be spawned into a room with a boomerang in their inventory. The user will press the use item key when facing the enemy, in a range for the boomerang to hit them, and observe if the specified output is achieved.

17. FR-PLYR-11

    Type: Functional, Dynamic, Manual

    Initial State: Player with final objective item in room

    Input: Player collides with final objective item

    Output: "Game Complete" screen appears, game over

    How test will be performed: The player will be spawned into a room with the final objective item. The user will move the player towards the item and observe what happens when the player and item collide.

3.1.2 Environment Interaction

Enemy Interactions

17. FR-ENMY-01

    Type: Functional, Dynamic, Manual

    Initial State: Group of 3-4 keese in dungeon room (wall barriers at all edges)

    Input: N/A

Output: Keese stays within the confines of the window, passing through walls but not leaving the viewable area, not getting stuck

How test will be performed: The keese will be spawned in a room and the user will observe the keese for 1-2 minutes to determine whether the output is achieved.

18. FR-ENMY-02

    Type: Functional, Dynamic, Manual

    Initial State: Group of 3-4 keese in dungeon room (wall barriers at all edges)

    Input: N/A

    Output: Keese movement consistent to that of the original project

    How test will be performed: The user will observe the keese movement in both the original project and this project, and see if the behavior, speed, and general idea of the keese is consistent with the original project.

19. FR-ENMY-03

    Type: Functional, Dynamic, Manual

    Initial State: Group of 3-4 stalfos in dungeon room (wall barriers at all edges), with wall objects in the middle of the room

    Input: N/A

    Output: Stalfos collides with walls and does not go through them, not getting stuck

    How test will be performed: The stalfos will be spawned in a room and the user will observe the stalfos for 1-2 minutes to determine whether the output is achieved.

20. FR-ENMY-04

    Type: Functional, Dynamic, Manual

    Initial State: Group of 3-4 stalfos in dungeon room (wall barriers at all edges)

Input: N/A

Output: Stalfos movement consistent to that of the original project

How test will be performed: The user will observe the stalfos movement in both the original project and this project, and see if the behavior, speed, and general idea of the stalfos is consistent with the original project.

21. FR-ENMY-05

    Type: Functional, Dynamic, Manual

    Initial State: Room with dragon

    Input: N/A

    Output: Dragon constantly spawning fireballs at certain time intervals

    How test will be performed: The dragon will be spawned in a room and the user will observe the dragon 1-2 minutes to determine whether the output is achieved.

22. FR-ENMY-06

    Type: Functional, Dynamic, Manual

    Initial State: Room with dragon

    Input: N/A

    Output: Dragon attacks consistent to that of the original project

    How test will be performed: The user will observe the dragon attacks in both the original project and this project, and see if the behavior and general idea of the dragon is consistent with the original project.

Dungeon Interactions/Creation

23. FR-DUNG-01

    Type: Functional, Dynamic, Manual

    Initial State: Player in room with open door

    Input: Player collides with door

Output: Player moves to another room of the dungeon

How test will be performed: The player will be spawned into a room with a door, and a corresponding room connected to load once the player has collided with, or "walked through" the door. The user will move the player towards the door to then collide with it and observe the reaction.

24. FR-DUNG-02

    Type: Functional, Dynamic, Manual

    Initial State: Player with ¿0 keys in room with locked door

    Input: Player collides with door

    Output: Locked door turns into open door

    How test will be performed: The player will be spawned into a room with a locked door. The player will collide with the door and the user will observe the result.

25. FR-DUNG-03

    Type: Functional, Dynamic, Manual

    Initial State: Player with 0 keys in room with locked door

    Input: Player collides with door

    Output: Locked door stays locked (no change)

    How test will be performed: The player will be spawned into a room with a locked door. The player will collide with the door and the user will observe the result.

26. FR-DUNG-04

    Type: Functional, Dynamic, Manual

    Initial State: Player with ¿0 keys in room with locked door

    Input: Player collides with door

    Output: Locked door turns into open door

How test will be performed: The player will be spawned into a room with a locked door. The player will collide with the door and the user will observe the result.

27. FR-DUNG-05

    Type: Functional, Dynamic, Manual

    Initial State: Player with blocked objective door

    Input: Player completes objective (ex. kill all enemies in room)

    Output: Blocked objective door turns into open door

    How test will be performed: The player will be spawned into a room with an blocked objective door and an objective to complete. The player will complete the objective and observe the resulting door change.

28. FR-DUNG-06

    Type: Functional, Dynamic, Manual

    Initial State: Full dungeon created with player at starting room

    Input: Player walks through dungeon

    Output: Final rooms lead to dragon boss and final objective item

    How test will be performed: The player will be spawned into a full-game test, and observe whether the game produces a map where you progress from the start to the end of the map, being able to visit all rooms, and be led to the end of the dungeon where there is the dragon boss, and then the final item to collect for a game over

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel

1. NFC-LF-01

    Type: Structural, Manual

    Initial State: Game is launched and running.

    Input/Condition: User is asked to play the game.

Output/Result: The majority of the users report that there is no noticeable decline in quality with the animations.

How test will be performed: The game will be loaded and running for the user to sit down and spend time playing the game. The users will be asked a series of question about their immersion in the game world and whether or not they saw any oddities relating to the animations in the game.

2. NFC-LF-02

   Type: Manual

   Initial State: Users have completed the above test and will play a version of The Legend of Zelda on the Nintendo Switch.

   Input: Users play The Legend of Zelda on the Nintendo Switch.

   Output: The majority of users are not able to decern the differences in the animation between the original and the project.

   How test will be performed: The users will sit down with the original version of The Legend of Zelda to play and answer some questions about how the animations compare between the two games.

3.2.2 Usability

3. NFC-USE-01

   Type: Manual

   Initial State: Game is launched and running.

   Input: User is asked to play the game.

   Output: The majority of users report that the controls were easy to learn and the learning curve of game mechanics is low.

   How test will be performed: The game will be loaded and running for the user to play. The user will spend some time with the game and be asked question about how easy it was to learn the game and its mechanics.

4. NFC-USE-02

   Type: Manual

   Initial State: Users have completed the above test and will play a version of The Legend of Zelda on the Nintendo Switch.

   Input: Users play The Legend of Zelda on the Nintendo Switch

   Output: The majority of the users are not able to distinguish between the quality of gameplay between the two versions

   How test will be performed: The users will sit down with the original version of The Legend of Zelda to play and they will answer questions on how the quality of gameplay and mechanics compare between the two versions.

3.2.3 Performance

5. NFC-PER-01

   Type: Dynamic, Structural

   Initial State: Game is launched in stress test mode.

   Input: No input is required.

   Output: The game outputs statistics about how the game ran (i.e. average frames per second, max frames per second, min frames per second, etc.)

   How the test will be performed: The game gets launched in stress test mode where it will load a room of enemies and other game objects and let them interact with the environment which allows for the judgment of performance quality for the game. These results can be compared to the performance of the project under regular stress, and how far it degrades.

6. NFC-PER-02

   Type: Dynamic, Structural

   Initial State: Game is launched in stress test mode

Input: No input required

Output: The test outputs the statistics on the loading performances (i.e. average load time, max load time, min load time, etc)

How the test will be performed: The game gets launched in stress test mode where it will load rooms with large amounts of enemies and other game objects and time how long it takes to load for each room loaded in this way. The test then reports the measured statistics.

4 Tests for Proof of Concept

Proof of Concept testing was focused on verifying the functionality of the user input from the keyboard and if the game could be effectively tested using manual testing. The manual testing would require the Player character to spawn at a pre-determined location on the screen, and be able to move around freely in the bounds of the level. The user would then be able to interact with enemy AI and interact with them to defeat them. Therefore, the testing for the Proof of Concept involved the ability for the user to control the playable character consistently, interact with basic enemy AI and the functionality of enemy and wall collision.

4.1 Input Testing

1. INP-1

   Type: Functional, Manual

   Initial State: Link sprite is rendered to the screen at a pre-determined (x,y) location

   Input: The [W,A,S,D] keys on a standard keyboard, representing the [+y,-x,-y,+x], moving the player character in the respective directions

   Ouput: The Link sprite will update its location at interval of of 3 units per frame in the respective +x,-x, +y,-y direction, at a 60 Hz refresh rate

How test will be performed: The user will use any of the [W,A,S,D] to move the sprite in the respective direction. The user will observe if the sprite updates 3 units for every frame when the key is held. Once the key is released the sprite should remain stationary at the new (x,y) position. If they key is pressed and the sprite does not update its (x,y) correctly on the screen or when the key is released does not remain the in the last position, then an error has occurred.

2. INP-2

Type: Function, Manual

Initial State: Link sprite is rendered to the screen at a pre-determined (x,y) location. The sprite could have been updated to another (x,y) location after movement.

Input: The user uses the [K,L] keys on the keyboard for weapon use and item use respectively.

Output: When the user uses the [K] the sword sprite will be updated and will damage of the enemy AI when there is contact made between the two sprites. The current item inventory will display a 'Item Used' message to the developer console

How test will be performed: The [K] key press will update the sprite according to the proper orientation (Up,Down,Left,Right) with a Link attacking sprite. This will be checked for all directions and ensure the once the [K] key is pressed, the user cannot move the player character. If the sprite collides with an enemy AI sprite the enemy AI will be removed off screen and a randomly generated item will be displayed. If at any point the [K] key does not display the sprite animation, aswell as not damaging the enemy AI on collision, then an error has occurred. If the [L] key is used and the 'Item Used' message does not appear at any point, then an error has also occurred

4.2 Enemy Testing

1. ENM-1

   Type: Function, Manual

   Initial State: The Keese enemy AI is spawned at a pre-determined location

   Input: No user input is taken

   Output: The Keese enemy AI moves within the bounds of the screen. The sprite animates for every new location it moves to on the screen. If the sprite collides with the user character at any point, the Link sprite should be moved in the direction the Keese was moving it as well as update the player health by lowering it by a half each time the Link sprite collides withe Keese sprite

   How test will be performed: The test will be performed by placing a Keese actor onto to the screen and observing its interactions with the player character and if the user stays within the bounds

4.3 Player Collision

1. COL-1

   Type: Function, Manual

   Initial State: Link sprite is rendered to the screen at a pre-determined (x,y) location

   Input: The user inputs the movement keys [W,A,S,D] on keyboard

   Output: The user will not be able to update its (x,y) when the sprite collides with a wall sprite. Will not be able to leave the bounds of the

room sprite

How test will be performed: The user will input a movement command, and will observe if the sprite is bounded by the wall sprites. An error has occurred if the player is able to move outside of the viewable space of the window.

5 Comparison to Existing Implementation

There are four tests in this test plan that compare this project to it's existing implementation as a check for correctness. Please view them at their corresponding test ID's:

- FR-ENMY-02
- FR-ENMY-04
- FR-ENMY-06
- NFC-USE-02

6 Unit Testing Plan

The unit testing suite PyTest will be used for unit testing this project. PyTest-cov will be used for testing statement coverage.

6.1 Unit testing of internal functions

Unit testing works on internal functions that return testable values. With this knowledge, unit test are best produced when modules are separated from one another to test the how they perform in a vacuum. Tests will give methods some input and compare against the expected output to check for validity. These inputs can be either valid in which case the module shall produce the desired output, or they could be invalid inputs which shall produce exceptions that the unit tests will be able to catch. With the PyTest-cov plug-in, statement coverage can be tested while unit testing is done. The goal is to cover as high a percent of unit testable code as possible with the

unit tests to ensure correctness and stability in the project. Our goal is at least 70 percent coverage.

6.2 Unit testing of output files

N/A, as the game does not create any output files.