SE 3XA3: Test Report
Legend of Python

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

December 4, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
December 3rd	1.0	Rev 1 Submission

1 Functional Requirements Evaluation

1.1 User Input Tests

Test Name: FR-USR-01

Result: The user is able to move in all intended directions.

Test Name: FR-USR-02

Result: Player is locked into one direction as long as they hold that key (intended behavior).

Test Name: FR-USR-03

Result: The user is able to enter and leave the attack state.

Test Name: FR-USR-04

Result: The user is unable to move during the attack state.

Test Name: FR-USR-05

Result: The user experiences an input delay between attacks.

Test Name: FR-USR-06

Result: THe player is able to use the boomerang item.

1.2 Player Interaction Tests

Test Name: FR-PLYR-01

Result: The player collides with the wall and is unable to move through it.

Test Name: FR-PLYR-02

Result: The player collides with the enemy, takes damage, and is knocked back.

Test Name: FR-PLYR-03

Result: The player collides and picks up the item with the rupee count incrementing by 1.

Test Name: FR-PLYR-04

Result: The player collides and picks up the item with the key count incrementing by 1.

Test Name: FR-PLYR-05

Result: The player collides and picks up the item with the health count incrementing by 1.

Test Name: FR-PLYR-06

Result: The player collides with the boomerang and it is added to the player's inventory.

Test Name: FR-PLYR-07

Result: The player enters the attack state, the player's sword collides with the enemey, and the enemy takes the appropriate amount of damage.

Test Name: FR-PLYR-08

Result: The player dies and the game over screen appears.

Test Name: FR-PLYR-09

Result: The player uses the boomerang, it moves in the same direction the player is facing, the boomerang returns to the player.

Test Name: FR-PLYR-10

Result: The player throws the boomerang at the enemy stunning it for the set amount of time.

Test Name: FR-PLYR-11

Result: The player collides with the final item and the game complete screen appears.

1.3 Enemy Interaction Tests

Test Name: FR-ENMY-01

Result: Keese stays within the confines of the room.

Test Name: FR-ENMY-02

Result: The keese movement is consistent with the original projects.

Test Name: FR-ENMY-03

Result: Stalfos stays within the confines of the room.

Test Name: FR-ENMY-04

Result: The stalfos movement is consistent with the original projects.

Test Name: FR-ENMY-05

Result: Aquamentus attacks at specified time intervals.

Test Name: FR-ENMY-06

Result: Aquamentus's attacks are consistent with the original version.

1.4 Dungeon Interactions/Creation Tests

Test Name: FR-DUNG-01

Result: The player is able to move between rooms.

Test Name: FR-DUNG-02

Result: The player is able to unlock the door.

Test Name: FR-DUNG-03

Result: The player collides with the door and it stays locked.

Test Name: FR-DUNG-04

Result: The player is able to unlock the door.

Test Name: FR-DUNG-05

Result: The player completes the objective and the door opens.

Test Name: FR-DUNG-06

Result: The player is spawned into the full map and is able to traverse every room.

2 Nonfunctional Requirements Evaluation

2.1 Look and Feel

Test Name: NFC-LF-01

Result: 4 of 4 users did not indicate a decline in animation quality.

Test Name: NFC-LF-02

Result: 4 of 4 users did not discern any difference between this implementation and the original project.

2.2 Usability

Test Name: NFC-USE-01

Result: 4 of 4 users indicated that the controls were easy to learn with a low learning curve.

Test Name: NFC-USE-02

Result: 4 of 4 indicated that the mechanics between the two implementations are similar.

2.3 Performance

Test Name: NFC-PER-01

Result: The game showed no signs of slowed down during stress testing.

Test Name: NFC-PER-02

Result: The game showed quick loading times under 1 second.

3 Comparison to Existing Implementation

There are several differences between the existing implementations of The Legend of Zelda and our version of it. The majority of the differences come with the sheer amount of content lacking from our implementation as our scope had to be narrowed in order to keep on track with the project. This encapsulates features such as the main overworld, additional items for the player's use, secrete areas, and the remaining 7 dungeons not imagined in our implementation. With those differences out of the way, the remaining comparison comes to the look and feel of each of the implementations and their functionality compared to the original. The open source project we chose had no animations and was not a running in a game loop (i.e. everything would update when the player made a move). Our implementation aimed to be closer to the original version of the game than the open source project with clear benefit in playability over the open source implementation. Provided we continue with the project in the future, our implementation has potential to encapsulate a better experience than the open source project and perhaps the original.

4 Unit Testing

4.1 Aquamentus

Test Name	AQU1
Initial State	Aquamentus Constructor
Input	Valid x and y coordinates to create an aquamentus object in
Expected Output	The aquamentus object is at the given x value, and at the y value plus the game's Y-offset, for the hud

Table 2: Test for AQU1

Test Name	AQU2
Initial State	Aquamentus moving in an x direction (forward)
Input	Function to reverse direction
Expected Output	Aquamentus velocity is now negative of the original state (moving backwards)

Table 3: Test for AQU2

Test Name	AQU3
Initial State	Aquamentus not attacking
Input	Function to make aquamentus object attack
Expected Output	Aquamentus is now attacking

Table 4: Test for AQU3

4.2 Boss

Test Name	BOSS1
Initial State	Boss Constructor
Input	Valid x and y coordinates
Expected Output	The boss object is at the given x value, and at the y value plus the game's Y-offset, for the hud

Table 5: Test for BOSS1

Test Name	BOSS2
Initial State	Boss with speed 3
Input	Function to make the boss move
Expected Output	Boss x-coordinate is now 3 pixels more than what it was before ($xnew = xold + 3$)

Table 6: Test for BOSS2

Test Name	BOSS3
Initial State	Boss object
Input	Function to make boss take one point of health damage
Expected Output	The boss is now hit (boolean is now true)

Table 7: Test for BOSS3

4.3 Enemy

Test Name	ENM1
Initial State	Enemy Constructor
Input	Valid x and y coordinates
Expected Output	The enemy object is at the given x value, and at the y value plus the game's Y-offset, for the hud

Table 8: Test for ENM1

Test Name	ENM2
Initial State	Enemy with speed 3
Input	Function to make the enemy move
Expected Output	Enemy x-coordinate is now 3 pixels more than what it was before (xnew = xold + 3)

Table 9: Test for ENM2

Test Name	ENM3
Initial State	Enemy object
Input	Function to make enemy take one point of health damage
Expected Output	The enemy is now hit (boolean is now true)

Table 10: Test for ENM3

4.4 Fireball

Test Name	FBL1
Initial State	Fireball Constructor
Input	Valid x and y coordinates, along with x and y directional speeds
Expected Output	The fireball object is at the given x and y coordinates, with the same given x and y direction speeds

Table 11: Test for FBL1

Test Name	FBL2
Initial State	Fireball object
Input	Function to start movement of fireball, giving an (x, y) position and (x, y) velocity
Expected Output	The fireball is now at the given x and y coordinates, with the new x and y directional speeds

Table 12: Test for FBL2

Test Name	FBL3
Initial State	Fireball object
Input	Function to end movement of fireball
Expected Output	The fireball is now out of screen, with its x and y coordinates at -1000, and its x and y speeds at 0

Table 13: Test for FBL3

Test Name	FBL4
Initial State	Fireball object
Input	Function to start movement of fireball, giving an (x, y) position and (x, y) velocity, along with a move function
Expected Output	The fireball is now at the given x and y coordinates, plus the x and y velocity, respectively, with the new x and y directional speeds

Table 14: Test for FBL4

4.5 Item

Test Name	ITEM1
Initial State	Item Constructor
Input	Valid x and y coordinates, along with an item type (integer from 0 to 5)
Expected Output	The item object is at the given x and y coordinates, with the same type as the one given

Table 15: Test for ITEM1

4.6 Keese

Test Name	KSE1
Initial State	Keese Constructor
Input	Valid x and y coordinates
Expected Output	The keese object is at the given x value, and at the y value plus the game's Y-offset, for the hud

Table 16: Test for KSE1

Test Name	KSE2
Initial State	Keese object
Input	Function to make the keese object rest for a random amount of time
Expected Output	The keese object is either set to wait 1 or 2 seconds, and it's sprite frames are on rest (ie only show keese with closed wings)

Table 17: Test for KSE2

Test Name	KSE3
Initial State	Keese object
Input	Function to generate a travel point for the keese to go to
Expected Output	The distance of the keese to the generated point is greater than the minimum set value

Table 18: Test for KSE3

Test Name	KSE4
Initial State	Keese object with sprite index at 0
Input	Function to make the keese object switch sprites
Expected Output	The keese object sprite index is now at 1

Table 19: Test for KSE4

Test Name	KSE5
Initial State	Keese object
Input	Function to make keese stop moving
Expected Output	The keese object's x and y velocity are set to 0

Table 20: Test for KSE5

Test Name	KSE6
Initial State	Keese object
Input	Function to set movement speed based on a specific travel point
Expected Output	The x and y velocity of the keese moves the object towards the given point

Table 21: Test for KSE6

4.7 Stalfos

Test Name	STAL1
Initial State	Stalfos Constructor
Input	Valid x and y coordinates
Expected Output	The stalfos object is at the given x value, and at the y value plus the game's Y-offset, for the hud

Table 22: Test for STAL1

Test Name	STAL2
Initial State	Stalfos object
Input	Function to make stalfos generate a travel path
Expected Output	The stalfos object has a direction from 0 to 3, and has walking frames from 0 to 3

Table 23: Test for STAL2

Test Name	STAL3
Initial State	Stalfos object
Input	Setting walking speed in different directions
Expected Output	If the direction is 0 or 2 (horizontal), the stalfos' x velocity is it's speed. If the direction is 1 or 3 (vertical), the stalfos' y velocity is it's speed.

Table 24: Test for STAL3

Test Name	STAL4
Initial State	Stalfos object
Input	Function to make stalfos stop movement
Expected Output	The stalfos' x and y velocity are both 0

Table 25: Test for STAL4

5 Changes Due to Testing

5.1 Input Testing

There have been no changes to the specified input methods as a result of completed tests.

5.2 GUI Testing

There have been no changes to the specified GUI methods as a result of completed tests.

5.3 Display Ouput Testing

There have been no changes to the specified output methods as a result of completed tests.

5.4 Level Testing

Upon testing the level transitioning for the manualTests.py module, the testing for this showed the non-playable characters showing unknown behavior when animating and moving around each level tile of the game. Upon further inspection, the cause for this was due to the sprites of a previous level being loaded below the new sprites of the current level. Each sprite in the game has a collide-able object and responds accordingly when collided with.

This was the cause for the collision between the sprites to occur and was solved with a creation of a function which remembered to clear the sprite lists rendered on screen when transitioning to another level tile in the game.

6 Automated Testing

For automated testing for the project we used Pytest as the framework for our unit testing. Pytest was used to run all out unit tests, which can be found in section 4.0 of this document labeled 'Unit Testing'. All the tests cases written in our tests suite resulted in positive results, all asserting to be correct by the Pytest framework.

7 Trace to Requirements

Test	Requirements
	Functional Requirements Testing
FR-USR-01-06	FR1, FR2, FR3, FR12, FR13
FR-PLYR-01-11	FR7, FR8, FR9, FR10, FR11, FR12, FR13, FR17, FR18, FR19, FR21, FR22, FR23, FR25, FR26
FR-ENMY-01-06	FR7, FR14, FR15, FR16
FR-DUNG-01-06	FR5, FR6, FR7, F24
	Non-functional Requirements Testing
NFC-LF-01	NFR1, NFR3, NFR5, NFR11
NFC-LF-02	NFR1
NFC-USE-01	NFR2
NFC-USE-02	NFR1, NFR6, NFR 10
NFC-PER-01	NFR7
NFC-PE-02	NFR3, NFR4
	Unit Testing
AQU1-3	FR7, FR14, FR15, FR16
BOSS1-3	FR7, FR14
ENM1-3	FR7, FR14
FBL1-4	N/A
ITEM1	FR8, FR21
KSE1-6	FR7, FR14, FR15, FR16
STAL1-4	FR7, FR14, FR15, FR16

8 Trace to Modules

Test	Modules
Functional Requirements Testing	
FR-USR-01-06	M2, M10, M15, M23
FR-PLYR-01-11	M8, M7, M10, M12, M20, M24, M25, M5
FR-ENMY-01-06	M1, M3, M5, M9, M14, M20
FR-DUNG-01-06	M16, M18, M19
Non-functional Requirements Testing	
NFC-LF-01	M13, M23
NFC-LF-02	M*
NFC-USE-01	M10, M23
NFC-USE-02	M10, M23
NFC-PER-01	M*
NFC-PE-02	M18, M19
Unit Testing	
AQU1-3	M1, M3
BOSS1-3	M3
ENM1-3	M5
FBL1-4	M6
ITEM1	M8
KSE1-6	M5, M9
STAL1-4	M5, M14

9 Code Coverage Metrics

The Legend of Python project has managed to produce code coverage for 15 of our 25 modules. This yields an approximate 70 percent code coverage for the entire project. This number was determined by our unit testing modules which consisted of 7 internal modules, which had unit test cases written for the functionality of each module. The remaining 8 modules were covered during our manual testing suite, where levels would be loaded, along with the corresponding data, and would observe the functionality of each non-playable character and the collide- able objects within the room. This can be seen documented within our module trace section. The remaining modules could only be tested by executing the game state of the program and testing all bounds of the player character and its interaction with the environment, non-playable characters and consumable items.