SE 3XA3: Test Plan
Zombie Survival Kit

Team #6, Group 6ix
Mohammad Hussain hussam17
Brian Jonatan jonatans
Shivaansh Prasann prasanns

December 4, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
2018-10-26	1.0	First Revision of Test Plan
2018-11-28	2.0	Updated Final Revision of Test Plan

1 General Information

1.1 Purpose

This document is a description of the testing, validation and verification procedures that are to be implemented for Group 6's SFWR ENG 3XA3 project titled 'Zombie Survival Kit'. These test cases were conceived before the majority of the implementation and are therefore to be used by the project team for future references during implementation, testing and maintenance.

1.2 Scope

The project titled 'Zombie Survival Kit" is a game development kit to be used by aspiring game developers to use as a starting ground for their projects. As such, the scope of testing shall theoretically cover various game mechanics and functionalities, control schemes and performance.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Definitions**

Term	**Definition**
Player	The in-game persona of the user
User	The person using Zombie Survival Kit
Pickable Object	An in-game object that the player can pick, drop, use and store in the Inventory
Inventory	An in-game storage option used to store various objects

1.4 Overview of Document

This document comprises of details regarding the test initiatives, test procedures, tools and techniques to be used, a testing schedule and facts which together serve as a reference for the test team to successfully test the software being developed.

2 Plan

2.1 Software Description

Zombie Survival Kit is a starting ground for aspiring game developers to help them create their own Zombie Survival First Person Shooter game, and has been built using the Unity 3D engine and C#. Testing would involve verifying that the various mechanics and functionalities in place are working correctly and reliably.

2.2 Test Team

The test team comprises of the three members of Group 6, as well as all volunteers who help test the product via feedback provided in a Google Form after trying a demo of the product.

2.3 Automated Testing Approach

The Unity3D engine shall play an instrumental role in the automated testing process. Many features of the product need no user input and are implemented automatically, which verifies their correctness. ~~The interaction of certain game features with others (like collisions) are proved via mutual implication.~~ For unit testing, the Unity Test framework built into the Unity 3D engine will be used. Test Cases will be generated and implemented to test the various methods implemented in certain modules which require user input either directly or indirectly, and implement a mechanic which requires various GameObjects, for eg: the shooting of a bullet. Certain modules cannot be tested using Unity Test as they have one or more GameObject(s) which are not supported by Unity Test. For these modules, manual testing will be conducted.

2.4 Testing Tools

~~Since the game is being developed using the Unity 3D game engine, which incorporates a compiler and a debugger, we will be using the outputs seen in the engine to guide our testing processes.~~The main tools being used for testing are the Unity Test framework (incorporated within the Unity 3D

engine) as well as Google Forms which will be used to receive feedback from the volunteer testers.

2.5 Testing Schedule

See Gantt Chart at the following url : https://gitlab.cas.mcmaster.ca/jonatans/Zombie_Survival_Kit/blob/018d6c62a03216446c1a934df1e510ec412cc91e/Doc/DevelopmentPlan/Gantt%20Chart%20-%20Group%206.gan

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 F1. The player must be able to move forward, left, down, and right using the WASD keys on the keyboard.

Player Movement

1. Player-Movement-M-1

   Type: Manual

   Initial State: A scene is initially paused. The scene will have the player initialized at ~~the center of~~ a certain position on the playable map, and ~~a tree~~ some trees ~~will be initialized at a position beside~~ are present around the player to act as a stationary marker.

   Input: Start scene button, WASD keys

   Output: Changes on screen.

   How test will be performed:
   The user will start the scene and the user will press down on the WASD keys. If the user notices that the position of the player changes in the scene relative to the ~~tree~~ trees, the test is concluded and is considered to have passed. If the player position stays the same relative to the tree's position while the user is pressing down on the WASD keys, the

test is considered to have failed.

3.1.2 F2. The player must be able to look in all directions by moving their mouse.

Camera Movement

1. Camera-Movement-M-1

   Type: Manual

   Initial State: A scene is initially paused. The scene will have the player initialized at ~~the center of~~ a certain position on the playable map, and ~~a tree~~ some trees ~~will be initialized at a position beside~~ are present around the player to act as a stationary marker.

   Input: Start scene button, mouse movement

   Output: Changes on screen.

   How test will be performed:
   The user will start the scene and the user will move the mouse in any direction. If the user notices that the perspective of the player changes in the scene using the ~~tree~~ trees as a marker, the test is concluded and is considered to have passed. If the ~~tree~~ trees stays at the same position on screen while the mouse is moving, the test is considered to have failed.

3.1.3 F3. Zombie enemies must walk back and forth between random points in their spawn circle, which imitates them walking around.

'Spawn' Circle

1. Spawn-Circle-A-1

Type: ~~Automated~~Manual

Initial State: A zombie enemy object is initialized at the center of its 'spawn' circle that will have a predetermined position on the playable map.

Input: void

Output: void or assertion error.

How test will be performed:
~~A test script will be created that initializes the test with the specified initial state. Within the test script, commands will be given for the zombie to move in any direction on the x, y plane of the the playable map, and will continue to move in that same direction until the zombie's position hits any point on the 'spawn' circle's perimeter; to which the zombie will change its directional movement and the process starts over again. Assert statements will be used to check if the zombie's position never passes any point along the 'spawn' circle's perimeter. After each change in direction occurs, a counter will be recorded. Once the counter reaches a specified amount, the test is concluded and the test is considered to have passed and nothing is returned (void). Any assertion errors will be considered as a failed test.~~ The user can see the zombie intialized at a certain position on the screen. If the zombie seems to be moving around a certain location on the ground, the test is said to have passed. If such a behaviour is not shown by the zombie, the test is considered to be a failure. The results are recorded with the help of a Google Form.

3.1.4 F4. Zombie enemies must start attacking the player when they come within a certain radius.

Attack Radius

1. Attack-Radius-A-1

Type: ~~Automated~~Manual

Initial State: A zombie enemy object is initialized at the center of its 'spawn' circle that will be have a predetermined position on the playable map. The player ~~object~~ is initialized outside the zombie's attack radius.

Input: ~~void~~ Keyboard input (WASD keys and Shift button)

Output: void or assertion error

How test will be performed:
~~A test script will be created that initializes the test with the specified initial state. Within the test script, commands will be given to the player to move towards the zombie's attack radius. Once the player is within the zombie enemy's attack radius, the zombie will be given commands to attack the player, and the zombie's attack animation will be enabled. Assertion statements will be used to check if the zombie's attack animation is enabled. No assertion errors indicate a passed test, whereas assertion errors indicate a failed test.~~ The user takes control of the player and moves towards the zombie instantiated on the screen using the WASD keys on the keyboard. Once the player is within a certain distance of the zombie, the zomvie should start attacking the player, and the zombie character model will trigger the attack animation. If such a behavior is observed then the test is said to have passed, otherwise the test is deemed a failure. The results are recorded with the help of a Google Form.

3.1.5 F5. Zombie enemies must follow the player if ~~they start running away while the zombie is attacking them~~ the player is at a certain distance from the zombie.

Zombie Follows Player

1. Zombie-Follows-Player-A-1

Type: ~~Automated~~Manual

Initial State: A zombie enemy object is initialized at the center of its 'spawn' circle that will be have a predetermined position on the playable map. The player ~~object~~ is initialized at a distance outside of the zombie's 'spawn' circle.

Input: ~~void~~ Keyboard input (WASD keys and Shift button)

Output: void or assertion error

How test will be performed:
~~A test script will be created that initializes the test with the specified initial state. Within the test script, commands will be given for the zombie to move in any direction on the x, y plane of the the playable map and will have its position constrained by it's spawn circle. Commands will be given to the player to move towards the zombie's 'spawn' circle center. If the position of the player and the zombie is at a certain distance from each other, commands will be given to the zombie to move its position towards the player and for the player to move away from the zombie's position. The change in position of the player and the zombie will be the same. Assertion statements will be used to check if the distance between the zombie's position and the player's position remains the same, and if the position of the zombie and player are not the same; while the zombie's position is not greater than a specified distance from the center of the 'spawn' circle. Once the zombie's position reaches this distance, the test is concluded. No assertion errors indicate a passed test, whereas assertion errors indicate a failed test.~~
The user assumes control of the player object and moves towards the zombie using the WASD keys on the keyboard. Once the player is within a certain distance of the zombie(the attack radius of the zombie), the zombie should start attacking the player. When this happens, the user should try to move the player away from the zombie. If the zombie tries to follow the player while the player is moving away from the zombie, the test is considered successful. If this behavior is not observed, the test is said to have failed. The results are recorded with

the help of a Google Form.

3.1.6 F6. If the player runs past a certain radius from the zombies original spawn location, the zombie must return to their circle.

Zombie Returns to 'Spawn' Circle Center

1. Zombie-Return-Spawn-Circle-A-1

   Type: ~~Automated~~Manual

   Initial State: A zombie enemy object is initialized at the center of its 'spawn' circle that will be a predetermined position on the playable map. The player ~~object~~ is initialized at a distance outside of the zombie's 'spawn' circle.

   Input: ~~void~~ Keyboard input (WASD keys and Shift button)

   Output: void or assertion error

   How test will be performed:
   ~~A test script will be created that initializes the test with the specified initial state. Within the test script, commands will be given for the zombie to move in any direction on the x, y plane of the the playable map and will have its position constrained by it's spawn circle. Commands will be given to the player to move towards the zombie's 'spawn' circle center. If the position of the player and the zombie is at a certain distance from each other, commands will be given to the zombie to move its position towards the player and for the player to move away from the zombie's position. The change in position of the player and the zombie will be the same. Assertion statements will be used to check if the distance between the zombie's position and the player's position remains the same, and if the position of the zombie and player are not the same; while the zombie's position is not greater than a specified distance from the center of the 'spawn' circle. Once the zombie's~~

~~position reaches this distance, the zombie will return to the center of its 'spawn' circle, concluding the test. No assertion errors indicate a passed test, whereas assertion errors indicate a failed test.~~ The user assumes control of the player and moves towards the zombie instantiated on the screen using the WASD keys on the keyboard. Once the player is within a certain distance of the zombie (the attack radius of the zombie), the zombie should start attacking the player. Once this is observed, the user tries to run away from the zombie by holding down the Shift key on the keyboard while using the WASD keys to move away from the zombie. After reaching a certain distance away from the zombie, the zombie should give up following the player and move back to its original spawn location. If this behaviour is observed, the test is considered a success, otherwise it is considered a failure. The results are recorded with the help of a Google Form.

3.1.7 F7. Different types of zombies must have different statistics (health, damage, and attack speed). Different types of zombies will be distinguishable by their different appearances.

Zombie Stats

1. Zombie-Stats-A-1

   Type: ~~Automated~~Manual

   Initial State: ~~All different zombie enemy type objects are initialized.~~ The game scene consists of the player object and diferent zombie objects.

   Input: ~~void~~ Keyboard and Mouse

   Output: void ~~or assertion error~~

   How test will be performed:
   ~~A test script will be created that initializes the test with the specified initial state. Within the test script, instructions will be given to compare all the different stats fields of each zombie enemy type. An assertion~~

~~statement will be used to check if all values within a stat field are different for each zombie enemy type. Once all assertion tests are passed, the test is conlcuded. No assertion errors indicate a passed test, whereas assertion errors indicate a failed test.~~ The game scene consists of the player and some enemy objects instantiated on a terrain. The user takes control of the player and uses the keyboard to move towards a zombie enemy and uses the left mouse button to attack the zombie. The user attacks and kills multiple zombies in this manner. If different zombies take a different number of hits to kill, the test is said to have passed, otherwise the test is considered a failure.

3.1.8 F8. Zombies must have randomly generated chance of dropping items the player can ~~use~~ equip or consume when killed.

Appearing Drops

1. ~~Appearing-Drops-A-1~~ Appearing-Drops-DM-1

   Type: ~~Automated~~ Dynamic, Manual

   Initial State: Two zombie enemy objects are initialized at the center of its 'spawn' circle that will be have a predetermined position on the playable map. The player object is initialized at a distance of the zombie's attack radius.

   Input: ~~void~~ Start scene button, WASD keys, E key, I key, left mouse button, mouse movement

   Output: ~~void or assertion error~~

   How test will be performed:
   ~~A test script will be created that initializes the test with the specified initial state. Within the test script, commands will be given to the player to attack one of the zombies. An assertion statement will be used to check if that zombie's current health value is lower than than its initial health value. Once the zombie's health value reaches zero,~~

~~an item object will appear at a chance of 100% at the position of the zombie before the zombie object is set to null (representing the first zombie's death). An assertion statement will be used to check if there exists an item object at the position where the zombie had died. The second zombie will undergo the same process as the first zombie, but an item object appearing at this zombie's position will have a chance of 0%. An assertion statement will be used to check if there does not exists an item object at the position of the zombie before the zombie object is set to null (representing the second zombie's death), and the test is concluded. No assertion errors indicate a passed test, whereas assertion errors indicate a failed test.~~

The user will start the scene and the user will press down on the WASD keys to move the player towards one of the zombies. Once the player is within attacking distance, the user will attack the zombie by pressing the left mouse button until the zombie dies. The user will check if a drop has appeared and then pick up the drop using the E key. The user will press the I key to check if the item has been stored in the invenotry. The user will repeat this process for the second zombie. If any of the checks fail, the test has failed. Otherwise, the test has passed.

3.1.9 ~~Func. Req. 9: Dropped items must be automatically deleted if not picked up within a certain time~~

~~Disappearing Drops~~

1. ~~Disappearing-Drops-A-1~~

   ~~Type: Automated~~

   ~~Initial State: A zombie enemy objects is initialized at the center of its 'spawn' circle that will be have a predetermined position on the playable map. The player object is initialized at a distance of the zombie's attack radius.~~

   ~~Input: void~~

~~Output: void or assertion error~~

~~How test will be performed:~~
~~A test script will be created that initializes the test with the specified initial state. Within the test script, commands will be given to the player to attack one of the zombies. An assertion statement will be used to check if that zombie's current health value is lower than than its initial health value. Once the zombie's health value reaches zero, an item object will appear at a chance of 100% at the position of the zombie before the zombie object is set to null (representing the zombie's death). An assertion statement will be used to check if there exists an item object at the position where the zombie had died. Instructions to wait for the amount of time it takes for the item to disappear are given. An assertion statement will be used to check if there does not exists an item object at the same position, and the test is concluded. No assertion errors indicate a passed test, whereas assertion errors indicate a failed test.~~

3.1.10 F9. The player must be able to pick up items they are looking at with the interact key.

Pickup Drops

1. ~~Pickup-Drops-A-1~~ Pickup-Drops-DM-1

   Type: ~~Automated~~ Dynamic, Manual

   Initial State: A player object is initialized at any predetermined position, and an item object is also initialized near the player's position.

   Input: ~~void~~ Start scene button, WASD keys, E key, mouse movements

   Output: ~~void or assertion error~~ The item object will disappear once the item has been picked up by the player

How test will be performed:
~~A test script will be created that initializes the test with the specified initial state. Within the test script, commands will be given to the player to move towards the item and, once close enough, to change the camera's foward position to the item. Then, the 'interact' key will be inputted. An assertion statement will be used to check if there exists the item in the player's inventory, and the test is concluded. No assertion errors indicate a passed test, whereas assertion errors indicate a failed test.~~
The user will start the scene and the user will press down on the WASD keys to move towards the item. The user will move the mouse until the reticle is on the item, and then they will press the E key to pick up the item. If the item disappears from the scene, then the test was successful. If it stays in the scene, the test has failed

3.1.11 F10. The player must be able to access their inventory by pressing the inventory key, and will have a maximum space of 20 items.

Inventory Access

1. ~~Inventory-Access-A-1~~ Inventory-Access-DM-1

   Type: ~~Automated~~ Dynamic, Manual

   Initial State: A player object is initialized at any predetermined position

   Input: ~~void~~ Start scene button, I key

   Output: ~~void or assertion error~~ The Inventory UI will appear on screen

   How test will be performed:
   ~~The user will start the scene and the user will press down on the WASD keys. If the user notices that the position of the player changes in the scene relative to the tree, the test is concluded and is considered to~~

~~have passed. If the player position stays the same relative to the tree's position while the user is pressing down on th WASD keys, the test is considered to have failed.~~
The user will start the scene. Once the scene has started, the user will press down on the I key. If the user observes that the Inventory UI appears on the screen, the test has succeeded; if not, the test has failed

3.1.12 F11. The player must be able to equip, consume, ~~delete, and move around~~ and drop items in their Inventory UI using the ~~mouse~~ LMB.

~~Use Items Move Items~~ Use Remove Items

1. ~~Use-Move-Items-M-1~~ Use-Remove-Items-DM-1

   Type: Dynamic, Manual

   Initial State: A scene is initially paused. A player object is initialized at the center of the playable map. Equipment and consumable item objects are initialized close to the player's position.

   Input: Start scene button, WASD keys, mouse movement, mouse left-click, ~~'inventory' key, 'interact' key, 'de-equip' button~~ I key, E key, U key, remove button

   Output: Items picked up by the player will appear in the Inventory UI. Equipment items used will be equipped and shown in the Equipment UI. Consumable items used will dissappear from the Inventory UI and the player's health will increase. Items removed from the Inventory UI will be removed and appear as an item game object in front of the player and drop to the terrain.

   How test will be performed:
   The user will start the scene and the user will press down on the WASD keys to move the player towards the items and stop once the player is close enough to the item to interact with it. The user will then use

the mouse to change the camera view until rectile is over the item and then the user will press the E key to pick it up. The user will do this for all items. The user will then press the ~~'inventory'~~ I key to open the inventory UI, and use the mouse to hover over either item icon. The user will press down on the left mouse button to use the item. If the item used is the consumable, the user will check the player's health value to see if it has increased by a specified amount. The user will also check if the item no longer exists in the player's inventory. The user will then press down on the left mouse button to use one of the equipment item. The user will check if the equipment has been equipped in one of the player's equipment slot and if the equipment item has disappeared from the player's inventory. ~~The user will then press the 'de-equip' button and will check if the equipment is no longer in the player's equipment slot, but back in the player's inventory. The user will move the equipment item to another slot by pressing and holding down the left mouse button on the item icon, and drag the mouse to an empty slot in the player's inventory. The user will check if the item has moved to its intended spot.~~ The user will then press the left mouse button on the remove button ~~in the top right corner of the slot the item is in~~ for all items in the inventory. The user will check if the ~~item has~~ items have been removed from the player's inventory and ~~has~~ have appeared at some close distance in front of the player. If any of the user's check fails, the test is considered to have failed. Otherwise, the test was passed.

2. Use-Remove-Items-BUA-2

   Type: Black Box, Unit, Automatic

   Initial State: void

   Input: void

   Output: void

How test will be performed:
The methods that implement adding items to the inventory, using items, and removing items from the inventory will be tested. Assertion statements will be used for expected outputs. If an assertion errors does not occur, then the test was successful. If an assertion error does occur, then the test has failed.

3.1.13 F12. The player must be able to unequip items in their Equipment UI using the LMB or the 'unequip all key.

Unequip

1. Unequip-DM-1

   Type: Dynamic, Manual

   Initial State: A player object and equipment item objects are initialized at a predetermined position

   Input: Start scene button, WASD keys, E key, I key mouse movement, left mouse button

   Output: The equipment items, when used, will appear in the Equipment UI. Items removed from the Equipment UI will reappear in the Invenotry UI

   How test will be performed:
   The user will start the scene and the user will press down on the WASD keys to move the player towards the items and stop once the player is close enough to the item to interact with it. The user will then use the mouse to change the camera view until rectile is over the item and then the user will press the E key to pick it up. The user will do this for all items. The user will then press the I key to open the inventory UI, and use the mouse to hover over the equipment item icon. The user will press down on the left mouse button to equip the item. The user will do this for all equipment items in the inventory, and check if each item has been transferred over to the Equipment UI. The user will then

use the the remove button on one of the items in the Equipment UI, and then press the U key. The user will check if all items have been removed from the Equipment UI and has reappeared in the Inventory UI. If any of the user's check fails, the test is considered to have failed. Otherwise, the test was passed.

2. Unequip-BUA-2

   Type: Black Box, Unit, Automatic

   Initial State: void

   Input: void

   Output: void

   How test will be performed:
   The methods that implement equipping and unequipping items will be tested. Assertion statements will be used for expected outputs. If an assertion errors does not occur, then the test was successful. If an assertion error does occur, then the test has failed.

3.1.14 F13. The player must be able to fire/use ~~their equipped weapon~~ an equipped gun or axe by pressing the left mouse button (LMB).

Attack Items

1. Attack-Items-M-1

   Type: Manual

   Initial State: A scene is initially paused. A player object is initialized at the center of the playable map. A weapon type equipment is initialized to the player's corresponding equipment slot. A zombie enemy object is initialized close to the enemy

Input: Start scene button, WASD keys, mouse movement, left mouse button

Output: Changes on screen.

How test will be performed:
The user will start the scene and the user will press down on the WASD keys to move the player towards the zombie until the player is within attacking range from the zombie. The user will move the mouse until the zombie is in the middle of the user's screen and will then press the left mouse button. The user will observe if the zombie's health is decreasing after each attack. If any of the user's check fails, the test is considered to have failed. Otherwise, the test was passed.

3.1.15 F14. If the player has a firearm equipped, ~~they must be able to aim down sights using the right mouse button (RMB)~~ the user can reload the gun using the reload key. The mazimum amount of bullets will be 7.

Aiming

1. Aiming-M-1

Type: Changes on screen

Initial State: A scene is initially paused. A player object is initialized at the center of the playable map. A weapon type equipment is initialized to the player's corresponding equipment slot.

Input: Start scene button, mouse movement, right left-click

Output: void

How test will be performed:
The user will start the scene and the user will press down and hold on the right mouse button. The user will check if the firearm object moves to a position that shows the player is "aiming down their sights".

The player will then move the mouse while still holding down the right mouse button and will check if where the firearm is pointing at changes. If any of the user's check fails, the test is considered to have failed. Otherwise, the test was passed.

3.1.16 F15. The environment must slowly go through a day and night cycle.

Day and Night

1. Day-Night-A-1

   Type: Automated

   Initial State: Day time at the current system time.

   Input: void

   Output: void or assertion error

   How test will be performed:
   A test script will be created that initializes the test with the specified initial state. Within the test script, instructions will be given for the function that changes day time into night time to run, and to check how much time has passed from when the test script started. Once the time needed for day time to become night time has passed, an assertion statement will check if it is night time. Once again, instructions will be given to check how much time has passed after the previously mentioned assertion statement has been checked. Once the time needed for night time to become day time, an assertion statement will check if it is day time. No assertion errors indicate a passed test, whereas assertion errors indicate a failed test.

3.1.17 F16. The player must lose health when hit by a zombie.

Zombie Attack

1. Zombie-Attack-A-1

   Type: ~~Automated~~ Manual

   Initial State: A zombie enemy object is initialized at the center of its 'spawn' circle that will be have a predetermined position on the playable map. The player object is initialized outside the zombie's attack radius.

   Input: ~~void~~ Keyboard

   Output: void or assertion error

   How test will be performed:
   ~~A test script will be created that initializes the test with the specified initial state. Within the test script, commands will be given to the player to move towards the zombie's attack radius. Once the player is within the zombie enemy's attack radius, the zombie will be given commands to attack the player and the zombie's attack animation will be enabled. An assertion statements will be used to check after each zombie attack if the player's health is lower than its previous health. Once the player's health reaches zero, the test is concluded. No assertion errors indicate a passed test, whereas assertion errors indicate a failed test.~~ The user assumes control of the player and guides the player to a zombie using the WASD keys on the keyboard. Once inside the zombie's attack radius, the zombie should start attacking the player. The player's health bar at the bottom left corner of the screen should become shorter in lenght, indicating a loss of health. If this behaviour is observed, the test is said to have passed, otherwise it is considered a failure. The results are recorded with the help of a Google Form.

3.1.18 F17. Zombies must lose health when hit by the player by using the LMB once the player is within attacking distance from the zombie.

Player Attack

1. Player-Attack-A-1

   Type: ~~Automated~~ Manual

   Initial State: A zombie enemy object is initialized at the center of its 'spawn' circle that will be have a predetermined position on the playable map. The player object is initialized outside the zombie's attack radius.

   Input: ~~void~~ Keyboard and Mouse

   Output: void or assertion error

   How test will be performed:
   ~~A test script will be created that initializes the test with the specified initial state. Within the test script, commands will be given to the player to move and change its camera forward positon towards the zombie. Once the player is within attacking distance, the player will be given commands to attack the zombie and the player's attack animation will be enabled. An assertion statement will be used to check, after each player attack, if the zombie's health is lower than its previous health. Once the zombie's health reaches zero, the test is concluded. No assertion errors indicate a passed test, whereas assertion errors indicate a failed test.~~ The user assumes control of the player by using the WASD keys to move the player and the mouse to move the camera and attack. The player walks up to a zombie and keeps attacking the zombie till the zombie dies. If the zombie dies, this test is considered a success, a failure otherwise.

3.1.19 F18. Players or zombies must die when they reach 0 health.

Death

1. Death-A-1

   Input: ~~void~~ Keyboard and Mouse

   Initial State: Two zombie enemy objects is initialized at the center of its 'spawn' circle that will be have a predetermined position on the playable map. The player object is initialized outside the zombie's attack radius.

   Type: ~~Automated~~ Manual

   Output: void or assertion error

   How test will be performed:
   ~~A test script will be created that initializes the test with the specified initial state. Within the test script, commands will be given to the player to move and change its camera forward positon towards the first zombie. Once the player is within attacking distance, the player will be given commands to attack the first zombie and the player's attack animation will be enabled. Assertion statements will be used to check, after each player attack, if the first zombie's health is lower than its previous health. Once the first zombie's health reaches zero, the first zombie will have died. An assertion statement will be used to check that the first zombie not longer exists (null). Commands will then be given to the second zombie to move towards the player. Once the player is within the zombie's attack radius, the zombie's attack animation will begin. An assertion statements will be used to check after each zombie attack if the player's health is lower than its previous health. Once the player's health reaches zero, the player is dead. An assertion statement will be used to check if the player is dead and the test is concluded. No assertion errors indicate a passed test, whereas assertion errors indicate a failed test.~~ The user assumes control of the player object and guides the player towards a zombie. When the player enters the zombie's attack radius, the zombie starts attacking the player. The user attacks the zombie using the left mouse button. After multiple attacks, the zombie dies and drops a pickup on the ground. The user then guides

the player to another zombie, and once in the zombie's attack radius, the zombie starts attacking the player. This time, the player does not attack the zombie or run away. The health bar at the bottom left corner should continues to deplete and once it reaches zero, the screen displays the "Game Over" message. Only if all these conditions are observed is the test considered a pass.

3.1.20 ~~Func. Req. 19: Upon player death, the game must reset~~

~~Reset~~

1. ~~Reset-M-1~~

   ~~Type: Manual~~

   ~~Initial State: A player object is initialized in the center of the playable map and a zombie enemy object is initialized at a distance from the player that is within the zombie's attack radius.~~

   ~~Input: Scene start button~~

   ~~Output: Changes on screen~~

   ~~How test will be performed:~~
   ~~The user will start the scene. The user will check if the zombie is attacking the player and if the player's health is decreasing after each zombie attack. Once the player's health decreases to zero, the user will check if the scene resets to its inital state where the process begins all over again. If any of the user's check fails, the test is considered to have failed. Otherwise, the test was passed.~~

3.1.21 F19. An Inventory UI will give the user a visual representation of the player's inventory and will be accessable by pressing the inventory key.

InventoryUI

1. InventoryUI-DM-1

   Type: Dynamic, Manual

   Initial State: A player object will be instantiate at a predetermined location

   Input: Start scene button, I key

   Output: The Inventory UI will appear on the bottom right corner of the screen after the user presses down on the I key, and the Inventory UI will disappear when the I key is pressed once again.

   How test will be performed:
   The user will start the scene and the user will press down on the I key. The user will check if the Inventory UI appears. on the bottom right corner of the screen. Then the user will press the I key again and check if the Inventory UI has disappeared from the screen. If any of the user's check fails, the test is considered to have failed. Otherwise, the test was passed.

3.1.22 F20. An Equipment UI will give the user a visual representation of the player's equipped items and will be appear with the Inventory UI by pressing the inventory key.

Equipment UI

1. EquipmentUI-DM-1

   Type: Dynamic, Manual

   Initial State: A player object will be instantiate at a predetermined location

Input: Start scene button, I key

Output: The Equipment UI will appear on the top left corner of the screen after the user presses down on the I key, and the Equipment UI will dissappear when the I key is pressed once again.

How test will be performed:
The user will start the scene and the user will press down on the I key. The user will check if the Equipment UI appears. on the top left corner of the screen. Then the user will press the I key again and check if the Equipment UI has disappeared from the screen. If any of the user's check fails, the test is considered to have failed. Otherwise, the test was passed.

3.1.23 F21. A Player UI will give the user a visual representation of the player's health and remaining bullets if a gun is equipped to the player. The Player UI will also consist of a reticle.

Player UI

1. PlayerUI-SM-1

   Type: Static, Manual

   Initial State: A player object will be instantiate at a predetermined location

   Input: Start scene button

   Output: A Player UI on the bottom left corner of the screen

   How test will be performed:
   The user will start the scene and the user will check if there exists a Player UI on the bottom left corner of the screen. If any of the user's

check fails, the test is considered to have failed. Otherwise, the test was passed.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel Requirements

Look and Feel

1. Graphics-F-1

   Type: Functional

   Initial State: Scene is paused. All game mechanics, game functions/features, game objects, and interactions will be intialized for the user to begin playing the current game Zombie Survival Kit.

   Input/Condition: Follow the instructions of the surveyor.

   Output/Result: Filled Survey.

   How test will be performed: A person (not in group 6) will be selected to try the game Zombie Survival Kit. The user will follow the surveyor's instructions as to what they should make the player (in the game) do. The surveyor will ask the user to start the game and to look at all the initialized objects and playable map.
   Afterwards, they will fill in an answer to a question relating to how the graphics appear to them.

2. Ease-of-Use-F-1

   Type: Functional T-NF2

   Initial State: Scene is paused. All game mechanics, game functions/features, game objects, and interactions will be intialized for the user to begin

playing the current game Zombie Survival Kit.

Input: Follow the instructions of the surveyor.

Output: Filled Survey.

How test will be performed:
A person (not in group 6) will be selected to try the game Zombie Survival Kit. The user will follow the surveyor's instructions as to what they should make the player (in the game) do.
The surveyor will tell the user all the possible features of the game. Then the surveyor will ask the user to use the keyboard or mouse to perform the features.
Afterwards, they will fill in an answer to a question relating to how intuitive or easy it is to use all the possible input commands through the mouse and keyboard.

3. Ease-of-Use-F-1

Type: Functional

Initial State: Scene is paused. All game mechanics, game functions/features, game objects, and interactions will be intialized for the user to begin playing the current game Zombie Survival Kit.

Input: Follow the instructions of the surveyor.

Output: Filled Survey.

How test will be performed:
A person (not in group 6) will be selected to try the game Zombie Survival Kit. The user will follow the surveyor's instructions as to what they should make the player (in the game) do. The surveyor will ask the user to input all the possible input commands. Afterwards, they will

fill in an answer to a question relating to how responsive the game feels when inputting all the commands through the mouse and/or keyboard.

3.2.2 Usability and Humanity Requirements

1. Min-Learn-F-1

   Type: Functional

   Initial State: Scene is paused. All game mechanics, game functions/features, game objects, and interactions will be intialized for the user to begin playing the current game Zombie Survival Kit.

   Input/Condition: Follow the instructions of the surveyor.

   Output/Result: Filled Survey.

   How test will be performed:
   A person (not in group 6) will be selected to try the game Zombie Survival Kit. The user will follow the surveyor's instructions as to what they should make the player (in the game) do.
   The surveyor will teach the user all the possible input commands.
   Afterwards, they will fill in an answer to a question relating to how easy it was to learn all the input commands through the mouse and/or keyboard.

3.2.3 Performance Requirements

1. Start-Time-F-1

   Type: Functional

   Initial State: Scene is paused. All game mechanics, game functions/features, game objects, and interactions will be intialized for the user to begin

playing the current game Zombie Survival Kit.

Input/Condition: Follow the instructions of the surveyor.

Output/Result: Filled Survey.

How test will be performed:
A person (not in group 6) will be selected to try the game Zombie Survival Kit. The user will follow the surveyor's instructions as to what they should make the player (in the game) do.
The surveyor will ask the user to time how long it takes to start the game. Then the surveyor will ask the user to start the game.
Afterwards, they will fill in an answer to a question relating to how fast did it take for the scene to load.

2. FPS-F-1

Type: Functional

Initial State: Scene is paused. All game mechanics, game functions/features, game objects, and interactions will be intialized for the user to begin playing the current game Zombie Survival Kit. The script developed by Dave Hampson to display the framerate of the scene in Unity will be added and initalized to the scene.

Input/Condition: Start scene button

Output/Result: Frames per second

How test will be performed:
The user will start the scene and will read the framerate on the screen produced by the script by Dave Hampson (http://wiki.unity3d.com/index.php/FramesPerSec
If the framerate runs at a speed equal to or higher than 60 frames per second while performing all the features of Zombie Survival Kit, the test passes. Otherwise the test failed.

3. Close-Time-F-1

   Type: Functional

   Initial State: Scene is paused. All game mechanics, game functions/features, game objects, and interactions will be intialized for the user to begin playing the current game Zombie Survival Kit.

   Input/Condition: Follow the instructions of the surveyor.

   Output/Result: Filled Survey.

   How test will be performed:
   A person (not in group 6) will be selected to try the game Zombie Survival Kit. The user will follow the surveyor's instructions as to what they should make the player (in the game) do.
   The surveyor will ask the user to start the scene, and once the scene starts, the surveyor will ask the user to time how long it takes to close the game. The surveyor will then ask the user to close the game.
   Afterwards, they will fill in an answer to a question relating to how fast did it take for the scene to stop playing.

4. ~~Close-Time-F-1~~ Enough-Ram-F-1

   Type: Functional

   Initial State: void

   Input/Condition: Checking to see that the ~~space~~ ram needed to ~~download the playable executable of~~ play Zombie Survival Kit does not surpass 1GB.

   Output/Result: The space needed to download the playable executable of Zombie Survival Kit

How test will be performed:
The user will check how much ~~space~~ ram ~~is required to download the playable executable of~~ Zombie Survival Kit takes up. If ~~space~~ the amount ram used ~~required~~ does not exceed 1GB, the test has passed. Otherwise, the test has failed.

3.2.4 Safety Critical Requirements

1. Enough-Space-F-1

   Type: Functional

   Initial State: The playable exectuable of Zombie Survival Kit is not downloaded.

   Input/Condition: The playable exectuable of Zombie Survival Kit will not be downloaded if there is not enough space in the hard-drive.

   Output/Result: Downloaded the playable exectuable of Zombie Survival Kit, or the playable exectuable of Zombie Survival Kit was not downloaded

   How test will be performed:
   A computer without the space required to download the playable exectuable of Zombie Survival Kit will try to download it. If the download is interuppted due to lack of hard-drive space, the test has passsed. Otherwise the test has failed.

3.2.5 Precision Requirements

1. Player-Attack-Animation-to-Effect-F-1

Type: Functional

Initial State: Scene is paused. All game mechanics, game functions/features, game objects, and interactions will be intialized for the user to begin playing the current game Zombie Survival Kit.

Input/Condition: Follow the instructions of the surveyor.

Output/Result: Filled Survey.

How test will be performed:
A person (not in group 6) will be selected to try the game Zombie Survival Kit. The user will follow the surveyor's instructions as to what they should make the player (in the game) do.
The surveyor will ask the user to move the player towards a zombie, and to have the player attack the zombie.
Afterwards, they will fill in an answer to a question relating to how precise the zombie's attack animation was to the effect it had on the player's health.

2. Time-Day-Night-Transition-F-1

Type: Functional

Initial State: Scene is paused. All game mechanics, game functions/features, game objects, and interactions will be intialized for the user to begin playing the current game Zombie Survival Kit.

Input/Condition: Follow the instructions of the surveyor.

Output/Result: Filled Survey.

How test will be performed:
A person (not in group 6) will be selected to try the game Zombie Sur-

vival Kit. The user will follow the surveyor's instructions as to what they should make the player (in the game) do.
The surveyor will tell the user before starting the scene, how long it will take for the environment to change from day to night and night to day. Then the surveyor will ask the user to time how long it takes for day to become night and night to become day, after starting the scene.
Afterwards, they will fill in an answer to a question relating to how precise the change in day or night in the game was to the amount of time that has passed.

3. Zombie-Attack-Animation-to-Effect-F-1

   Type: Functional

   Initial State: Scene is paused. All game mechanics, game functions/features, game objects, and interactions will be intialized for the user to begin playing the current game Zombie Survival Kit.

   Input/Condition: Follow the instructions of the surveyor.

   Output/Result: Filled Survey.

   How test will be performed:
   A person (not in group 6) will be selected to try the game Zombie Survival Kit. The user will follow the surveyor's instructions as to what they should make the player (in the game) do.
   The surveyor will ask the user to move the player towards a zombie, and let the zombie attack the player.
   Afterwards, they will fill in an answer to a question relating to how precise the zombie's attack animation was to the effect it had on the player's health.

4. Distance-to-Pickup-Item-F-1

Type: Functional

Initial State: Scene is paused. All game mechanics, game functions/features, game objects, and interactions will be intialized for the user to begin playing the current game Zombie Survival Kit.

Input/Condition: Follow the instructions of the surveyor.

Output/Result: Filled Survey.

How test will be performed:
A person (not in group 6) will be selected to try the game Zombie Survival Kit. The user will follow the surveyor's instructions as to what they should make the player (in the game) do.
The surveyor will ask the user to move the player towards the the item, and to stop at any distance from the item before pressing the 'interact' key to pick up the item. Afterwards, they will fill in an answer to a question relating to how intuitive it was to bring the player to a certain distance from the item before they pressed the 'interact' key to pick up the item.

3.2.6 Reliability and Availability Requirements

1. Reliable-Input-Commands-F-1

   Type: Functional

   Initial State: Scene is paused. All game mechanics, game functions/features, game objects, and interactions will be intialized for the user to begin playing the current game Zombie Survival Kit.

   Input/Condition: Follow the instructions of the surveyor.

   Output/Result: Filled Survey.

How test will be performed:
A person (not in group 6) will be selected to try the game Zombie Survival Kit. The user will follow the surveyor's instructions as to what they should make the player (in the game) do.
The surveyor will ask the user to perform all the possible input commands and will tell the user what will happen in the game before the commands are inputted.
Afterwards, they will fill in an answer to a question relating to how reliabile the intended output would appear, after inputting each input command from the mouse and keyboard.

2. Play-When-On-F-1

   Type: Functional

   Initial State: Scene is paused. All game mechanics, game functions/features, game objects, and interactions will be intialized for the user to begin playing the current game Zombie Survival Kit.

   Input/Condition: Follow the instructions of the surveyor.

   Output/Result: Filled Survey.

   How test will be performed:
   A person (not in group 6) will be selected to try the game Zombie Survival Kit. The user will follow the surveyor's instructions as to what they should do.
   The surveyor will instruct the user to execute the executable for Zombie Survival Kit, start the scene, and then close the game ten times.
   Afterwards, they will fill in an answer to a question relating to how often the game executed from the ten times the user opened and closed the game.

3.2.7 Cultural Requirements

1. Culture-Discretion-F-1

   Type: Functional

   Initial State: Scene is paused. All game mechanics, game functions/features, game objects, and interactions will be intialized for the user to begin playing the current game Zombie Survival Kit.

   Input/Condition: Follow the instructions of the surveyor.

   Output/Result: Filled Survey.

   How test will be performed:
   A person (not in group 6) will be selected to try the game Zombie Survival Kit. The user will follow the surveyor's instructions as to what they should do.
   The surveyor will instruct the user to perform all the features implemented in Zombie Survival Kit
   Afterwards, they will fill in an answer to a question relating to if they found any animations, game objects, or sounds offensive to any culture or political background the user is able to think of.

3.2.8 Non-function Requirements not being tested on

1. Operational and Environmental Requirements

   The operational and Environmental requirements are expected system requirements that the user will have to run the game, and includes requirements detailing what partner applications are used to develop Zombie Survival Kit. These are design decisions that Group 6ix feels to bring out the best experience of Zombie Survival Kit which do not require testing.

2. Security Requirements
   ~~1. Zombie Survival Kit shall not require any personal information from the user.~~

   We cannot test what we do not require the game to do. Therefore there are no tests for the Security Requirements.

3. Legal Requirements

   We cannot test the license in which Zombie Survival is under.

4. Health and Safety Requirements
   ~~1. Users with a history of epilepsy or sensitivity to flashing lights should notuse this product.~~

   ~~2. The product should be used in a well lit environment~~

   ~~3. Correct posture should be ensured by the user while sitting on computer~~

   ~~4. After every hour of use, a 10 minute break is highly recommended.~~

   ~~5. If the user experiences watering of eyes, a sensation of dizziness or nausea,~~

   ~~use of the product should be ceased immediately.~~

   ~~6. The product should never be used if the user is feeling sleepy.~~

   ~~7. The computer being used should be well ventilated and should be cooled adequately.~~

   ~~8. The frame rate should be at least 60 FPS at all times to prevent motion sickness, and frame time variance should be checked and maintained~~

~~at a maximum of 16 ms.~~

~~9. The age of the user should be at least 12 years.~~

These requirements concerning the safety of the user and the equipment required for Zombie Survival Kit will not be tested on because failure to any of these requirements may consitute a health hazard.

3.3 Traceability Between Test Cases and Requirements

~~All the functional and non-function test cases as stated above have detailed instructions as to what constitutes a passed or failed test case. Each step within each test case are also described allowing for high traceability between test cases and requirements.~~

Functional Requirements	**Tests**
F1	T-F1
F2	T-F2
F3	T-F3
F4	T-F4
F5	T-F5
F6	T-F6
F7	T-F7
F8	T-F8
F9	T-F9
F10	T-F10
F11	T-F11, T-F12
F12	T-F13, T-F14
F13	T-F15
F14	T-F16
F15	T-F17
F16	T-F18
F17	T-F19
F18	T-F20
F19	T-F21
F20	T-F22
F21	T-F23

Table 3: Trace Between Functional Requirements and Tests

Non-Functional Requirements	**Tests**
NF1	T-NF1
NF2	T-NF2
NF3	T-NF3
NF4	not tested
NF5	T-NF4
NF6	not tested
NF7	T-NF5
NF8	T-NF6
NF9	T-NF7
NF10	T-NF8
NF11	T-NF9
NF12	T-NF10, T-NF12
NF13	T-NF11
NF14	T-NF13
NF15	T-NF14
NF16	T-NF15
NF17	not tested
NF18	not tested
NF19	not tested
NF20	not tested
NF21	not tested
NF22	not tested
NF23	not tested
NF24	not tested
NF25	not tested
NF26	T-NF16
NF27	not tested
NF28	not tested
NF29	not tested
NF30	not tested
NF31	not tested
NF32	not tested
NF33	not tested
NF34	not tested
NF35	not tested
NF36	not tested

Table 4: Trace Between Non-Functional Requirements and Tests

4 Tests for Proof of Concept

4.1 Movement and Camera

1. Movement

   Type: Manual

   Initial State: The player is instantiated at the center of the terrain and can see a tree on the screen.

   Input: W-A-S-D, Shift and Space keys on the keyboard.

   Output: Change of player perspective in the game scene.

   How test will be performed: The user will press the W key on the keyboard. This causes the player to move towards the tree seen on the screen, and cross a red marking on the ground. This confirms that the player is moving forward with respect to the starting position. If the player's position remains the same as the starting position after pressing the W key, the test is considered a failure. The A, S and D keys move the player left, backwards and right respectively in a similar manner.
   By pressing and holding the Shift key on the keyboard, the player should move at a faster speed compared to when the Shift key is not held down. If this behavior is not observed, the test is a failure.
   Pressing the Space bar shall cause the player to jump in the current position. If the player is in motion when the Space bar is pressed, the player should move in the direction of movement even when off the ground. If this behavior is not observed, the test is a failure.

2. Camera

   Type: Manual

   Initial State: The player is instantiated at the center of the terrain and can see a tree on the screen.

   Input: Mouse movement

   Output: Change of player perspective in the game scene.

How test will be performed: The user stays in the initial starting point, and moves the mouse in an arbitrary direction. The in-game camera should move in the same direction as the mouse, causing a change in the perspective of the player with respect to the game world. If this behavior is observed, the test is successful.
The speed of movement would depend on the hardware settings and are subject to change on each computer. If the player perspective fails to change after moving the mouse, the test is declared a failure.

4.2 Inventory System

1. Inventory UI

   Type: Manual

   Initial State: The player is instantiated at the center of the terrain and can see a tree on the screen.

   Input: I button on keyboard

   Output: A UI element shows up on the screen.

   How test will be performed: At any point of time, the user shall press the I key on the keyboard. This shall halt mouse-based camera movement, and a grid of items shall appear on the right hand side of the game screen. In case this behavior is not observed, the test is considered to have failed.

2. Item pickup

   Type: Manual

   Initial State: The player is instantiated at the center of the terrain and can see a tree on the screen.

   Input: W-A-S-D, E and I buttons on keyboard, mouse movement.

   Output: A new item shall be visible in the Inventory UI.

   How test will be performed: The user moves towards a pickable object using the W-A-S-D keys, and then uses the mouse to look at it. When the user is looking at the pickable object, the user presses the E button on the keyboard. This should make the pickable object disappear

from the game screen. At this point, by pressing the I button on the keyboard, the user can see the pickable object appear as a small icon in the Inventory UI. This would indicate success of this test.

3. Item drop

   Type: Manual

   Initial State: The player is instantiated at the center of the terrain and can see a tree on the screen.

   Input: I button on keyboard, mouse movement.

   Output: An item shall be removed from the Inventory UI.

   How test will be performed: The user opens the Inventory UI using the I button. If the Inventory UI shows no objects in it, it means that the Inventory is empty and nothing can be dropped from it. If there is an object in the Inventory UI, the object icon has a small 'x' button on the top right corner of the icon. By left-clicking on this 'x' icon, the user can drop this item from the Inventory. If after clicking the 'x' button, the item still remains in the inventory, this test has failed.

4. Item consumption

   Type: Manual

   Initial State: The player is instantiated at the center of the terrain and can see a tree on the screen.

   Input: I button on keyboard, mouse movement.

   Output: Player stats shall change.

   How test will be performed: The user opens the Inventory UI using the I button. If the Inventory UI shows no objects in it, it means that the Inventory is empty and nothing can be consumed from it. If there is an object in the Inventory UI, the user can left-click on the object icon to use the item. As an example, if the player has a fruit object in the inventory, then by clicking on the icon for the fruit object in the inventory UI, the player's health level is increased.

4.3 Enemy pathfinding

1. Enemy moving to player

   Type: Automated

   Initial State: The player is instantiated at the center of the terrain and can see a tree on the screen, enemy zombies are seen moving around at various places in the game scene.

   Input: W-A-S-D and Shift keys on keyboard.

   Output: Player stats shall change.

   How test will be performed: The user sees many enemies walking around a position in the game scene. When the user moves the player (using the WASD and Shift keys) to within a certain distance of an enemy zombie, the zombie starts moving towards the player. If the player now moves away from the zombie, then at reaching a certain distance from the zombie, the zombie stops chasing the player and then starts moving about a position.

5 Comparison to Existing Implementation

~~Group 6's version of this implementation has almost caught up to and replicated the existing open source implementation's functionalities. Currently, both implementations have a basic testing environment, and not a full playable terrain created using Unity. Both programs also have an inventory and equipment system implemented, allowing the user to pick up various items into their inventory and consume/equip them as they please.~~ Group 6 has finished the implementation of all the functionalities present in the existing open source implementation. Group 6's implementation now has a sample terrain with various models like trees, bridges and mountains, as well as fully functional melee and range weapons, a working inventory system, a day and night cycle, as well as sound effects to indicate that an attack has been completed. The sounds effects are an extra feature not present in the exisiting implementation.

The open source project has implemented visibly equipping melee and ranged weapons and using them with animations, ~~whereas Group 6 has not~~

~~yet developed that far into the aesthetics of the project and currently only has a basic left-click enemy damage system~~, which differs from Group 6's implementation in that Group 6's implementation uses still images for weapons. ~~Although behind on that aspect, a~~ A basic enemy AI system has been developed where the enemies wander around in a certain radius, and follow/attack the player when they get too close. The existing implementation does not have enemy AI and only has a still object that you can practice attacking with your weapons.

~~Parallel testing will not be used since much of the existing implementation is not fully complete or put together. Although it would have been useful, there are not many components to test that would be beneficecial in testing the current implementation aswell. The current project was inspired by the existing implementation in an attempt to finish and make it what it was designed to be like, but since the start of development it has become very independent and not attached to the open source code, making running tests on it not an effective use of time~~ Testing for the implementation has been conducted using a mix of Automated and Manual testing. Unity Test scripts have been used for various functional and non-functional requirements. Due to some restrictions imposed by the nature of some GameObjects (such as in the modules 'Enemy', 'Zombie' and 'ZombieStats'), some components have been tested manually with the help of volunteers, who provide feedback through a Google Form..

6 Unit Testing Plan

6.1 Unit testing of internal functions

Unit tests for internal functions will be accomplished using ASSERT statements. ~~As of now, these statements will be embedded in the C# scripts in each testable function. In order to test that the code is functional, each statement will be placed under the main part of each function~~ These ASSERT statements are located in dedicated test scripts which use the Unity Test framework. Each test script corresponds to one module in the implementation.. Inside, certain values will be asserted to check if they are equal, or certain statements will be inputted and checked if they evaluate to true or false which will prove the correctness of the code.

For example, assertion statements will be used to make sure the combat system is correct. After the code for the enemy attacking the player runs, there will be an assertion to check if the player's new health is equal to the player's old health minus the enemy's attack value. If the assertion passes, it is known that the the enemy has successfully attacked and reduced the player's health. The same thing is tested for the opposite scenario, where the player attacks the enemy. There is an assertion to check if the enemy's new health is equal to the enemy's old health minus the player's attack value.

Another example of unit testing that will be done is for the inventory system. Once the player picks up an item, it should be added into the first available inventory slot. Assertion statements can be used to check if the item was successfully placed into the slot. This is done by running an assertion that checks if the first available inventory slot is filled with an Item of the type that was just picked up. If the assertion passes, the player has successfully picked up the item into their inventory

All test scripts for unit testing of internal functions are found in:
Test-Scripts

Below is the outline of how the Unit Test scripts verify correctness:

6.1.1 Test_CharacterCombat

Purpose Verify that the Attack() method functions properly.

Setup: The following variables are needed for this test:

- GameManager (GameObject): Required for Unity Test
- character (GameObject): The character that carries out an attack
- character2 (GameObject): The character that is attacked
- GunUI (GameObject): The Gun ammo count UI
- AxeUI (GameObject): The Axe UI element of the canvas

- attacker (CharacterCombat): Reference to the CharacterCombat script attached to 'character'.
- attackerStats (CharacterStats): Reference to the CharacterStats script attached to 'character'.
- defenderStats (CharacterStats): Reference to the CharacterStats script attached to 'character2'.

Description: This script asserts that the health of 'character2' after the Attack method is called by 'character' is less than its original value.

Teardown: All the variables created in the Setup section are removed.

6.1.2 Test_CharacterStats

Purpose Verify that the TakeDamage() method functions properly.

Setup: The following variables are needed for this test:

- GameManager (GameObject): Required for Unity Test
- player (GameObject): A reference to the player object
- GunUI (GameObject): The Gun ammo count UI
- AxeUI (GameObject): The Axe UI element of the canvas
- stats (CharacterStats): The CharacterStats script attached to the player.

Description: This script asserts that the health level (which is initially 100) is equal to 49 after the *TakeDamage* method is called and a damage of 51 is caused.

Teardown: All the variables created in the Setup section are removed.

6.1.3 Test_ConsumableItem

Purpose Verify that the Use() method functions properly.

Setup: The following variables are needed for this test:

- GameManager (GameObject): Required for Unity Test
- player (GameObject): A reference to the player object
- GunUI (GameObject): The Gun ammo count UI
- AxeUI (GameObject): The Axe UI element of the canvas
- item1 (ConsumableItem): A reference to an instance of a ConsumableItem script.

Description: This script asserts when item1 is equipped, the inventory is not empty, and after the Use() method is called on item1, item1 is 'consumed' and the inventory is now empty.

Teardown: All the variables created in the Setup section are removed.

6.1.4 Test_DayLightController

Purpose Verify that the 'Directional Light' GameObject rotates periodically.

Setup: The following variables are needed for this test:

- daylightobject (GameObject): A reference to the Directional Light object.

Description: This script asserts that the rotation of the Directional Light object at a certain time differs from the rotation of the Directional Light object after 2 seconds.

Teardown: All the variables created in the Setup section are removed.

6.1.5 Test_EquipmentItem

Purpose Verify that the Use() method functions properly.

Setup: The following variables are needed for this test:

- GameManager (GameObject): Required for Unity Test
- player (GameObject): A reference to the player object
- GunUI (GameObject): The Gun ammo count UI
- AxeUI (GameObject): The Axe UI element of the canvas
- item1 (EquipmentItem): A reference to an instance of a EquipmentItem script.

Description: This script asserts when item1 is not equipped, the inventory is not empty, the equipment UI is empty, and that after the Use() method is called on item1, item1 is 'equipped' and the inventory is now empty, whereas the equipment UI is no longer empty.

Teardown: All the variables created in the Setup section are removed.

6.1.6 Test_EquipmentManager

Purpose Verify that the Equip() method functions properly.

Setup: The following variables are needed for this test:

- GameManager (GameObject): Required for Unity Test
- player (GameObject): A reference to the player object
- GunUI (GameObject): The Gun ammo count UI
- AxeUI (GameObject): The Axe UI element of the canvas
- InventoryUI (GameObject): The Inventory UI element of the canvas
- item1 (EquipmentItem): A reference to an instance of a EquipmentItem script.
- item1 (EquipmentItem): An item with the name "ChestArmor", that has the EquipmentItem script attached as a component.

- item2 (EquipmentItem): An item with the name "Cloak", that has the EquipmentItem script attached as a component.
- item3 (EquipmentItem): An item with the name "RangeWeapon", that has the EquipmentItem script attached as a component.

Description: This script asserts when the Equip() method is called on item1, item2 and item3, they are all assigned to a different slot in the equipment UI.

Teardown: All the variables created in the Setup section are removed.

6.1.7 Test_Gun

Purpose Verify that the shoot() and reload() methods functions properly.

Setup: The following variables are needed for this test:

- bullet (GameObject): An instance of a 'bullet' gameObject
- cam (GameObject): A reference to the camera in the game.
- GameManager (GameObject): Required for Unity Test
- PlayerUI (GameObject): A reference to the player UI element of the canvas
- GunUI (GameObject): The Gun ammo count UI
- AxeUI (GameObject): The Axe UI element of the canvas
- rangeWeapon (EquipmentItem): A reference to an instance of a EquipmentItem object with the Gun script attached to it.

Description: This script asserts when the shoot() method is called on the gun, a bullet gameObject is instantiated and that the value of the ammo in the gun is reduced by 1. This script also asserts that when the reload() method is called, the value of the ammo in the gun is reset back to the max value of 7.

Teardown: All the variables created in the Setup section are removed.

6.1.8 Test_Inventory

Purpose Verify that the Add(), InventoryEquipmentConsumable() and RemoveFromInventory() methods functions properly.

Setup: The following variables are needed for this test:

- GameManager (GameObject): Required for Unity Test
- player (GameObject): A reference to the player UI element of the canvas
- GunUI (GameObject): The Gun ammo count UI
- AxeUI (GameObject): The Axe UI element of the canvas
- item1 (EquipmentItem): A reference to an instance of a EquipmentItem object with the Gun script attached to it.

Description: To test the Add() method, this script asserts that when item1 is added to the inventory using the Add() methods, the inventory contains item1 and is not empty.
To test the InventoryEquipmentConsumable() method, this script asserts that before calling the InventoryEquipmentConsumable() method, the inventory contains item1, but not after calling the method.
To test the RemoveFromInventory() method, this script asserts that when this method is called, item1 exists but is independent from the inventory (the inventory does not contain item1).

Teardown: All the variables created in the Setup section are removed.

6.1.9 Test_ItemStore

Purpose Verify that the Interact() method functions properly.

Setup: The following variables are needed for this test:

- GameManager (GameObject): Required for Unity Test
- player (GameObject): A reference to the player UI element of the canvas
- GunUI (GameObject): The Gun ammo count UI
- AxeUI (GameObject): The Axe UI element of the canvas
- GameItem (GameObject): An instance of a "HeadArmor" prefab.
- item (EquipmentItem): A reference to a "HeadArmor" object which shall be residing in the inventory.

Description: This method asserts that when the Interact() method is called, the inventory contains item and GameItem does not exist anymore.

Teardown: All the variables created in the Setup section are removed.

6.1.10 Test_PlayerStats

Purpose Verify that the OnEquipmentChanged() and Eat() methods function properly.

Setup: The following variables are needed for this test:

- GameManager (GameObject): Required for Unity Test
- player (GameObject): A reference to the player UI element of the canvas
- GunUI (GameObject): The Gun ammo count UI
- AxeUI (GameObject): The Axe UI element of the canvas
- stats (PlayerStats): A reference to the PlayerStats script attached to 'player'.
- item1 (ConsumableItem): A ConsumableItem instance.
- item2 (ConsumableItem): A ConsumableItem instance.

Description: To test the Eat() method, this script asserts that the value of 'curHealth' for 'stats' before and after calling the Eat() method is different, and that the value of 'curHealth' after the Eat() method has been called is greater than what it was before by a difference of the value of the 'healthModifier' variable for 'item1'. In this test, 'item1' has the name 'Apple'.
To test the OnEquipmentChanged() method, this script asserts that when 'item1' is equipped, the values of armor and damage stats for the player are increased by the values of the defence and attack modifiers associated with 'item1' respectively. Also, when item1 is swapped with item2 (by calling the OnEquipmentChanged() method), the values of armor and damage stats for the player are updated by the values of the defence and attack modifiers associated with 'item2' respectively. For this test, 'item1' and 'item2' are named 'Axe' and 'RangeWeapon' respectively.

Teardown: All the variables created in the Setup section are removed.

6.1.11 Test_Stat

Purpose Verify that the AddToStat(), GetValue() and RemoveFromStat() methods function properly.

Setup: The following variables are needed for this test:

- GameManager (GameObject): Required for Unity Test
- player (GameObject): A reference to the player UI element of the canvas
- GunUI (GameObject): The Gun ammo count UI
- AxeUI (GameObject): The Axe UI element of the canvas
- damage (Stat): A reference to the Stat script attached to 'player'.

Description: To test the AddToStat() method, this script asserts that when a value is added to 'stat' by calling the AddToStat(), stat then contains that value.
To test the GetValue() method, this script asserts that when two values are passed to 'stat' using the AddToStat() method, GetValue() then returns the

sum of those two values.
To test the RemoveFromStat() method, this script asserts that when 4 values are added to 'stat' using the AddToStat() method, and then one value (say 'x') is removed from 'stat' using the RemoveFromStat() method, 'stat' does not contain 'x'.

Teardown: All the variables created in the Setup section are removed.

6.2 ~~Unit testing of output files~~

~~Currently, the implementation does not generate any output files, so no tests are necessary. As a design choice, it was decided to reset the game when the player dies to give the development kit more of a survival theme. Also since this is just a dev kit and not a fully fledged video game, saving the game and generating output files is not necessary. If any output files are added, this section will be updated and test cases will be added.~~

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

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

7.2 Usability Survey Questions?

In order to complete manual integration testing, Google Surveys will be given out to a set number of people with instructions to play our game, and provide relevant feedback in the form. Sample questions include:

1. On a scale of 1-5, rate the functionality and fluidness of the enemy AI system. Provide any relevant comments, bug reports, or other feedback below.

2. On a scale of 1-5, rate the functionality and fluidness of the combat system. Provide any relevant comments, bug reports, or other feedback below.

3. On a scale of 1-5, rate the functionality and fluidness of the inventory/equipment system. Provide any relevant comments, bug reports, or other feedback below.

4. On a scale of 1-5, rate the functionality and fluidness of the movement system. Provide any relevant comments, bug reports, or other feedback below.

5. On a scale of 1-5, rate the functionality and fluidness of the environment and picking up items. Provide any relevant comments, bug reports, or other feedback below.