SE 3XA3: Test Report

Team 10, MacLunky
Albert Zhou, zhouj103
Abeer Al-Yasiri, alyasira
Niyatha Rangarajan, rangaran

April 12, 2021

Contents

List of Tables

List of Figures

Table 1: Revision History

Date	Developer(s)	Change
April 1, 2021	Albert, Abeer, Niyatha	Version 0 made

This document ...

1 Functional Requirements Evaluation

All of the following test cases will be ran using a special map file that puts the player character at different use cases and isolated environments. If the test result is a pass then the corresponding FR for that test case implies that the system has successfully satisfied those FR.

TC 3.1.1.1: System launch from source files

This test case evaluates FR1 by manually testing if the system launches correctly from the source files. The input of this test is compiling the files and the output is the system launching with a player and level. The result of the test case is a pass since player and level appear on the screen.

TC 3.1.1.2: System launch from executable

Untested.

TC 3.1.1.3: System termination

This test case evaluates FR2 by manually testing if the system terminates correctly. The input of this test is key down press of keyEsc and the output is the system terminating. The result of the test case is a pass since the system terminates without error.

TC 3.1.2.1: Camera position

This test case evaluates FR3, 4 by manually testing if the camera is positioned correctly. The input of this test is cam() and the output is the camera coordinates. The result of the test case is a pass since the camera doesn't move beyond the boundaries of the level and has the player in view.

TC 3.1.2.2: Camera move

This test case evaluates FR3, 4 by manually testing if the camera moves correctly. The input of this test is move() and the output is the camera

coordinates. The result of the test case is a pass since the camera doesn't move beyond the boundaries of the level and has the player in view.

TC 3.1.2.3: Health display

This test case evaluates FR5 by manually testing if the health is displayed correctly. The input of this test is ui() and the output is the health on the screen. The result of the test case is a pass since player's health is displayed at the top left of the screen.

TC 3.1.2.4: Health update

This test case evaluates FR5 by manually testing if the health changes are displayed correctly. The input of this test is ui() and the output is the health on the screen. The result of the test case is a pass since player's health is displayed at the top left of the screen.

TC 3.1.2.5: Gold display

This test case evaluates FR5 by manually testing if the gold is displayed correctly. The input of this test is ui() and the output is the gold on the screen. The result of the test case is a pass since player's gold is displayed at the top left of the screen under health.

TC 3.1.2.6: Gold update

This test case evaluates FR5 by manually testing if the gold changes are displayed correctly. The input of this test is ui() and the output is the gold on the screen. The result of the test case is a pass since player's gold is displayed at the top left of the screen under health.

TC 3.1.2.7: Game over

This test case evaluates FR74 by manually testing if the game ends correctly. The input of this test is ui() and the output is the game over screen. The result of the test case is a pass since the game over screen is displayed.

TC 3.1.2.8: Bombs display

This test case evaluates FR5 by manually testing if the bombs is displayed correctly. The input of this test is ui() and the output is the bombs on the screen. The result of the test case is a pass since player's bombs is displayed at the top left of the screen next to health.

TC 3.1.2.9: Bomb update

This test case evaluates FR5 by manually testing if the bombs changes are displayed correctly. The input of this test is ui() and the output is the bombs on the screen. The result of the test case is a pass since player's bombs is displayed at the top left of the screen next to health.

TC 3.1.2.10: Ropes display

This test case evaluates FR5 by manually testing if the ropes is displayed correctly. The input of this test is ui() and the output is the ropes on the screen. The result of the test case is a pass since player's ropes is displayed at the top left of the screen next to bombs.

TC 3.1.2.11: Rope update

This test case evaluates FR5 by manually testing if the ropes changes are displayed correctly. The input of this test is ui() and the output is the ropes on the screen. The result of the test case is a pass since player's ropes is displayed at the top left of the screen next to bombs.

TC 3.1.3.1: Player Jumps

This test case evaluates FR7, 15 by manually testing the player can jump maxJump distance when keyJump is pressed and player is at a stationary position. The input of this test is key down press of keyJump and the output is the displayed sprite change of the player character on the screen that represents the player with a jump speed and change in y-position upward. The result of the test case is a pass since the displayed character appears above the start position by maxJump when the keyJump is pressed.

TC 3.1.3.2: Player Moves Left

This test case evaluates FR7, 14 by manually testing the player movement to the left by one position when keyLeft is pressed and player is at stationary position. The input of this test is key down press of keyLeft and the output is the player x-position decreased and display the character at the left of the initial position. The test result is a pass since the displayed character appears one position to the left from the start position when the keyLeft is pressed.

TC 3.1.3.3: Player Moves Right

This test case evaluates FR7, 14 by manually testing the player movement to the left by one position when keyRight is pressed and player is at stationary position. The input of this test is key down press of keyRight and the output is the player x-position increased and display the character the right of the initial position. The test result is a pass since the displayed character appears one position to the right from the start position when the keyRight is pressed.

TC 3.1.3.4: Player Can't Jump at an Occurring Jump Action

This test case evaluates FR15 by manually testing the player's jumping height and behaviour when pressing keyJump and player is at jumping state with y-direction speed greater than 0. The input of this test is key down press of keyJump and the output is the player reaching a maximum height of maxJump from the stationary position and having the second key press not affecting the character height. The test result is a pass since the character was displayed to jump at a height equivalent to the player jumping once. The keyJump press within the jumping state has no affect on the player's position and speed.

TC 3.1.3.5: Player Collect

This test case evaluates FR8 by manually testing the player's functional ability to collect points and objects in the game when pressing keyInt and player is at the same position as a collectable game object, such as a gold object. The testing case start with the player's gold set to zero and at same

x and y position of a gold placed on the game path. The input of the test case is key down press of keyInt and the output is observing the change to the player's gold score at the top of the screen. The test result is a pass since player gold score was updated to valGoldBar and that value was displayed on the screen after the keyInt press.

TC 3.1.3.6: Player Opens Box

This test case evaluates FR8 by manually testing the player's functional ability to open a reward box object placed int he game path by pressing the control input keyInt. The testing environment puts the player in the same position of a reward box object. The input of the test case is key down press of keyInt and the output is the game's response of creating a reward object, such as a gold object, and displaying that object on the same box position to the screen. The result of the test case is a pass since after the keyInt press the reward was observed to be displayed at the box position and that it can be used by the player character at any time.

TC 3.1.3.7: Player Stomps Enemy

This test case evaluates FR37 by manually testing the player's functional ability to successfully attack an enemy with stomping action. The attack is considered successful when the player is able to inflict damage upon the enemy within the range of the attack. The testing environment puts the player at a position that ensures the player can successfully and safely attack an enemy with a stomp action. A safe attack is one where the player doesn't lose any health points. In this testing cases, a successful attack is considered where the enemy object is destroyed and removed from the game because the player killed the enemy with the stomping action. The input of the testing case is a keyLeft that will allow the player to be directly above the enemy and land directly on the enemy to stomp the enemy. The output is the enemy object being destroyed and is no longer displayed on the screen. The test case result is a pass since the player took a step the left and exercised the stomping attack on the enemy within range and killed the enemy. The enemy was observed to be destroyed because it is no longer displayed on the screen and the player's health points remained the same before and after the attack.

TC 3.1.3.8: Player Releases Rope

This test case evaluates FR24 by manually testing the player's functional ability to stop climbing the rope and return to the game path. The test environment positions the player at a climbing state on an already existing rope object in the game path with ground blocks surrounding that positions that the player can return to. The input of the test is a key down press of keyJump and the output is the player changing positions from being on the rope to a nearby ground block along with the character sprite changing to standing which indicates the player is no longer interacting with the rope object. The test result is a pass since the screen displayed the character at the ground with a standing sprite after the keyJump stroke.

TC 3.1.3.9: Player Grabs Rope

This test case evaluates FR22 by manually testing the player's functional ability to grab and get on the rope to climb when pressing keyUp. The test environments positions the player at a standing state within range of an existing rope object on the game path. The test input is a key down press of keyUp and the output is a player being at the same position as the rope object and changing to a climbing sprite to indicate player is interacting with rope object. The test result is a pass since the screen displayed the character on the rope object with a climbing sprite after the keyUp pressed.

TC 3.1.3.10: Player Climbs Up

This test case evaluates FR23 by manually testing the player's functional ability to move up the rope in a climbing state when keyUp is pressed. The test environment positions the player on the rope with a climbing state to indicate interaction between rope object and player. The input of the test case is a key down press of keyUp and the output is a player increasing y-position by playerClimbSpeed while the player-rope interaction is maintained. The result of the test case is a pass since the player was observed to increase in y-position while remaining in a climbing state with the rope object.

TC 3.1.3.11: Player Climbs Down

This test case evaluates FR23 by manually testing the player's functional ability to move down the rope in a climbing state when keyDown is pressed. The test environment positions the player on the rope with a climbing state to indicate interaction between rope object and player. The player position on the rope should have at least playerClimbSpeed distance in the negative y-direction. The input of the test case is a key down press of keyDown and the output is a player decreasing y-position by playerClimbSpeed while the player-rope interaction is maintained. The result of the test case is a pass since the player was observed to decrease in y-position while remaining in a climbing state with the rope object.

TC 3.1.3.12: Player Throws Object

This test case evaluated FR10, 18 by manually testing the player's functional ability to throw objects when pressing keyThrow. The test environment positions the player at a stationary state surrounded by open ground space in both x-directions and positions a throwable object in the player's hands. The input of the test is a key down press of keyThrow and the output is animation of the thrown object being at $speed > 0$ and player having empty hands. The result of the test case is a pass since it was observed that player no longer has the object in their hands and the throwable object is at a different position than the player by throwSpeed.

TC 3.1.3.13: Player Picks Up Object

This test case evaluates FR17 by manually testing the player's functional ability to pick up objects when pressing keyPick. The test environment positions the player at a stationary state with empty hands at the same position as an existing collectable object. The input of the test is a key down press of keyThrow and the output is animation of the thrown object being at $speed > 0$ and player having empty hands. The result of the test case is a pass since it was observed that player no longer has the object in their hands and the throwable object is at a different position than the player by throwSpeed.

TC 3.1.3.14: Player Use Bomb

This test case evaluates FR9 by manually testing the player's functional ability to place a bomb on ground while advancing in the game's path when keyBomb is pressed. The test environment positions the player in standing stationary state at open space area. The input of the test is a key down press of keyBomb and the output is the system's response by creating a new bomb object positioned at the same position as when the player pressed the key control input. The result of the test case is a pass since the player was able to successfully place a bomb at its position upon pressing keyBomb.

TC 3.1.3.15: Player Use Weapon

This test case evaluates FR12 by manually testing the player's functional ability to use a weapon object at any point needed in the game by pressing keyweap. The test environment positions the player at stationary standing state at an open space. The test input is a key down press of keyweap and the output is the system response in creating a weapon object to be used by the player and displaying an animation of the weapon in the player's hand being used. The test result is a pass since the player was observed with a weapon in their hands that is animated to extend to form an attack mechanism for the player.

TC 3.1.3.16: Player Use Weapon for Period of Time and Can't use anything else

This test case evaluates FR27 by manually testing the player's functional ability to only use a weapon object at any point needed in the game by pressing keyweap and not being able to utilize his hands to use any other interactive game objects. The test environment positions the player at stationary standing state with weapon being at use because keyweap is pressed. Within range the player's range of interaction there is an interactive object, such as Box.The test input is a key down presses of keyweap and keyPick and the output is the system response of creating a weapon object in the player's hands and displaying an animation of only the weapon in use. The test result is a pass since the player was observed with only a weapon in their hands and the interactive object remained on the ground. This test case was exercised

on all interactive objects to guarantee the FR was completely satisfied by the system. The result of each iteration was a pass.

TC 3.1.3.17: Player Hits and Damages Snake Enemy

This test case evaluates FR25 by manually testing the player's functional ability to damage a snake upon using a weapon to hit the snake. The testing environment positions the player at stationary standing position within range of weaponAttack to a snake. The input of the test is a key down press of keyweap and the output is the enemy's health reduced by weaponDmg. For testing purposes, the reduction of health will be met zeroing the enemy's health to observe the enemy disappearing from the game to display the hit it took from the weapon attack. The test result is a pass since the player was able to use the weapon with keyweap press and causing the snake to no longer exist in the game because there the snake health decreased by weaponDmg.

TC 3.1.3.18: Player Hits and Damages Spider Enemy

This test case evaluates FR25, 26 by manually testing the player's functional ability to damage a spider upon using a weapon to hit the spider. The testing environment positions the player at stationary standing position within range of weaponAttack to a spider. The input of the test is a key down press of keyweap and the output is the enemy's health reduced by weaponDmg. For testing purposes, the reduction of health will be met by zeroing the enemy's health to observe the enemy disappearing from the game to display the hit it took from the weapon attack. The test result is a pass since the player was able to use the weapon with keyweap press and causing the spider to no longer exist in the game because there the spider health decreased by weaponDmg.

TC 3.1.3.19: Player Use Rope

The test case evaluates FR10 by manually testing the player's functional ability to place a rope at open space in the game when pressing keyRope. The testing environment positions the player at stationary standing state with open area space in the upward and side directions. The input of the test is a key down press of keyRope and the output is system response in animating a rope build upward and one x-position from the facing player

direction. The test result is a pass since upon pressing keyRope the system created an new rope object and it was displayed onto the screen at the correct position and length.

TC 3.1.3.20: Player Falls

The test case evaluates FR28 by manually testing the player's falling behaviour when the player moves to a position with no ground underneath. The testing environment positions the player at stationary standing state on the ground with a step away from a no ground space. The testing input is a key down press of keyLeft or keyRight that causes player to move to the no ground area and the output of the test case is having a $speed < 0$ because the player is changing position in the negative direction. The test result is a pass since the player was displayed in a falling state when it stepped into the no ground area and the speed of the player was noted to be fallSpeed.

TC 3.1.3.21: Player Lands Back on the Ground

The test case evaluates FR28 by manually testing the player's behaviour to land back on the ground after falling from a safe distance that is less than fallDmgDist to reach a ground position. The testing environment positions the player at stationary standing state on the ground with a step away from a no ground space with a ground block less than fallDmgDist distance away. The testing input is a key down press of keyLeft or keyRight that causes player to move to the no ground area and the output of the test case is the player changing position to the closest ground block that is underneath the falling starting position and leaving the player with $speed = 0$. The testing result is a pass since the player was observed to fall safely to the ground underneath and land on the ground with $speed = 0$ and retaining its full functionality.

TC 3.1.3.22: Player Can't move past blocks

This test case evaluates FR28 by manually testing the player's behaviour to not being able to move past blocking wall blocks when a key is pressed to move left, right, and up. The testing environment positions the player at stationary standing state with blocking blocks from all directions. The test input is a key down keyMove, such as keyLeft/keyRight/keyJump, and the

output is the player not being able to change position to a point that surpass the block with just that input and playerSpeed remains the same. The test result is a pass because it was observed that the player was blocked from advancing to a point pass the block and instead remaining at the same position and state. This test case was exercised on all move key input controls. The system passed all the test case input variation proving the completeness of FR28 being satisfied.

TC 3.1.3.23: Player Can't move outside the map

This test case evaluates FR28 by manually testing the player's behaviour to not being able to move past map boundaries when a key is pressed to move left, right, down, and up. The testing environment positions the player at stationary standing state with map boundaries from all directions. The test input is a key down keyMove, such as keyLeft/keyRight/keyJump or going through a safe fall, and the output is the player not being able to change position to a point that surpass the map boundaries and playerSpeed remains the same. The test result is a pass because it was observed that the player was blocked from advancing to a point pass the boundaries and instead remaining at the same position and state. This test case was exercised on all move key input controls. The system passed all the test case input variation proving the completeness of FR28 being satisfied.

TC 3.1.3.24: Player Crouches

This test case evaluates FR11 by manually testing the player's functional ability to crouch to change state when pressing keyCrouch. The testing environment position player at stationary standing state. The input of the test is a key down press of keyCrouch and the output is player state changing to crouching state and displaying the animation on the screen

TC 3.1.3.25: Player Health Reduced When Falling From a Peak

This test case evaluates FR30 by manually testing the player's behaviour after falling a $distance \geq fallDmgDist$. The testing environment positions the player at stationary standing state on the ground with a step away from a no ground space with a ground block fallDmgDist distance away or higher.

The testing input is a key down press of keyLeft or keyRight that causes player to move to the no ground area and the output of the test case is the player changing position to the closest ground block that is underneath the falling starting position and leaving the player with $speed = 0$ and decreased health points by fallDmg. The result of the test is a pass since the player health was displayed to be reduced by fallDmg after landing on the ground block that is a falling $distance \geq fallDmgDist$ down.

TC 3.1.3.26: Player Health Reduced When Hit by Snake Enemy

This test case evaluates FR36 by manually testing the player's state after being attacked by a snake enemy due to system response of detecting a player near the snake. The testing environment positions a player at a stationary state within attack range of a snake and allows the snake to attack only once to see the effect per attack. The test case has no inputs because it tests the system builtin effect on the player. The test output is the player's health being reduced by snakeDmg. The test result is a pass since the health points of the player was displayed to be reduced by snakeDmg.

TC 3.1.3.27: Player Health Reduced When Hit by Spider Enemy

This test case evaluates FR36 by manually testing the player's state after being attacked by a spider enemy due to system response of detecting a player near the spider. The testing environment positions a player at a stationary state within attack range of a spider and allows the spider to attack only once to see the effect per attack. The test case has no inputs because it tests the system builtin effect on the player. The test output is the player's health being reduced by spiderDmg. The test result is a pass since the health points of the player was displayed to be reduced by spiderDmg.

TC 3.1.3.28: Player Health Reduced When Hit by Arrow

This test case evaluates FR71 by manually testing the player's state after being attacked by a arrow shot from an arrow trap due to system response of

detecting a player near the arrow trap. The testing environment positions a player at a stationary state within attack range of an arrowTrap and allows the arrow to hurt only once to see the effect per attack. The test case has no inputs because it tests the system builtin effect on the player. The test output is the player's health being reduced by arrowDmg. The test result is a pass since the health points of the player was displayed to be reduced by arrowDmg.

TC 3.1.3.29: Player Health Reduced When Overlap with Spike

This test case evaluates FR73 by manually testing the player's state after passing by a spike trap due to system response of detecting a player on the trap. The testing environment positions a player at a stationary state on the spike and the spike hurts only once to see the isolated effect of the trap. The test case has no inputs because it tests the system builtin effect on the player. The test output is the player's health being reduced by spikeDmg. The test result is a pass since the health points of the player was displayed to be reduced by spikeDmg.

TC 3.1.3.30: Player Health Reduced When hit by bomb Explosion

This test case evaluates FR16 by manually testing the player's state after undergoing a bomb explosion due to system behaviour of bomb explosion after bomb tick time. The testing environment positions a player of a stationary state at bombing area that will have an explosion in bomb tick time. The test case has no inputs because it tests the system builtin effect on the player. The test output is the player's health being reduced by bombDmg. The test result is a pass since the health points of the player was displayed to be reduced by bombDmg.

TC 3.1.3.31: Player Killed When Health Reaches Zero

This test case evaluates FR74 by manually testing the player's state after reaching zero health points due to unsuccessful game run. The testing environment positions the player in an area of constant damage to its health

points which guarantees health points will be zero eventually. The test case has no inputs becuase it tests the system builtin effect on the player. The test output is player object being deleted and the player character is no longer displayed. The test result is a pass because the game state shows no record of a Mover object and the game displays the game over screen with player is dead message.

TC 3.1.3.32: Player Dodges a Moving Arrow Attack

This test case evaluates FR7 by manually testing the player's state after dodging a moving arrow. The goal of this test is to show the range of mobility available to the player. The testing environment position the player stationary with an arrow attack that is about to collide with and open space surrounding the player. The test input is a key down press of keyJump and the output is the player moving out of the arrow range attack after the arrow not being able to change direction. The test result is a pass since the player showed no health points reduction as it jumped away and the arrow passed without colliding with the player.

TC 3.1.3.33: Player Start the game with heartStartAmount heart, bombStartAmount bombs, and ropeStartAmoundt ropes

This test case evaluates FR13 by manually testing the player's state at the start of the game. The testing environment is starting the game file in a stress testing manner. The test input is the command line execution of running the game file and the output is the player has heartStartAmount hearts, bombStartAmount bombs, and ropeStartAmount ropes. The test result is a pass since the game displayed the same player state information on the screen.

TC 3.1.4.1: Bomb placed

This test case evaluates FR9 by manually testing if a bomb created by player has the correct coordinates. The input of this test is makeEnt() and the output is the creation and display of a bomb with the same coordinates as

player. The result of the test case is a pass since a bomb is created at player's position.

TC 3.1.4.2: Bomb time set

This test case evaluates FR16 by manually testing if a bomb has the correct time. The input of this test is makeEnt() and the output is the time remaining on the bomb. The result of the test case is a pass since the time of a new bomb is bombTime.

TC 3.1.4.3: Bomb explosion time

This test case evaluates FR16 by manually testing if a bomb explodes after the correct amount of time. The input of this test is tick() and the output is the bomb exploding. The result of the test case is a pass since the bomb explodes after bombTime seconds.

TC 3.1.4.4: Bomb explosion size

This test case evaluates FR16 by manually testing if a bomb explosion is the corrrect size. The input of this test is explode() and the output is the explosion size. The result of the test case is a pass since the explosion is bombSize blocks large.

TC 3.1.4.5: Bomb explosion destruction

This test case evaluates FR16 by manually testing if a bomb explosion destroys all overlapping blocks are destroyed and player and Enemy objects are dealt bombDamage damage. The input of this test is explode() and the output is the removal of affected blocks/player/Enemy objects on the screen. The result of the test case is a pass since the explosion is destroys all overlapping blocks, player, and Enemy objects..

TC 3.1.4.6: Bomb pickup

This test case evaluates FR17 by manually testing if a bomb can be picked up by player when when pressing keyPick. The input of this test is key down press of keyPick and the output is the movement of the bomb to player's

hand. The result of the test case is a pass since player appears to be holding the bomb.

TC 3.1.4.7: Bomb throw

This test case evaluates FR18 by manually testing if a bomb can thrown when when pressing keyThrow. The input of this test is key down press of keyThrow and the output is the movement of the bomb out of player's hand at a speed of throwSpeed pixels per second. The result of the test case is a pass since the bomb leave's player's hand at throwSpeed pixels per second.

TC 3.1.4.8: Bomb throw explosion

This test case evaluates FR19 by manually testing if a thrown bomb explodes upon colliding with a solid block. The input of this test is move() and the output is the bomb hitting a solid block and exploding. The result of the test case is a pass since the explodes upon hitting a solid block.

TC 3.1.5.1: Rope throw

This test case evaluates FR10 by manually testing if a rope created by player has the correct coordinates. The input of this test is makeEnt() and the output is the creation and display of a rope with the same coordinates as player. The result of the test case is a pass since a rope is created at player's position.

TC 3.1.5.2: Rope throw speed

This test case evaluates FR10 by manually testing if a thrown rope moves at the correct speed. The input of this test is throw() and the output is speed of the rope. The result of the test case is a pass since the rope's speed is throwSpeed.

TC 3.1.5.3: Rope max height

This test case evaluates FR20 by manually testing if a thrown rope stops at the correct coordinates after reaching the max height. The input of this test

is move() and the output is the coordinates of the rope. The result of the test case is a pass since the rope stops at the middle centre of a block.

TC 3.1.5.4: Rope attachment in block

This test case evaluates FR20 by manually testing if a thrown rope stops at the correct coordinates after hitting a block. The input of this test is move() and the output is the coordinates of the rope. The result of the test case is a pass since the rope stops at the middle centre of a block.

TC 3.1.5.5: Rope extension

This test case evaluates FR21 by manually testing if a stopped rope extends down the correct length. The input of this test is makeRope() and the output is the coordinates of the bottom of the rope. The result of the test case is a pass since the rope stops after ropeLength blocks or before a solid block.

TC 3.1.5.6: Rope crouch throw

This test case evaluates FR11 by manually testing if a rope created by player when crouching has the correct coordinates. The input of this test is makeEnt() and the output is the creation and display of a rope with the same coordinates as player. The result of the test case is a pass since a rope is created at player's position.

TC 3.1.6.1: Weapon swing

This test case evaluates FR25 by manually testing if a weapon created by player has the correct coordinates. The input of this test is makeEnt() and the output is the creation and display of a weapon with the same coordinates as player. The result of the test case is a pass since a weapon is created at player's position.

TC 3.1.6.2: Weapon damage

This test case evaluates FR26 by manually testing if a weapon deals the correct damages. The input of this test is tick() and the output is the amount

of damage dealt to a Snake object. The result of the test case is a pass since the snake takes WeaponDmg damage.

TC 3.1.6.3: Weapon delay

This test case evaluates FR27 by manually testing if a weapon can be used again after the correct amount of time. The input of this test is swing() and the output is the amount of time until the use. The result of the test case is a pass since the weapon cannot be used until after weaponDelay seconds.

TC 3.1.7.1: Gold bar pickup

This test case evaluates FR32, 48 by manually testing if collecting a Gold object adds the correct amount of gold. The input of this test is use() and the output is player's gold. The result of the test case is a pass since player's gold increases by valGold.

TC 3.1.7.2: Ruby pickup

This test case evaluates FR32, 45 by manually testing if collecting a Ruby object adds the correct amount of gold. The input of this test is use() and the output is player's gold. The result of the test case is a pass since player's gold increases by valRuby.

TC 3.1.7.3: Emerald bar pickup

This test case evaluates FR32, 47 by manually testing if collecting an Emerald object adds the correct amount of gold. The input of this test is use() and the output is player's gold. The result of the test case is a pass since player's gold increases by valEmerald.

TC 3.1.7.4: Sapphire pickup

This test case evaluates FR32, 46 by manually testing if collecting a Sapphire object adds the correct amount of gold. The input of this test is use() and the output is player's gold. The result of the test case is a pass since player's gold increases by valSapphire.

TC 3.1.7.5: Diamond pickup

This test case evaluates FR32, 44 by manually testing if collecting a Diamond object adds the correct amount of gold. The input of this test is use() and the output is player's gold. The result of the test case is a pass since player's gold increases by valDiamond.

TC 3.1.7.6: Bomb pile pickup

This test case evaluates FR32, 88 by manually testing if collecting a BombPile object adds the correct number of bombs. The input of this test is use() and the output is player's bombs. The result of the test case is a pass since player's bombs increases by valBombPile.

TC 3.1.7.6: Rope pile pickup

This test case evaluates FR32, 89 by manually testing if collecting a RopePile object adds the correct number of ropes. The input of this test is use() and the output is player's ropes. The result of the test case is a pass since player's ropes increases by valRopePile.

TC 3.1.8.1: Idle sprite

This test case evaluates FR34 by manually testing if the correct sprite is used while idle. The input of this test is changeState() and the output is the player sprite on the screen. The result of the test case is a pass since player's sprite is idle.

TC 3.1.8.2: Falling sprite

This test case evaluates FR34 by manually testing if the correct sprite is used while falling. The input of this test is changeState() and the output is the player sprite on the screen. The result of the test case is a pass since player's sprite is falling.

TC 3.1.8.3: Climbing sprite

This test case evaluates FR34 by manually testing if the correct sprite is used while climbing. The input of this test is changeState() and the output is the

player sprite on the screen. The result of the test case is a pass since player's sprite is climbing.

TC 3.1.8.4: Crouching sprite

This test case evaluates FR34 by manually testing if the correct sprite is used while crouching. The input of this test is changeState() and the output is the player sprite on the screen. The result of the test case is a pass since player's sprite is crouching.

TC 3.1.9.1: Chest placement

This test case evaluates FR39 by manually testing if a chest is placed at the correct coordinates. The input of this test is makeEnt() and the output is the chest coordinates. The result of the test case is a pass since the chest has the correct coordinates specified in the map file.

TC 3.1.9.2: Chest open

This test case evaluates FR40, 41, 42, 43 by manually testing if a chest releases a possible treasure at the correct coordinates. The input of this test is use() and the output is a treasure on the screen. The result of the test case is a pass since the chest is replaced with a possible treasure with the same coordinates.

TC 3.1.9.3: Chest pickup

This test case evaluates FR91 by manually testing if a chest can be picked up by player when when pressing keyPick. The input of this test is key down press of keyPick and the output is the movement of the chest to player's hand. The result of the test case is a pass since player appears to be holding the chest.

TC 3.1.9.4: Chest throw

This test case evaluates FR91 by manually testing if a chest can thrown when when pressing keyThrow. The input of this test is key down press of keyThrow and the output is the movement of the chest out of player's hand

at a speed of throwSpeed pixels per second. The result of the test case is a pass since the chest leave's player's hand at throwSpeed pixels per second.

TC 3.1.10.1: Enemy map placement

This test case evaluates FR79 by manually testing if a snake or spider enemy generated on the map is consistent with the map file coordinates specifying such enemies. The input of this test is ReadMap and Mapcell that create that map and the output is motionless on the map and do not attack the player and reduce their health. The result of the test case is a pass since the since the map entities if generated in a solid block can not continue motion. As expected the entities generated are stuck in the block unable to reduce the player's health. Hence, the player's health remains unchanged as expected.

TC 3.1.10.2: Player-enemy Defense - Whip

This test case evaluates FR12 by manually testing if the enemy is damaged when a whip is latched onto the enemy. The input to the test is the lshift key to draw out the whip and the output is the player's health being affected by a certain weapomDmg. The result of the test is a pass since the player's health is reduced by certain weaponDmg amount.

TC 3.1.10.3: Player-enemy Defense - Stomp

This test case evaluates FR37 by manually testing if the enemy is killed when a player jumps into the enemy entity. The input to the test is ent.damage() and the output is the player's health reduced by a certain stompDmg. The test is a pass since the player health is reduced by the specified stomp damage.

TC 3.1.11.1: Snake creation

The test case evaluates FR50 and FR52 by unit testing if all the snake entities specified in the map file are present on the map at the start of the game. The input for the test is the map object created by MapObj and ReadMap and the ouput is the multiple snakes moving in a straight line around the map. They furthermore, turn at solid blocks. The result of the test case is a pass since the initial start positions for snake movement are in correlation with map coordinates in the map file for the snake entity.

TC 3.1.11.2: Snake moves towards the right and alters direction upon a solid block (moves left)

The test case evaluates FR51 by unit testing if the snake switches direction when the snake encounters a solid block. The input of the test is not required because the solid blocks are generated by the map file. The output is a snake starting with a right direction movement and switches to a left direction movement upon encountering a solid block. The result og the test case is a pass since we tested if the snake changes direction only when it encounters a solid block which it did as expected.

TC 3.1.11.3: Snake attacks a player on the way of its straightline path of movement

The test case evaluates FR54 by unit testing if the player's health is reduced when the player encounters a snake. The input of the test is non existent since the map file loads the snake entities. The output involves the player's initial health reducing by snakeDmg. The result of the test is a pass since the player's health reduces after being attacked by a snake by the specified damage amount.

TC 3.1.11.4: Snake attacks the player killing the player

The test case evaluates FR54 by unit testing if the player dies if their health can not withstand the snakeDmg amount. The input of the test is non existent since the map file loads the snake entities. The output of the test is game over as with a health of 0 the player dies. The result of the test is a pass since the game over message is displayed thus, ending the game.

TC 3.1.11.5: Player dodges snake by jumping over it

The test case evaluates FR15 by unit testing if the player is able to jump over the snake without taking any health damage.The input of the test is the upward arrow keyboard input. The output is the player's health remains the same before and after the jump. Since the player's health is unaffected and the player is able to be shown on screen jumping and dodging the snake, the test is a pass.

TC 3.1.12.1: Spider creation

The test case evaluates FR55 by unit testing if the spider entities are generated on the map at the start of the game. The input of the test is the Map Object used by MapObj and ReadMap. The output is the spider created at different points of the map at a dormant state. The map file and on screen location coordinates match and hence the result of the test is a pass.

TC 3.1.12.2: Spider jumps on the player if the player is within a range

The test case evaluates FR56 by unit testing if the player is within the SpideSense range the player is attacked by the spider as the spider jumps on them. The input of the test is non existent since the map file loads the spider entities. The output is the spider attacks the player and then cool offs and continues to attack the player if the player is still in range. The result of the test is a pass since the spider attacks the player within the SpideSense range before it cools off to be dormant.

TC 3.1.12.3: Spider attacks the player by jumping on them reducing their health

The test case evaluates FR62 by unit testing if the player's health reduces for the time player is within the SpideSense range. The input of the test is non existent since the map file loads the spider entities. The output is the player's health reduces by spiderDmg continuously till player moves away from the SpideSense range. The result of the test is a pass since the SpideSense range gives the attacking range and the player's health is reduced by spiderDmg accordingly if they stay within that range.

TC 3.1.12.4: The spider remains dormant during the cool off stage after attacking the player in range

The test case evaluates FR60 by unit testing if the Spider remains dormant if the player is out of range and checks if the spider does not attack the player in such a case. The input of the test is non existent since the map file loads the spider entities. The output is the spider continues to be the original number of blocks away from the player since it is dormant. The result of the

test is a pass since the player's health is not affected by a spider attack as the spider remains dormant.

TC 3.1.12.5: Spider attacks the player killing the player

The test case evaluates FR62 if the player is killed when a spider attacks them. This happens when the player's health is not sufficient to withstand a spider attack. The input of the test is non existent since the map file loads the spider entities. The output is the player's health is 0. The test result is a pass since the player's health is 0 and the game over message is displayed.

TC 3.1.12.6: Spider continuously attacks the player killing the player

The test case evaluates FR62 if the player is killed when a spider attacks them. This happens when the player's health is continuously reduces as the player does not move out of the range of the spider. The input of the test is non existent since the map file loads the spider entities. The output is the player's health is 0. The test result is a pass since the player's health is 0 and the game over message is displayed.

TC 3.1.13.1: Arrow Trap Box Generated With The Game Starting

This test case evaluates FR63 by manually testing the game state of generating arrow trap object. The testing environment initializes a position for arrowTrap object in the map file that ensures displaying visibility. The input to the test case is the map file to the game file being initialized on the command line. The output is arrowTrap object appearing at the same specified position in the map file on the game screen map. The test result is a pass since the arrowTrap object was observed at the correct position indicating that it was initialized as a trap in the game state information.

TC 3.1.13.2: Arrow Trap Box is placed on a block.

This test case evaluates FR64 by manually testing the appearance of the arrowTrap object placement on the game screen. The testing environment

initializes a position for arrowTrap object in the map file that ensures displaying visibility. The input to the test case is the map file to the game file being initialized on the command line. The output is arrowTrap object appearing at the same specified position in the map file on the game screen map and above a ground block. The test result is a pass since the arrowTrap object was observed at a correct position above the ground block.

TC 3.1.13.3: Arrow Trap Box Detects Player Approach from Left.

This test case evaluates FR65,66,67 by manually testing the arrow trap state in the game when a player is within range of trap attack. The testing environment positions the player walking into the range of trap attack from the left. The test has no input because it tests the system response to a player movement. The test output is arrow object being created as a system response. The test result is a pass since the arrow object was created and displayed on the arrowTrap object.

TC 3.1.13.4: Arrow Trap Box Detects Player Approach from Right.

This test case evaluates FR65,66,67 by manually testing the arrow trap state in the game when a player is within range of trap attack. The testing environment positions the player walking into the range of trap attack from the right. The test has no input because it tests the system response to a player movement. The test output is arrow object being created as a system response. The test result is a pass since the arrow object was created and displayed on the arrowTrap object.

TC 3.1.13.5: Arrow Trap Box Cant Shoot Arrow When there is an arrow in action shot by the same box.

This test case evaluates FR67 by manually testing the arrow trap state in the game after an arrow was shot and the player stays within attack range during the arrow movement. The testing environment position the player with trap attack range and starts testing after the arrow moves. The test has no input because it tests the system behaviour of the activated arrow

trap. The output is the lack of new arrows being created because the first one is still in action. The test result is a pass since only one arrow was shot as long as that arrow staying in action.

TC 3.1.13.6: Arrow Trap Box Shoots Arrow Toward the Player from Left.

This test case evaluates FR65,66 by manually testing the arrow trap state in the game of shooting an arrow object after detection of a player from left. The testing environment positions player stationary to the left of the trap and within attack range. The test has no input because it tests the system behaviour of the activated arrow trap. The output is an arrow moving to the left of the trap toward where the player was detected. The test result is a pass since the game displayed animation of the arrow moving in the left direction.

TC 3.1.13.7: Arrow Trap Box Shoots Arrow Toward the Player from Right.

This test case evaluates FR65,66 by manually testing the arrow trap state in the game of shooting an arrow object after detection of a player from right. The testing environment positions player stationary to the right of the trap and within attack range. The test has no input because it tests the system behaviour of the activated arrow trap. The output is an arrow moving to the right of the trap toward where the player was detected. The test result is a pass since the game displayed animation of the arrow moving in the right direction.

TC 3.1.13.8: Arrow Hits the Static Player.

This test case evaluates FR71 by manually testing the arrow state when colliding with a static player. The testing environment positions player stationary within attack range and arrow is moving towards player. The test has no input because it tests the system behaviour of the activated arrow attack. The output is reduction in the player's health by arrowDmg. The test result is a pass since the player's health was displayed with a reduction of arrowDmg.

TC 3.1.13.9: Arrow Hit the Moving Player.

This test case evaluates FR71 by manually testing the arrow state when colliding with a moving player. The testing environment positions player walking within attack range and arrow is moving towards player. The test has no input because it tests the system behaviour of the activated arrow attack. The output is reduction in the player's health by arrowDmg. The test result is a pass since the player's health was displayed with a reduction of arrowDmg.

TC 3.1.13.10: Arrow fly in a straight line unaffected by gravity

This test case evaluates FR69 by manually testing the arrow state after being shot towards where the player was detected. The testing environment simply executes an arrow shoot without the player to test the flying animation of the arrow. The test input is called the function of arrow shoot() and the output is arrow having no change in y-position as the x-position changes. The test result is a pass since the arrow moves at the same y-position of when it was shot from the trap.

TC 3.1.13.11: Arrow fly stops when encountering an object in the game

This test case evaluates FR70 by manually testing the arrow state when approaching collision with a game object. The testing environment places a game object such as wall block in the path of the shooting arrow. The test has no input because it tests the system behaviour of the activated arrow attack. The output is the arrow stops moving and reaches a stationary state with $speed = 0$. The test result is a pass since it was observed that the arrow stopped moving at the point of collision with the object.

TC 3.1.13.12: Arrow self destructs when encountering an object in the game

This test case evaluates FR70 by manually testing the arrow state at point of collision with a game object. The testing environment places a game object such as wall block in the path of the shooting arrow. The test has no

input because it tests the system behaviour of the activated arrow attack. The output is the arrow object no longer exists in the game state information because it was deleted. The test result is a pass since it was observed that the arrow was removed from the game because the screen was updated without the object's image.

TC 3.1.13.13: Arrow pickup

This test case evaluates FR90 by manually testing if an arrow can be picked up by player when when pressing keyPick. The input of this test is key down press of keyPick and the output is the movement of the arrow to player's hand. The result of the test case is a pass since player appears to be holding the arrow.

TC 3.1.13.14: Arrow throw

This test case evaluates FR91 by manually testing if an arrow can thrown when when pressing keyThrow. The input of this test is key down press of keyThrow and the output is the movement of the arrow out of player's hand at a speed of throwSpeed pixels per second. The result of the test case is a pass since the arrow leave's player's hand at throwSpeed pixels per second.

TC 3.1.14.1: Spike Trap Generated With The Game Starting.

This test case evaluates FR63 by manually testing the game state of generating a new spike trap object. The testing environment initializes a position for spike object in the map file that ensures displaying visibility. The input to the test case is the map file to the game file being initialized on the command line. The output is new spike trap object appearing at the same specified position in the map file on the game screen map. The test result is a pass since the new spike object was observed at the correct position indicating that it was initialized as a trap in the game state information.

TC 3.1.14.2: Spike Trap is placed on a block

This test case evaluates FR72 by manually testing the appearance of the spike trap object placement on the game screen. The testing environment

initializes a position for spike object in the map file that ensure displaying visibility. The input to the test case is the map file to the game file being initialized on the command line. The output is spike object appearing at the specified position in the map file on the game screen map and above a ground block. The test result is a pass since the spike object was observed at a correct position on top of the ground block.

TC 3.1.14.3: Spike Damages Player

This test case evaluates FR73 by manually testing the spike effect upon player stepping on the trap. The testing environment position a moving player to pass by the spike trap and allows the trap to inflict damages only once per pass. The test has no input because it tests the spike attack behaviour. The output is spike reducing the player's heath point by spikeDmg. The test result is a pass since it the player's health points was observed decreased by spikeDmg.

TC 3.1.15.1: Exit door

This test case evaluates FR75 by manually testing if reaching the exit door ends the game. The input of this test is move() and the output is the game terminating. The result of the test case is a pass since game terminates.

TC 3.1.15.2: Score save

This test case evaluates FR75,76 by manually testing if reaching the score is saved. The input of this test is gameover() and the output is a file. The result of the test case is a pass since the score is written to the file.

TC 3.1.15.3: Score display

This test case evaluates FR77 by manually testing if reaching the score file can be read by the user. The input of this test is score file and the output is the score. The result of the test case is a pass since the user can read the score from the score file.

TC 3.1.16.1: Level creation

This test case evaluates FR78,79 by manually testing if the placement of all blocks, doors, and entities are correct. The input of this test is ReadMap() and the output is the game map. The result of the test case is a pass since all blocks, doors, and entities are placed correctly.

TC 3.1.16.2: Player starting location

This test case evaluates FR80 by manually testing if the player starts at the correct location. The input of this test is Mover() and the output is the player's coordinates. The result of the test case is a pass since the player's coordinates are the same as the ones specified in the map file.

TC 3.1.16.3: Clear path

This test case evaluates FR81 by manually testing if the map has a clear path from the entrance to the exit. The input of this test is the map file and the output is the path. The result of the test case is a pass since there is a sequence of empty blocks from the entrance door to the exit door.

TC 3.1.16.4: Block types

This test case evaluates FR82 by manually testing if all blocks are correct according to the map. The input of this test is ReadMap() and the output is the game map. The result of the test case is a pass since the types of all blocks are correct.

TC 3.1.17.1: Sign placement

This test case evaluates FR85 by manually testing if signs are placed correctly. The input of this test is ReadMap() and the output is the sign locations. The result of the test case is a pass since the signs are placed correctly.

TC 3.1.17.2: Sign text

This test case evaluates FR86 by manually testing if sign messages are displayed correctly. The input of this test is sign() and the output is the message. The result of the test case is a pass since the message is displayed.

TC 3.1.17.3: Sign reading

This test case evaluates FR87 by manually testing if sign messages are displayed at the correct location. The input of this test is sign() and the output is the message location. The result of the test case is a pass since the message is displayed at the bottom of the screen.

2 Nonfunctional Requirements Evaluation

2.1 Usability

TC 3.2.1.1: Game feeling

This test case evaluates NFR1 by evaluating user's game feeling. Users were given 10 minutes with the game and then asked about how much it felt like Spelunky. The test result is a pass as over 80% of users felt like they were playing Spelunky.

TC 3.2.1.2: Immersion

This test case evaluates NFR2 by evaluating user's immersion. Users were given 10 minutes with the game and then asked about how much they felt like the player character. The test result is a pass as over 80% of users felt like the player character.

TC 3.2.1.3: Control feeling

This test case evaluates NFR3, 9 by evaluating user's control feeling. Users were allowed to play the game and then asked about how responsive the controls felt. The test result is a pass as all inputs made by users immediately moved the player.

TC 3.2.1.4: Game understanding

This test case evaluates NFR4, 5 by evaluating user's game understanding. Users were given 10 minutes with the game and were asked if they needed help. The test result is a pass as over 80% of users didn't need outside help.

TC 3.2.1.5: Game progress

This test case evaluates NFR6 by evaluating user's game progress. Users were allowed to play the game and then asked about how well they understood their game progress. The test result is a pass as users read signs detailing their progress.

TC 3.2.1.6: Game text

This test case evaluates NFR7 by evaluating the game text. Users were allowed to play the game and then asked about how well they could read the text. The test result is a pass as users could read the text easily.

2.2 Stress Testing

TC 3.2.2.1: Large maps

This test case evaluates NFR10 by evaluating the game frame rate with a large map. The test result is a pass as the system has a consistent frame rate of at least 30 fps with a 100 by 100 block map.

TC 3.2.2.2: Large number of entities

This test case evaluates NFR10 by evaluating the game frame rate with a large number of entities. The test result is a pass as the system has a consistent frame rate of at least 30 fps with 100 entities.

2.3 Performance

TC 3.2.3.1: Game response time

This test case evaluates NFR8 by evaluating the game response time. The test result is a pass as the game responds to user inputs in at least 10ms.

TC 3.2.3.2: Game frame rate

This test case evaluates NFR10 by evaluating the game frame rate. The test result is a pass as the system has a consistent frame rate of at least 30 fps.

TC 3.2.3.3: Game storage

This test case evaluates NFR11 by evaluating the game storage. The test result is a pass as the entire systems takes up less than 16GB.

TC 3.2.3.4: Missing/incomplete file Exception handling

This test case evaluates NFR25 by evaluating error handling for missing/incomplete files. The test result is a pass as the system terminates with a missing/incomplete file error.

3 Comparison to Existing Implementation

The existing implementation has no test report or test cases documentation. However, some of the project's test cases can be applied on the existing implementation and a test report can be generated. The existing system functionalities include and when test cases exercised on the result is:

1. Player movement: The player can jump and move right and left using keyboard inputs. Damage is taken for falling too far. Passes 3.1.3.1-3.1.3.5 & 3.1.3.20-3.1.3.23 & 3.1.3.25 & 3.1.3.31 and fails all other test cases

2. Map entities: Entities such as treasure, signs, map solid blocks and block spaces are created as part of the game map as soon as the game is started. Passes 3.1.9

3. Chests: Chests can be interacted with and creates gold bars or rubies that are collected automatically. Passes 3.1.9

4. Display: The camera is focused on the player. Health and gold are displayed. Passes 3.1.2.1-3.1.2.2 & 3.1.2.5-3.1.2.7 and fails all other test cases

5. Sign: Sign messages are displayed at the bottom of the screen when the player is near them. Passes 3.1.17

4 Unit Testing

Unit Testing was executed manually by running the game and following the test cases as can be seen in section 1. Unit testing was performed bottom-up. The individual parts of modules will be tested first and then the parts that require other modules. Since the modules are complete, there is no need for stubs or drivers.

5 Changes Due to Testing

- Due to time constraints and the long length of the test report testing for coverage metrics was not exercised.
- Due to time constraints and the long length of the test report implementing and testing the executable file was not exercised.
- Due to the detailed unit testing breakdown the system requirements were traced in a clear manner showcasing the complete functionality of the game is correct and requires no changes.

6 Automated Testing

The project will not be using automated testing approach in the testing phase of the project. The testing team decided to use manual testing approach. The reason behind this decision is because the project involves the redevelopment of game the main focus of testing will be to provide a fully functional and enjoyable gaming experience. The testing will showcase the importance of validating user interactions with the system and presenting an easy to use implementation of the original game that is more challenging. Another reason for not using automated testing is that the development team dont have access to the appropriate tools required to executes tests in this approach therefore using manual testing approach increase the testing team confidence in executing test correctly and efficiently

7 Trace to Requirements

Test Cases Results	**Req.**
3.1.1.*	FR1-2
3.1.2.*	FR3-5, 74
3.1.3.*	FR7-16, 18, 22-30, 36-37, 71, 73-74
3.1.4.*	FR9, 16-19
3.1.5.*	FR10-11, 20-21
3.1.6.*	FR25-27
3.1.7.*	FR32, 44-48, 88-89
3.1.8.*	FR34
3.1.9.*	FR39-43, 91
3.1.10.*	FR12, 37, 79
3.1.11.*	FR15, 50-52, 54
3.1.12.*	FR55-56, 60, 62
3.1.13.*	FR63-67, 69-71, 90
3.1.14.*	FR63, 72-73
3.1.15.*	FR75-77
3.1.16.*	FR78-82
3.1.17.*	FR85-87
3.2.1.*	NFR1-7, 9
3.2.2.*	NFR10
3.2.3.*	NFR8, 10-11, 25

Table 2: Trace Between Requirements and Modules

8 Trace to Modules

Test Cases Results	**Modules**
3.1.1.*	M1
3.1.2.*	M2
3.1.3.*	M3, M4, M7
3.1.4.*	M4.12, 16
3.1.5.*	M4.13
3.1.6.*	M4.25-26
3.1.7.*	M4.2-10
3.1.8.*	M3
3.1.9.*	M4.14-15
3.1.10.*	M6
3.1.11.*	M4.19
3.1.12.*	M4.20
3.1.13.*	M4.22-23
3.1.14.*	M4.24
3.1.15.*	M1
3.1.16.*	M6
3.1.17.*	M4.11
3.2.1.*	All
3.2.2.*	All
3.2.3.*	All

Table 3: Trace Between Requirements and Modules

9 Code Coverage Metrics

The team did not not use any form of code coverage software since they decided to do all testing manually and given the time constraints there was not enough time to explore automated testing tools. However, during the development process the team modified the code to remove most of the redundant code and non-reachable. Hence, this improved the code coverage of the project and the metric used was simple peer review and peer programming techniques. In future revisions of the project, the team will look into

using automated coverage testing tools such as coverage.py that provides more formal testing metrics and reports for the program.