SE 3XA3: Test Plan
Zombie Survival Kit

Team #, Team Name
Mohammad Hussain hussam17
Brian Jonatan jonatans
Shivaansh Prasann prasanns

October 26, 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
Date 2	1.1	Notes

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.

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.

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 Func. Req. 1: The player must be able to move forward, left, down, and right using the WASD keys

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 the playable map, and a tree will be initialized at a position beside 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, 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 Func. Req. 2: The player must be able to lookin 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 the playable map, and a tree will be initialized at a position beside 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 as a marker, the test is concluded and is considered to have passed. If the tree stays at the same position on screen while the mouse is moving, the test is considered to have failed.

3.1.3 Func. Req. 3: 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

   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.

3.1.4 Func. Req. 4: Zombie enemies must start attacking the player when they come within a certain radius

Attack Radius

1. Attack-Radius-A-1

   Type: Automated

   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

   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.

3.1.5 Func. Req. 5: Zombie enemies must follow the player if they start running away while the zombie is in it's attack state

Zombie Follows Player

1. Zombie-Follows-Player-A-1

   Type: Automated

   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

   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.

3.1.6 Func. Req. 6: If the player runs past a certain radius from the zombie's original spawn loaction, the zombie must return to their circle

Zombie Returns to 'Spawn' Circle Center

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

   Type: Automated

   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

   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.

3.1.7 Func. Req. 7: Different types of zombies must have different statistics (health, damage,etc.)

Zombie Stats

1. Zombie-Stats-A-1

   Type: Automated

   Initial State: All different zombie enemy type objects are initialized.
   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 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.

3.1.8 Func. Req. 8: Zombies must have randomly generated chance of dropping items the player can use or consume when killed.

Appearing Drops

1. Appearing-Drops-A-1

   Type: Automated

   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

   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.

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   Func. Req. 10: The player must be able to pick up tiems they are looking at with the 'interact' key

Pickup Drops

1. Pickup-Drops-A-1

Type: Automated

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

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 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.

3.1.11 Func. Req. 11: The player must be able to access their inventory by pressing the 'invenotry' key

Inventory Access

1. Inventory-Access-A-1

   Type: Automated

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

   Input: void

   Output: void or assertion error

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 thatthe 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.

3.1.12 Func. Req. 12: The player must by able to equip, consume, delete, and move around items in their inventory using the mouse.

Use Items Move Items

1. Use-Move-Items-M-1

   Type: 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

   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 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 the item is in the center of screen and will press the 'interact' key on the item to pick it up. The user will do this for both items. The user will then press the 'inventory' 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 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 equpment 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. The user will check if the item has been removed from the player's inventory and has 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.

3.1.13   Func. Req. 13: The player must by able to fire/use their equipped weapon by pressing the left mouse button.

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.

3.1.14 Func. Req. 14: If the player has a firearm equipped, they must be able to aimdown sights using the right mouse button.

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.

14

3.1.15 Func. Req. 15: 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.

3.1.16 Func. Req. 16: The player must lose health when hit by a zombie

Zombie Attack

1. Zombie-Attack-A-1

   Type: Automated

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

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.

3.1.17   Func. Req. 17: Zombies must lose health when hit by the player

Player Attack

1. Player-Attack-A-1

   Type: Automated

   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

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.

3.1.18 Func. Req. 18: Players and zombies must die when they reach 0 health

Death

1. Death-A-1

   Type: Automated

   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.

   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 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.

3.1.19   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.

3.2   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

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 intialized 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/FramesPerSeco
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

   Type: Functional

   Initial State: void

   Input/Condition: Checking to see that the space needed to download the playable executable of 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 is required to download the playable executable of Zombie Survival Kit. If space 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. 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.

2. 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.

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.

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. Although behind on that aspect, 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 benefiecial 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.

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. 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.

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.