Unity Project

Test Plan Document

Members: Anthony Jamora, Jonny Nabors, Thomas Burke

Draft 1

*CSC 4420*

*Unity Group*

*2015*

[1 Introduction 4](#_Toc414704056)

[1.1 Testing Overview 4](#_Toc414704057)

[1.2 Test Flow 4](#_Toc414704058)

[1.3 Glossary 4](#_Toc414704059)

[2 Compatibility Testing 5](#_Toc414704060)

[2.1 Test Approach 5](#_Toc414704061)

[2.2 Items to be Tested 5](#_Toc414704062)

[2.3 Test Risks/Issues 5](#_Toc414704063)

[2.4 Test Environmental Needs 5](#_Toc414704064)

[2.5 Test Pass/Fail Criteria 5](#_Toc414704065)

[2.6 Test Entry/Exit Criteria 5](#_Toc414704066)

[2.7 Test Suspension/Resumption Criteria 6](#_Toc414704067)

[2.8 Test Cases 6](#_Toc414704068)

[3 Functional Testing 7](#_Toc414704069)

[3.1 Test Approach 8](#_Toc414704070)

[3.2 Items to be Tested 8](#_Toc414704071)

[3.3 Test Risks/Issues 12](#_Toc414704072)

[3.4 Test Pass/Fail Criteria 12](#_Toc414704073)

[3.5 Test Entry/Exit Criteria 12](#_Toc414704074)

[3.6 Test Suspension/Resumption Criteria 13](#_Toc414704075)

[3.7 Test Cases 13](#_Toc414704076)

[4 Integration Testing 30](#_Toc414704077)

[4.1 Test Approach 30](#_Toc414704078)

[4.2 Test Runner Functionality 30](#_Toc414704079)

[4.3 Items to be Tested 32](#_Toc414704080)

[4.4 Test Risks/Issues 32](#_Toc414704081)

[4.5 Test Pass/Fail Criteria 32](#_Toc414704082)

[4.6 Test Entry/Exit Criteria 32](#_Toc414704083)

[4.7 Test Deliverables 32](#_Toc414704084)

[4.8 Test Suspension/Resumption Criteria 32](#_Toc414704085)

[5 Performance Testing 32](#_Toc414704086)

[5.1 Test Approach 32](#_Toc414704087)

[5.2 Items to be Tested 32](#_Toc414704088)

[5.3 Test Risks/Issues 34](#_Toc414704089)

[5.4 Test Pass/Fail Criteria 34](#_Toc414704090)

[5.5 Test Entry/Exit Criteria 34](#_Toc414704091)

[5.6 Test Deliverables 34](#_Toc414704092)

[5.7 Test Suspension/Resumption Criteria 34](#_Toc414704093)

[6 User Acceptance Testing 34](#_Toc414704094)

[6.1 Test Approach 34](#_Toc414704095)

[6.2 Items to be Tested 34](#_Toc414704096)

[6.3 Test Risks/Issues 34](#_Toc414704097)

[6.4 Test Pass/Fail Criteria 34](#_Toc414704098)

[6.5 Test Entry/Exit Criteria 34](#_Toc414704099)

[6.6 Test Deliverables 34](#_Toc414704100)

[6.7 Test Suspension/Resumption Criteria 34](#_Toc414704101)

[7 References 34](#_Toc414704102)

[8 Document Revision History 34](#_Toc414704103)

[9 Appendix 35](#_Toc414704104)

1. Introduction

The purpose of this document is to give an overview of both how the Unity3D testing suite allows automated and manual test cases to be performed as well as how they specifically apply to the Unity Capstone game.

* 1. Testing Overview

The Unity Game will utilize the Integration and Assertion Test Components for test script creation and automation. Integration tests are entire GameObjects that exist in a hierarchy where each TestComponent that is a part of the GameObject is tested sequentially. Assertion Components are a code-free way of creation conditions that are always expected to be true and testing for failure, throwing an exception when a failure occurs. These two testing methods will be integrated into the various test cases and scenarios outlined in this document.

* 1. Test Flow

A test starts once the Test Object is enabled. The test may finish its run in multiple ways:

* Function Testing.Pass() is called. This will successfully finish the test
* Function Testing.Fail() is called. This will fail the test
* Execution times out. This can happen when none of the above functions is called within a specified period of time (you can set the timeout value per test).
* An unhandled exception is thrown.
* An expected exception is thrown (Expect exception must be checked)
* Every Assertion Component on objects under tests is checked at least once ( the "Succeed after all assertions are executed" option needs to be selected)
  1. Glossary

**Demo** - Playable demonstration of core gameplay mechanics

**Invariant** – Condition always expected to be true

**NPC –** Non-player character

**UI** – User interface

**HUD** – Heads-Up-Display

**GameObject** – Any hierarchical object in Unity

# Compatibility Testing

The Unity Capstone game is designed to run on personal computers using the Windows 7 or Windows 8 operating systems that have at least 4GB of RAM available to them, an Intel i3 processor, and an Intel HD 4000 series integrated graphics processor. With this combination of specifications, the Unity Capstone game should never drop below 20 frames per second using the lowest graphic quality settings allowed by the game.

* 1. Test Approach

The Unity Capstone game will be built and tested on both a Windows 7 and Windows 8 personal computer using the build settings outlined in the design document. It will have all of the game’s main assets and will be a build representative of the playable game’s final components.

* 1. Items to be Tested

|  |  |  |
| --- | --- | --- |
| ID | Item to Test | Test Description |
| CT\_1 | Game Installation | The game player is able to successfully install the game on their personal computer. |
| CT\_2 | Game Execution | Upon successful installation, the game player is able to launch the game via the executable (.exe) by double-clicking the icon and is presented with the Splash/Loading Screen |

* 1. Test Risks/Issues

The game player must have the minimum space available on their hard drive. The exact number will be visible during the game installation process when the installer package is asking for the user to choose an install destination on their machine.

* 1. Test Environmental Needs

All testing will be completed on both a personal computer with Windows 7 as its operating system and one with Windows 8. Both machines must meet the minimum specifications outlined in this and the requirements document.

* 1. Test Pass/Fail Criteria

Test Pass when 100% of the test cases are completed and fully working on Windows 7 and Windows 8 personal computers.

Test Fail on any unsuccessful completion of the Compatibility Test Cases.

* 1. Test Entry/Exit Criteria

Entry criteria for the test entry is as follows:

* Verify the test devices are ready to use by determining that the machine has either Windows 7 or Windows 8 as the operating system.
* Confirm that the target machine matches the minimum specifications outlined in this document as well as the Requirements Document.

The exit criteria will be when all tests have been completed successfully.

* 1. Test Suspension/Resumption Criteria

As soon as any test case fails, the testing will suspend. Upon successful bug & error fixing and the test case in question is executed again without fault, the test cases will resume.

* 1. Test Cases

|  |  |
| --- | --- |
| ID | CT\_1 |
| Item to Test | Game Installation |
| Pre-Conditions | All files required for game installation are present on the user’s machine  User must have the Windows 7 or Windows 8 Operating System installed on their machine |
| Test Steps | Repeat the below process for both Windows 7 and Windows 8 personal computers:   1. Locate game installer package Unity\_Game.msi 2. Double click the installer package Unity\_Game.msi 3. Choose the installation destination on the PC 4. Follow the steps presented in the installer until completion 5. Click the Finish button at the end of the installation steps |
| Expected Results | The installation package has created the game folder at the specified location via the installer package. In this game folder is the Unity\_Game.exe executable file to launch the game application. |
| Priority | High |
| Pass/Fail |  |

|  |  |
| --- | --- |
| ID | CT\_2 |
| Item to Test | Game Execution |
| Pre-Conditions | The game installation has been completed successfully and all of the necessary game files are present on the machine.  User must have the Windows 7 or Windows 8 Operating System installed on their machine. |
| Test Steps | Repeat the below process for both Windows 7 and Windows 8 personal computers:   1. Locate Unity\_Game.exe executable file in the game folder created during CT\_1 2. Double click the Unity\_Game.exe executable file |
| Expected Results | The Unity\_Game.exe executable file should, when double-clicked, launch the game application and the game player should see the Splash/Loading screen with the Unity logo. |
| Priority | High |
| Pass/Fail |  |

1. Functional Testing

The Unity Capstone functional requirements must all be tested before the application can be considered successful. Each functional test corresponds with at least one functional requirement outlined in the Requirements Document. Functional tests will be done using Unity’s Integration and Assertion Test tools, utilizing at least one these as the way to determine a success/fail.

* 1. Test Approach

Each functional test case will either be completed using an Integration or an Assertion Test provided in Unity’s suite of test tools. The functional tests can use either test or a mix of them based on the required functionality to test.

* 1. Items to be Tested

|  |  |  |
| --- | --- | --- |
| ID | Item to Test | Test Description |
| FT\_1 | Main Menu Screen | The game player is able to view and interact with the Main Menu Screen. Corresponding Functional Requirement: FR4. |
| FT\_2 | Options Screen | From the main menu, the game player is able to enter the options. Corresponding Functional Requirement: FR5. |
| FT\_3 | Change Resolution | From the options menu, the game player is able to change the available resolution options and see that the game responds and sizes accordingly. Ensure entrance and exit of submenu. Corresponding Functional Requirement: FR6. |
| FT\_4 | Adjust Graphics Quality | From the options menu, the game player is able to change the available graphics options and see that the game graphical quality adjusts accordingly. Ensure entrance and exit of submenu. Corresponding Functional Requirement: FR7. |
| FT\_5 | Adjust Brightness Level | From the options menu, the game player can adjust the brightness level using the provided slider GameObject and should see the game’s brightness adjust accordingly. Ensure entrance and exit of submenu. Corresponding Functional Requirement: FR9 |
| FT\_6 | View Control Scheme | From the options menu, the game player is able to select the option to view the control scheme and is then able to see the static image depicting each available keyboard control. Ensure entrance and exit of submenu. Corresponding Functional Requirement: FR10 |
| FT\_7 | View Game Credits | From the options menu, the game player is able to select the option to view the list of individuals that worked on the game. Corresponding Functional Requirement: FR11 |
| FT\_8 | Start Game | From the main menu, the game player should be able to select the ‘New Game’ option and be brought to the playable game world. Corresponding Functional Requirement: FR8 |
| FT\_9 | Exit to Desktop | From the main menu, the game player should be able to select the ‘Exit to Desktop’ option and the game application should terminate and return the game player to their desktop. Corresponding Functional Requirement: FR12 |
| FT\_10 | Game Asset Load Test | When the game has been started by the game player, test that each required GameObject is loaded into the scene. Corresponding Functional Requirement: FR13, FR14, FR16, FR17, FR18, FR19, FR21, FR30, FR38, FR39, FR40, FR41, FR42, FR43, FR44 |
| FT\_11 | Test Audio Components | When the game is begun, test to see that the game audio components have loaded in and are successfully playing. Corresponding Functional Requirement: FR15 |
| FT\_12 | Control Scheme Test | When the game has been started by the game player and the assets tested in FT\_10 have succeeded, test to see if the appropriate control scheme and controllers have been attached to the player GameObject. Corresponding Functional Requirement: FR20 |
| FT\_13 | Player Movement Test | When the game has been started by the game player and the assets testing in FT\_10 have succeeded, test to see if the player GameObject is able to successfully move in the game world. This should test the appropriate animations. Corresponding Functional Requirement: FR22, FR23 |
| FT\_14 | Player Attack & Block Test | When the game has been started by the game player and the assets testing in FT\_10 have succeeded, test to see if the player GameObject is able to successfully make attacks and blocks in the game world. This should test the appropriate animations and stamina losses. Corresponding Functional Requirement: FR24, FR25 |
| FT\_15 | Player Health Test | When the game has been started by the game player and the assets testing in FT\_10 have succeeded, test to see if the player GameObject is able to successfully gain and lose health as well as gain health by drinking a potion. This should test the appropriate animations and HUD elements. Corresponding Functional Requirement: FR26 |
| FT\_16 | Player Death | When the game has been started by the game player and the assets testing in FT\_10 have succeeded, test to see if the player GameObject will trigger the death animation and game end-game logic when the player’s health has reached zero. This should test the appropriate animations. Corresponding Functional Requirement: FR27 |
| FT\_17 | Player Essence Collection | When the game has been started by the game player and the assets testing in FT\_10 have succeeded, test to see if the player GameObject will collect the essences left on the ground by a dead enemy. Corresponding Functional Requirement: FR28 |
| FT\_18 | Player Climbing Test | When the game has been started by the game player and the assets testing in FT\_10 have succeeded, test to see if the player GameObject will be able to correctly interact with and climb up and down the ladder GameObject in the environment. Corresponding Functional Requirement: FR29 |
| FT\_19 | Enemy Spawn Location Test | When the game has been started by the game player and the assets testing in FT\_10 have succeeded, test to see if the enemy GameObjects are spawning at the correct location. Corresponding Functional Requirement: FR31 |
| FT\_20 | Enemy Combat Initiated Test | When the game has been started by the game player and the assets testing in FT\_10 have succeeded, test to see if the enemy GameObjects are successfully entering into the combat mode when the player GameObject comes within the minimum range to initiate combat. Corresponding Functional Requirement: FR32. |
| FT\_21 | Enemy Combat Deactivated Test | When the game has been started by the game player and the assets testing in FT\_10 have succeeded, test to see if the enemy GameObject will discontinue the in-combat game logic and return to their spawning location after the player GameObject has left the combat range of the enemy. Corresponding Functional Requirement: FR33 |
| FT\_22 | Enemy Basic Attack Test | When the game has been started by the game player and the assets testing in FT\_10 have succeeded, test to see if the enemy GameObject is successfully executing its basic attack function. This should include all corresponding animations and game logic. Corresponding Functional Requirement: FR34 |
| FT\_23 | Enemy Special Attack Test | When the game has been started by the game player and the assets testing in FT\_10 have succeeded, test to see if the enemy GameObject is successfully executing its special attack function. This should include all corresponding animations and game logic. Corresponding Functional Requirement: FR35 |
| FT\_24 | Enemy Death Test | When the game has been started by the game player and the assets testing in FT\_10 have succeeded, test to see if the enemy GameObject is dying when its health value reaches zero. This should include all corresponding animations and removal of the enemy GameObject. Corresponding Functional Requirement: FR36 |
| FT\_25 | Enemy Essence Drop Test | When the game has been started by the game player and the assets testing in FT\_10 have succeeded, test to see if the enemy GameObject is successfully dropping the appropriate number of essences. Corresponding Functional Requirement: FR37 |
| FT\_26 | Pause Game & In-Game Menu Test | When the game has been started by the game player and the assets testing in FT\_10 have succeeded, test to see if the game player is able to pause the game and enter its in-game menu by pressing the escape key. Corresponding Functional Requirement: FR45 |
| FT\_27 | In-Game Menu Exit Game Prompt | When the game has been started by the game player and the assets testing in FT\_10 and FT\_25 have succeeded, test to see if the in-game menu shows the pop up exit game confirmation GameObject on the screen when the game player selects ‘Exit to Desktop’ from the in-game menu. Corresponding Functional Requirement: FR46 |
| FT\_28 | Main Menu Exit Game Prompt | When the game has been started by the game player and the assets testing in FT\_1 have succeeded, test to see if the main menu shows the pop up exit game confirmation GameObject on the screen when the game player selects ‘Exit to Desktop’ from the in-game menu. Corresponding Functional Requirement: FR47 |
| FT\_29 | In-Game Menu Return to Main Menu Prompt | When the game has been started by the game player and the assets testing in FT\_10 and FT\_25 have succeeded, test to see if the in-game menu shows the pop up return to main menu confirmation GameObject on the screen when the game player selects ‘Main Menu’ from the in-game menu. Corresponding Functional Requirement: FR48 |

* 1. Test Risks/Issues

The only risks and issues for the Functional Tests are that the test machine must have the required minimum specifications as well as the Unity Test Tools installed alongside of the Unity 3D Framework. These tests are more memory intensive on the target machine than the game itself and the test user should be aware of the requirements to execute the tests.

* 1. Test Pass/Fail Criteria

The pass/fail criteria for the functional tests are as follows:

* If any assertions or integration tests fail, the test runner will continue on until a critical error is reached or the tests are finished.
* Any failed assertions or integrations will be output to the console.
* Any failed tests result in an overall fail of the functional test cases.
* If the functional tests return a failing case, further testing is suspended until the error at hand is fixed.
* After the errors have been resolved, the test runner will restart and test all components again.
* If no errors found, the functional test cases are successful.
  1. Test Entry/Exit Criteria

The criteria required for the entry/exit criteria is as follows:

* Confirm that the test environment is prepared
* Confirm that the test tools are installed to Unity
* Confirm that each integration and assertion test is in place and the corresponding GameObjects are in the scene
  1. Test Suspension/Resumption Criteria

Upon any reported failure to the console all further tests will be suspended until a proposed fix has been implemented to the code. Testing will then be resumed and the process is repeated until no errors are present.

* 1. Test Cases

|  |  |
| --- | --- |
| ID | FT\_1 |
| Item to Test | Main Menu Screen |
| Pre-Conditions | CT\_1 & CT\_2 have been successful. The game must currently be running after having opening the executable Unity\_Game.exe |
| Test Steps |  |
| Expected Results |  |
| Priority | High |
| Pass/Fail |  |

|  |  |
| --- | --- |
| ID | FT\_2 |
| Item to Test | Options Screen |
| Pre-Conditions | CT\_1, CT\_2, and FT\_1 must be successful. The game must be running and the game player must be at the Main Menu |
| Test Steps | 1. Execute *OptionsPress()* function from the MainMenu scene. This is a part of FT\_1. 2. Execute *BackPress()* function from the optionsCanvas Back Button script component 3. Read resulting output in console window to determine pass/fail. |
| Expected Results | Options menu shall appear then hidden and return to console window |
| Priority | High |
| Pass/Fail | Return message from Expected Results will result in a pass. Any message formatted as [“\_\_\_\_\_” not found] will result in a fail. |

|  |  |
| --- | --- |
| ID | FT\_3 |
| Item to Test | Change Resolution |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, and FT\_2 must be successful. The game must be running and the game player must be at the Options Menu |
| Test Steps | 1. Execute *VideoPress()* function from the optionsCanvas Video button script component. This is a part of FT\_2. 2. Execute *resolutionPress()* function from the videoCanvas Resolution Button script component 3. Execute *960x540Press()* function from the resoDrop Button script component 4. Read resulting output in console window to determine pass/fail 5. Loop for resolutions *1280x720, 1600x900, 1920x1080* using resolutionTest script |
| Expected Results | * + - 1. Video settings menu shall appear       2. Drop down menu shall appear       3. Pixel density shall change       4. Console window shall output result for corresponding resolution       5. Loop for remaining resolution settings |
| Priority | Medium |
| Pass/Fail | Return message from Expected Results will result in a pass. Any message formatted as [“\_\_\_\_\_” not found] will result in a fail for each loop (resolution value) |

|  |  |
| --- | --- |
| ID | FT\_4 |
| Item to Test | Adjust Graphics Quality |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, and FT\_2 must be successful. The game must be running and the game player must be at the Options Menu |
| Test Steps | 1. Execute *VideoPress()* function from the optionsCanvas Video button script component. This is a part of FT\_2. 2. Execute *qualityPress()* function from the videoCanvas Quality Button script component 3. Execute *FantasticPress()* function from the qualityDrop Button script component 4. Read resulting output in console window to determine pass/fail 5. Loop for quality settings, *Fastest, Fast, Simple, Good, and Beautiful,* using qualityTest script |
| Expected Results | * + - 1. Video settings menu shall appear       2. Quality Drop down menu shall appear       3. Game quality shall change       4. Console window shall output result for corresponding quality settings       5. Loop for remaining quality settings |
| Priority | Medium |
| Pass/Fail | Return message from Expected Results will result in a pass. Any message formatted as [“\_\_\_\_\_” not found] will result in a fail for each loop (quality value) |

|  |  |
| --- | --- |
| ID | FT\_5 |
| Item to Test | Adjust Brightness Level |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, and FT\_2 must be successful. The game must be running and the game player must be at the Options Menu |
| Test Steps | 1. Execute *VideoPress()* function from the optionsCanvas Video button script component. This is a part of FT\_2. 2. Execute *brightnessScroll()* function from the videoCanvas Brightness Scroll-Bar script component 3. Execute *dragLeft()* function from the brightness Test Script 4. Execute *dragRight()* function from brightness Test Script 5. Read resulting output in console window to determine pass/fail |
| Expected Results | 1. Video menu shall appear  2. The slider button shall change color  3. The game within the background shall decrease in brightness.  4. The game within the background shall increase in brightness. |
| Priority | Medium |
| Pass/Fail | Return message from Expected Results will result in a pass. Any message formatted as [“\_\_\_\_\_” not found] will result in a fail for and/or dragLeft(), dragRight() |

|  |  |
| --- | --- |
| ID | FT\_6 |
| Item to Test | View Control Scheme |
| Pre-Conditions | CT\_1, CT\_2, and FT\_1 must be successful. The game must be running and the game player must be at the Main Menu |
| Test Steps | 1. Execute *controlPress()* function from the Main Menu Controller script. This is a part of FT\_1. 2. Execute *BackPress()* function from the controlCanvas Back Button script component 3. Read resulting output in console window to determine pass/fail |
| Expected Results | 1. Controls Window shall appear and return to main menu |
| Priority | Low |
| Pass/Fail | Return message from Expected Results will result in a pass. Any message formatted as [“\_\_\_\_\_” not found] will result in a fail |

|  |  |
| --- | --- |
| ID | FT\_7 |
| Item to Test | View Game Credits |
| Pre-Conditions | CT\_1, CT\_2, and FT\_1 must be successful. The game must be running and the game player must be at the Main Menu |
| Test Steps | 1. Execute credits*Press()* function from the Main Menu Controller script. This is a part of FT\_1. 2. Execute *BackPress()* function from the creditsCanvas Back Button script component 3. Read resulting output in console window to determine pass/fail |
| Expected Results | 1. A new *Game Credits* window shall appear  2. The *Game Credits* window shall disappear and return to main menu |
| Priority | Low |
| Pass/Fail | Return message from Expected Results will result in a pass. Any message formatted as [“\_\_\_\_\_” not found] will result in a fail |

|  |  |
| --- | --- |
| ID | FT\_8 |
| Item to Test | Start Game |
| Pre-Conditions | CT\_1, CT\_2, and FT\_1 must be successful. The game must be running and the game player must be at the Main Menu |
| Test Steps | 1. Execute *playPress()* from the MainMenuController script. This is a part of FT\_1 2. Read resulting output in console window to determine pass/fail |
| Expected Results | 1. The main menu scene end and the Game Scene shall start |
| Priority | High |
| Pass/Fail | Return message from Expected Results will result in a pass. Any message formatted as [“\_\_\_\_\_” not found] will result in a fail |

|  |  |
| --- | --- |
| ID | FT\_9 |
| Item to Test | Exit to Desktop |
| Pre-Conditions | CT\_1, CT\_2, and FT\_1 must be successful. The game must be running and the game player must be at the Main Menu or the In-Game Menu. |
| Test Steps | 1. Execute *exitPress()* from the MainMenuController script. This is a part of FT\_1 2. Execute exitGame() from the MainMenuController 3. Read resulting output in console window to determine pass/fail |
| Expected Results | * + - 1. Exit warning canvas shall appear       2. Game shall close and all related processes |
| Priority | High |
| Pass/Fail | Return message from Expected Results will result in a pass. Any message formatted as [“\_\_\_\_\_” not found] will result in a fail to corresponding function |

|  |  |
| --- | --- |
| ID | FT\_10 |
| Item to Test | Game Asset Load Test |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, and FT\_8 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is part of FT\_8 2. Call AssetTest() function from AssetTest script with no arguments. 3. Read resulting output in console window to determine pass/fail. |
| Expected Results | “All Assets Loaded” returned to the console window. |
| Priority | High |
| Pass/Fail | Return message from Expected Results will result in a pass. Any message formatted as [“\_\_\_\_\_” not found] will result in a fail. |

|  |  |
| --- | --- |
| ID | FT\_11 |
| Item to Test | Test Audio Components |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, and FT\_8 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. 2. Call SoundTest() function from SoundTest script with no arguments. 3. Read resulting output in console window to determine pass/fail. |
| Expected Results | “All sound loaded and playing” returned to the console window |
| Priority | Medium |
| Pass/Fail | Return message from Expected Results will result in a pass. Any message formatted as [“\_\_\_\_\_” not found] will result in a fail. |

|  |  |
| --- | --- |
| ID | FT\_12 |
| Item to Test | Control Scheme Test |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps |  |
| Expected Results |  |
| Priority | High |
| Pass/Fail |  |

|  |  |
| --- | --- |
| ID | FT\_13 |
| Item to Test | Player Movement Test |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. 2. Reference the ThirdPersonController() script in the Assertion Test. 3. Call the Setup() function from the controller script with no arguments to initialize GameObject 4. Call the Start() function from the controller with no arguments and verify no output is sent to the console window 5. Call Update() function with no arguments to ensure animation is playing and that key presses are working correctly. There should be no output to the console window. 6. Call FixedUpdate() function with no arguments to test for animation and character position. Should run through case of “grounded” and “jumping” to make sure character is in bounds while jumping. No output to console window is expected. |
| Expected Results | No output sent to the console window. |
| Priority | High |
| Pass/Fail | Any error message in Unity’s console window is considered a failure. A blank console window is considered to be passing. |

|  |  |
| --- | --- |
| ID | FT\_14 |
| Item to Test | Player Attack & Block Test |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. |
| Expected Results |  |
| Priority | High |
| Pass/Fail |  |

|  |  |
| --- | --- |
| ID | FT\_15 |
| Item to Test | Player Health Test |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. |
| Expected Results |  |
| Priority | High |
| Pass/Fail |  |

|  |  |
| --- | --- |
| ID | FT\_16 |
| Item to Test | Player Death |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. |
| Expected Results |  |
| Priority | High |
| Pass/Fail |  |

|  |  |
| --- | --- |
| ID | FT\_17 |
| Item to Test | Player Essence Collection |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. |
| Expected Results |  |
| Priority | Menu |
| Pass/Fail |  |

|  |  |
| --- | --- |
| ID | FT\_18 |
| Item to Test | Player Climbing Test |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. |
| Expected Results |  |
| Priority | High |
| Pass/Fail |  |

|  |  |
| --- | --- |
| ID | FT\_19 |
| Item to Test | Enemy Spawn Location Test |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. 2. Call EnemyLocation() function from EnemyLocationTest() script. 3. All default enemy locations are listed in the script and will automatically will test to see if the enemy GameObjects are in one of these locations. 4. Read console output message |
| Expected Results | “All enemies are in their correct position.” |
| Priority | High |
| Pass/Fail | If any message returns to the console window in the format “Location \_\_ does not have an enemy GameObject” then the test is considered a failure. If “All enemies are in their correct position” is returned it has passed. |

|  |  |
| --- | --- |
| ID | FT\_20 |
| Item to Test | Enemy Combat Initiated Test |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. 2. Reference EnemyAI() script from the Assertion Test. 3. Call GetDistanceToPlayer() function with no arguments during the Update() function. 4. If ‘distanceToPlayer’ variable is ever below 0.6f, test to see if Chasing() function is called. 5. If Chasing() is ever called when the ‘distanceToPlayer’ variable is >= 0.6f, log an error message to the console. |
| Expected Results | No output to console window |
| Priority | Medium |
| Pass/Fail | If the console window returns “Chasing out of bounds” it is a failure. If there is no output to window, it is considered a pass. |

|  |  |
| --- | --- |
| ID | FT\_21 |
| Item to Test | Enemy Combat Deactivated Test |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. 2. Reference EnemyAI() script from the Assertion Test. 3. Call GetDistanceToPlayer() function with no arguments during the Update() function. 4. If ‘distanceToPlayer’ variable is ever above 0.6f, test to see if Patrolling() function is called. 5. If Patrolling() is ever called when the ‘distanceToPlayer’ variable is <= 0.6f, log an error message to the console. |
| Expected Results | No output to console window |
| Priority | Medium |
| Pass/Fail | If the console window returns “Patrolling out of bounds” it is a failure. If there is no output to window, it is considered a pass. |

|  |  |
| --- | --- |
| ID | FT\_22 |
| Item to Test | Enemy Basic Attack Test |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. 2. Reference EnemyAI() script from the Assertion Test. 3. Call GetDistanceToPlayer() function with no arguments during the Chasing() function. 4. If ‘distanceToPlayer’ variable is ever below 0.2f, test to see if Attacking() function is called. 5. If Attacking() is ever called when the ‘distanceToPlayer’ variable is >= 0.2f, log an error message to the console. 6. Test to see that BasicAttack() is called three times before SpecialAttack() 7. Log error to console if ‘basicAttackCounter’ variable is > 3. |
| Expected Results | No output to console window |
| Priority | Medium |
| Pass/Fail | If the console window returns “Attacking Out of Bounds” or “Attack Counter > 3” it is a failure. If there is no output to window, it is considered a pass. |

|  |  |
| --- | --- |
| ID | FT\_23 |
| Item to Test | Enemy Special Attack Test |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. 2. Reference EnemyAI() script from the Assertion Test. 3. Call GetDistanceToPlayer() function with no arguments during the Chasing() function. 4. If ‘distanceToPlayer’ variable is ever below 0.2f, test to see if Attacking() function is called. 5. If Attacking() is ever called when the ‘distanceToPlayer’ variable is >= 0.2f, log an error message to the console. 6. Test to see that SpecialAttack() is called once after BasicAttack() is called three times. 7. Log error to console if SpecialAttack() is ever called before ‘basicAttackCounter’ <3 |
| Expected Results | No output to console window |
| Priority | Medium |
| Pass/Fail | If the console window returns “Attacking Out of Bounds” or “Attack Counter < 3” it is a failure. If there is no output to window, it is considered a pass. |

|  |  |
| --- | --- |
| ID | FT\_24 |
| Item to Test | Enemy Death Test |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. 2. Reference EnemyAI() script from the Assertion Test. 3. Call EnemyHealth() function from controller. 4. Test to see that EnemyDeath() function gets called when the ‘enemyHealth’ variable reaches zero. 5. Test that the animation is triggered on EnemyDeath() being called and that after the animation completes, the enemy GameObject is destroyed. |
| Expected Results | No output to console window |
| Priority | Medium |
| Pass/Fail | If the console window returns “Enemy Died with more than 0 health” or “Animation Failed to Play” or “Enemy not Destroyed” it is a failure. If there is no output to window, it is considered a pass. |

|  |  |
| --- | --- |
| ID | FT\_25 |
| Item to Test | Enemy Essence Drop Test |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. 2. Reference EnemyAI() script from the Assertion Test. 3. Call EnemyDeath() function from controller. 4. Test that DropEssence() function is called at the end of the EnemyDeath() function lifecycle. 5. Verify that 5 essences are dropped when the enemy dies. |
| Expected Results | No output to console window |
| Priority | Medium |
| Pass/Fail | If the console window returns “Essences not Dropped” it is a failure. If there is no output to window, it is considered a pass. |

|  |  |
| --- | --- |
| ID | FT\_26 |
| Item to Test | Pause Game & In-Game Menu Test |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, and FT\_10 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. 2. Reference inGameMenuController() script from the Assertion Test. 3. Call Start() function from controller. 4. Test to see that the if(Input.GetKeyDown(KeyCode.Escape)) condition returns true when the Escape key is pressed. 5. If this function does not fire, log an error message to the console. |
| Expected Results | No output to console window |
| Priority | High |
| Pass/Fail | If the console window returns “Failed to register Escape key” it is a failure. If there is no output to window, it is considered a pass. |

|  |  |
| --- | --- |
| ID | FT\_27 |
| Item to Test | In-Game Menu Exit Game Prompt |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, FT\_10, and FT\_26 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. 2. Reference inGameMenuController() script from the Assertion Test. 3. Call Start() function from controller. 4. Call the ExitPress() function with no arguments from the controller. 5. Verify the pop-up exit confirmation dialog window is active from the exitButton.enabled line. |
| Expected Results | No output to console window |
| Priority | Medium |
| Pass/Fail | If the console window returns “Failed open dialog window” it is a failure. If there is no output to window, it is considered a pass. |

|  |  |
| --- | --- |
| ID | FT\_28 |
| Item to Test | Main Menu Exit Game Prompt |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, FT\_10, and FT\_26 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. 2. Reference menuController() script from the Assertion Test. 3. Call Start() function from controller. 4. Call the ExitPress() function with no arguments from the controller. 5. Verify the pop-up exit confirmation dialog window is active from the exitButton.enabled line. |
| Expected Results | No output to console window |
| Priority | Medium |
| Pass/Fail | If the console window returns “Failed open dialog window” it is a failure. If there is no output to window, it is considered a pass. |

|  |  |
| --- | --- |
| ID | FT\_29 |
| Item to Test | In-Game Menu Return to Main Menu Prompt |
| Pre-Conditions | CT\_1, CT\_2, FT\_1, FT\_8, FT\_10, and FT\_26 must be successful. The game must be running and the game player has selected ‘New Game’ from the Main Menu. |
| Test Steps | 1. Execute EnterGame() function from the MainMenu scene. This is a part of FT\_8. 2. Reference inGameMenuController () script from the Assertion Test. 3. Call Start() function from controller. 4. Test to see that the if(Input.GetKeyDown(KeyCode.Escape)) condition returns true when the Escape key is pressed. 5. Verify that the mainMenu.enabled = true condition is met. |
| Expected Results | No output to console window |
| Priority | Medium |
| Pass/Fail | If the console window returns “Failed to return to Main Menu” it is a failure. If there is no output to window, it is considered a pass. |

1. Integration Testing
   1. Test Approach

Integration tests are designed be run in a separate scene than the main game. In the integration tests, a Test Object is a GameObject in the scene that has a TestComponent attached to it. Everything under the Test Object in the hierarchy is considered to belong to this test. Any object not under a Test object will be common for every test on the scene such as the ocean or the floor. Only one test will be active at a time1.

When the test is run, the following steps are executed:

1. Play mode is enabled
2. The first test becomes active
3. Wait until the test has finished or timeout has occurred
4. The current active test gets disabled
5. If there are more tests, enable the next test and continue from step 3
6. Report results and finish test run
   1. Test Runner Functionality

The Integration Tests will utilize Unity’s Integration Test Runner. This Test Runner will automate the execution of the testing process. The flow is as follows1:

1. Run all tests in the scene (excluding ignored tests)
2. Run selected test(s).
3. Create a new test - creates new test object on the scene
4. Options - options for working with Integration Tests
   1. Add GameObjects under selected test - when selected, when you add a new object to the scene it will be automatically placed under the test GameObject instead of the hierarchy root
   2. Block UI when running - when selected, a dialog will appear during test execution
5. Test Filter - will filter out tests where name does not contain the string
   1. Show succeeded - show tests that succeeded
   2. Show failed - show tests that failed
   3. Show ignored - show tests that are ignored
   4. Show not ran - show tests that haven’t been run
6. Test list - list of all tests available in the scene
7. Test log and exception messages
8. Test name - name of the test
9. Included platform - on what platform the test should included
10. Timeout - number of second after the test will timeout
11. Ignored - ignore the test when running all tests
12. Succeed after all assertions are executed - select if the test should finish after all assertions from Game Object in the test got checked at least once.
13. Expect exception - the test will not fail if an exception if thrown.
14. Expected exception list - a list of exception that will not fail the test when thrown. Separate the exceptions with comma (","). Derived types from types on the list will also be considered as expected. If the list is empty, any exception type will be accepted.
15. Succeed when exception is thrown - the test will succeed when one of the excepted exceptions is thrown.
    1. Items to be Tested
    2. Test Risks/Issues
    3. Test Pass/Fail Criteria
    4. Test Entry/Exit Criteria
    5. Test Deliverables
    6. Test Suspension/Resumption Criteria
16. Performance Testing

The Unity Capstone game must perform well given the minimum technical specifications outlined in section 2 of this document and the Minimum Requirements section of the Requirements document. The main thing to be tested outside of the functional requirements outlined in section 2 are the non-functional requirements as well as making sure the framerate of the game never drops below 20 frames per second on a machine with the minimum specifications.

* 1. Test Approach

Each test will be performed either as an Assertion or Integration test with some corresponding script attached to it. The script will be the main driver that outputs an error message to Unity’s console window whenever an exception has occurred or invariant has returned an undesired value.

* 1. Items to be Tested

|  |  |  |
| --- | --- | --- |
| ID | Item to Test | Test Description |
| PT\_1 | Frames Per Second Test | The game should never drop below 20 frames per second during normal execution of the game loop. |
| PT\_2 | Item Load Benchmarks | When loading into the game from the executable, the load times of various game objects must be benchmarked and load within the desired amount of time. Corresponding Non-Functional Requirement: NFR1, NFR5 NFR6, NFR7, NFR8, NFR14 |
| PT\_3 | Menu Response Times | During the menu operations in the game, the response time between selecting various menu options should be within the desired amount of time. Corresponding Non-Functional Requirements: NFR2, NFR3, NFR4 |
| PT\_4 | Character Reaction Times | During gameplay, the player GameObject should react to happenings in the game world within the desired amount of time. Corresponding Non-Functional Requirements: NFR9, NFR10, NFR11 |
| PT\_5 | Enemy Reaction Times | During gameplay, the enemy GameObject should react to happenings in the game world within the desired amount of time. Corresponding Non-Functional Requirements: NFR9, NFR12, NFR13 |
| PT\_6 | Environment Reaction Times | During gameplay, the game world should react to happenings within the desired amount of time. Corresponding Non-Functional Requirements: NFR15, NFR16 |

* 1. Test Risks/Issues
  2. Test Pass/Fail Criteria
  3. Test Entry/Exit Criteria
  4. Test Deliverables
  5. Test Suspension/Resumption Criteria

1. User Acceptance Testing
   1. Test Approach
   2. Items to be Tested
   3. Test Risks/Issues
   4. Test Pass/Fail Criteria
   5. Test Entry/Exit Criteria
   6. Test Deliverables
   7. Test Suspension/Resumption Criteria
2. References

|  |  |  |  |
| --- | --- | --- | --- |
| Doc Number |  | Doc Version | Doc Name & Location |
| 1 |  | 1 | [Integration Tests in Unity](https://bitbucket.org/Unity-Technologies/unitytesttools/wiki/IntegrationTestsRunner) |
| 2 |  | 1 | [Assertion Component](https://bitbucket.org/Unity-Technologies/unitytesttools/wiki/AssertionComponent) |
|  |  |  |  |

1. Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Author | Changes |
| 1.1 | 3/17/2015 | Jonathan Nabors | Initial Draft |
| 1.2 | 2/22/2015 | Jonathan Nabors | Release Plan, Input Module, Script Flowchart |

1. Appendix

Material including referenced documentation the web or elsewhere, as well as alternative designs or items/ideas for future improvements.