CMPT 371 – Team 3 Testing Document

Virtual Reality Medical Imaging Software with Luxsonic Technologies Inc.

1 Introduction

The Testing Plan outlines what types of tests we will be running and how we will be running them each iteration. This document also contains the delta changes from the last iteration as well as information about our shift in testing and the alternatives that we will be providing. We outline the smoke tests, system tests, and integration tests along with what is currently implemented and how they are being performed. Additionally, regression testing will be documented as issues in the code base arise and fixes are implemented. All tests will be within the folder Assets/UnityTestTools/UnitTests/Editor because of how Unity Test Tools searches for tests in that specific folder and does not contain folders to allow for Smoke, System, or regression Testing. There is however a folder for Integration testing, though that is through the Unity editor by creating testing objects and attaching fake scripts to them. We will not be doing integration testing using this method as every time we make a change we would have to manually update these test objects and their scenes.User testing practices are outlined in this document along with some of the results, findings, and solutions of our user testing sessions. Acceptance testing methods and practices are documented here along with any important notes or changes to them. Lastly, this document will contain sections for coverage testing, though we will not be implementing any, as well as our plan for a testing matrix.
All tests will be written in this document with pseudo code using the design document as a reference. However, due to the limitations of the Unity editor, we will not be implementing and automating most all these tests with the exception of smoke tests. Instead, each test will be pair-reviewed where 2 testers will review the necessary code and decide if there is sufficient evidence to assume that the test will pass or fail. Occasionally, we will create a new test scene and create a script that is runnable only in that scene to manually execute some of these tests. These scenes and tests will be documented. The pair review sessions will be recorded in the “Pair Testing Reviews” document. On top of these reviews, testers will begin practicing more frequent manual acceptance testing as the developers work and recording the results of these on the appropriate wiki page. Lastly, we plan to follow a more strict issue tracking procedure in that only a tester may close an issue once they have reviewed it and deem it closed.

2 Delta Changes

2.1 ID 2 Changes

Many tests in ID1 are no longer useful since architecture of our system changed as a result of how certain unity features behaved with the Oculus Rift. Tests that were removed are removed because they no longer fit the design of the system, and other tests are still relevant, but had to be updated to reflect newer methods. Many of the tests in the document this iteration are being left as //TODO for ID3 as they are not currently relevant to our system. Lastly, we updated the

language of most every test to make it more clear on what we are testing and how we are testing it. Any other changes will be marked as such and given a description.

- Updated the Purpose description
- Added Delta Changes
- Added Testing Framework Issues
- Smoke Tests
    - Made verifying dependencies more detailed instead of just one line each
    - Deleted checking for the main menu
    - Check for registered default scene instead of workspace
    - Delete camera check because that is associated with the default scene check
    - Delete button alias check because that is associated with the Oculus libraries
- System Testing
    - Removed must/should/could haves
    - Removed Initialize the program
    - Removed Quit from the menu
    - Removed load DICOM file
    - Moved exit workspace to acceptance testing
    - Updated select images to display
    - Updated remove images from display
    - Updated adjust contrast
    - Updated adjust brightness
    - Updated apply filter
    - Updated zoom
    - Updated move
    - Added copy image
    - Added load file from file browser
    - Added next/prev image in display
- Integration Testing
    - Fixed grammar in introduction
    - Deleted everything else as a result of restructuring project. Will be in ID3 when we have pieces integrating.
- Regression
    - Added issues as regression tests
- Removed coverage testing for this iteration
- User Testing
    - Grammar
    - post -ask them after to see how they feel
    - look for studies and better questions
    - read Oculus warnings to them
- Coverage testing
    - Deleted for this iteration
- Acceptance Testing

        - Added methods for internal acceptance testing
- Testing Matrix
    - Deleted for this iteration

2.2 ID 3 Changes

- System Testing
    - Changed description and added test cases to Copy Image
    - Changed name of remove image to Remove copy
    - Updated description and added test cases to Remove copy
    - Changed description and added test cases to Load from file browser
    - Changed description and added test cases to select image in tray
    - Changed description and added test cases to remove image from tray
    - Deleted remove all images from tray
    - Changed description and added test cases to Next/prev Image in display
    - Updated description and added test cases to Change Contrast
    - Updated description and added test cases to Change brightness
    - Deleted move Copy tests
    - Updated description and added test cases to Change filter
    - Updated description and added test cases to Change zoom

- Integration Testing
    - Added all new tests to adhere to our new architecture
        - Test cases still need to be written

- Regression Testing
    - Added 2 regression tests from tracked bug reports.
        - Test cases still need to be written

- Coverage Testing
    - Re-added Coverage Testing section for this ID

- Testing Matrix
    - Re-added Testing matrix section

2.3 ID 4 Changes

- Updated introduction for ID4 changes
    - Also added descriptions for how our testing file structure works
- Added “Testing Shift” section
- Added issue #8 to “Testing Framework Issues”
- Updated System tests for the new UML and implementation
- Updated Integration tests for the new UML and implementation

- Added regression tests for resolved issues
- Overhauled the User testing with best practices and moved the user script to a separate section
- Updated Coverage Testing with new information on the matter
- Added a description for the future implementation of a testing matrix

3 Testing Shift

During iteration 4, our methods of testing have had to shift from automated testing to more manual testing as a result of our chosen frameworks being Unity and Visual Studio. In summary, the Unity Editor has very strict permissions that do not allow outside sources to access information that is crucial for us to be able to make interact objects with each other that do not exist in a scene. This limits our ability to do any amount of System or Integration testing. Among other issues, this also means that we have no ways of coverage testing our code, because of security exceptions, which trigger when trying to call certain methods within Unity in debug mode. Lastly, Unity lacks tools to properly mock objects which we have overcome by creating our own mocking methods. These issues are described in detail in the Testing Framework Issues below. As well, we will be contacting Unity about these issues once we have better formulated a description of our needs and the issues we are encountering. Any attempts to contact Unity along with responses will be documented or referenced in the Testing Framework Issues below.

4 Testing Framework Issues

Originally, we had researched that NUnit and NSubstitute were good testing and mocking frameworks to use for a C# project. With Unity, there is an official bundle called "Unity Test Tools" which integrates these 2 specific frameworks into a Unity project. This gave us a lot of reassurance that we had picked the right frameworks and made the mistake of thinking that using them would be easy. This section outlines all of the issues that we had setting up the testing framework and is in this document for the purpose of documentation and troubleshooting as we have already spent over 30 hours making this Framework function.

4.1 Issue #1: NUnit could not be found

The first issue that we ran into was that even though our unity project contained Unity Test Tools and Unity Test Tools contained NUnit, when we were writing scripts, they could not find NUnit. To try and fix this we tried importing NUnit into the project, but then we got an error

saying that there were 2 NUnits in the project. so we had to take out the one that we just put in because when we took out the one from Unity Test Tools, everything in the package broke. The second thing that we tried was to add the NUnit library to the Visual Studio and Monodevelop editors because as it turned out, Unity Test Tools only applied NUnit to the Unity editor, and not the C# editor.

Solution: add NUnit to Visual Studio and Monodevelop.

4.2 Issue #2: NUnit cannot be added to C# editor

Visual Studio/Monodevelop cannot add NUnit to the C# project because the project was not actually .NET Framework v3.5 or greater. It is a special type of project labelled Unity Full 3.5 and because Visual Studio and Monodevelop are set to compile to .Unity Full v3.5, they were not allowing us to add a .NET add-on for C# to a Unity project. On top of that, the best part was that the Visual Studio editor would not allow us to edit the project's settings to change it to .NET so that we could finally include NUnit.

Solution: We fixed this by compiling the project as Unity Full v3.5 and once it was compiled, the .csproj that the editor created could be unloaded from the project and then edited. When we edited the .csproj files we changed the version from Unity Full v3.5 to.NET Framework v4.5.2 (latest release to work with NUnit 3) and deleted a few other lines in the header that defined this as a Unity project. The project was now a .NET Framework v4.5.2 project and we then reloaded the .csproj files and went straight to NuGet extension manager to add NUnit to all of the project files. once NUnit was added we then rebuilt the project and NUnit had been added, it reestablished itself as a Unity Project, and our C# scripts now recognize the NUnit.Framework as usable.

4.3 Issue #3: NUnit does not discover tests

Now that NUnit is Usable, we can use the tests runner, but no tests are discovered.

Solution: NUnit3TestAdaptor had to be added to the C# editor using the same steps as the solution for Issue #2. Once we added the NUnit3TestAdapter to the project and then we finally ran tests, but they would only run sometimes and when they stopped working. When this happened the build would have to be cleaned and the version had to be reset to 4.5.2 manually every time (See Issue #5 for solution).

4.4 Issue #4: NUnit3TestAdapter does not discover tests

Some team members computers still could not see NUnit and discover tests even after applying fixes for issue #3.

Solution: This was fixed by reinstalling Unity with the editor as there were some packages that they exchange that were only put in when they were installed together using the newest Unity installer. We tried manually fetching these files and adding them in, but there were too many versions and serial numbers to deal with as a result of Visual Studio and Unity both being licensed products.

4.5 Issue #5: NUnit3TestAdapter STILL does not discover tests

We still could not run tests perfectly every time and had to consistently clean and reversion the build.

Solution was that we were using .NET Framework 4.5.2 and NUnit3TestAdapter only needed to use 3.6 and NUnitTestAdapter.

4.6 Issue #6: C# editor could not access because of Unity permissions

We could not access any Unity specific functions or classes because the C# editors did not have the permissions to access them from the Unity portion of the project.

Solution: This is fixed by running the tests from the Unity editor which makes it so that we have the proper permissions and it runs our tests from there. A small additional issue to this problem is that now some of our tests that passed in the C# editor are now failing in the Unity editor. luckily, Cloud Build builds the Unity project and runs tests from the Unity editor, so we just have to program so that our tests pass using the Unity Editor, not the C# editors.

4.7 Issue #7: Mock objects only return null objects

When using NSubstitute or Humble Objects the objects cannot properly mock methods and just throws exceptions.

Solution: We are creating our own empty game objects and adding the necessary components to them that have only blank values, thus essentially creating our own mock object.

4.8 Issue #8: we cannot test the methods of 2 GameObjects against each other, as that can only be done in a Scene

Unity requires the game to be running for objects to interact with each other. This means that we cannot test any methods that have dependencies on other objects outside of the Unity editor. To test these inside of the Unity editor would be a ridiculous amount of work as we would need to have a second scene that is at all times exactly the same as each development scene and contains all of the proper testing objects on top of that. This would take up way too much time and likely lead to more errors than it would solve.

Solution: Post on the Unity forums and await a response. We are currently reviewing this information before we post it so that we can be sure of it's accuracy.

Draft of the post:

"When writing system and integration tests for a project I am working on, I would like to be able to set up game objects in the testing scripts and have them call methods that affect one another. However, when this is not done in a scene in the Unity Editor, all of the components that an object might contain have to be added to it first, and then when I try and call methods on the objects they just get null reference exceptions to each other, ostensibly because the tests are taking place outside of the Unity editor.

I am using the Unity Test Tools Editor Test Runner"

5 Smoke Testing

These tests are aimed at ensuring the most basic functionality of the system. They are the first set of tests that will be run during a cloud build so that if they fail, the build fails early before it begins running more exhaustive testing.

5.1 Verify all dependencies (libraries, imports, etc)

Test 1: Check that OVR asset exists in the project

- Obtain the absolute path to the directory "Luxonic Project" by calling the Directory.GetCurrentDirectory () function.
- Use the Path.Combine() function to add the path "Assets/OVR" to the absolute path.

- Assert that the directory exists (Directory.Exists()) and that it contains files (Directory.GetDirectories(targetPath).Length > 0)) and it contains directories ((Directory.GetFiles(targetPath).Length > 0).

Test 2: Check that OVR Avatar asset exists in the project

- Obtain the absolute path to the directory “Luxonic Project” by calling the Directory.GetCurrentDirectory () function.
- Use the Path.Combine() function to add the path “Assets/OvrAvatar” to the absolute path.
- Assert that the directory exists (Directory.Exists()) and that it contains files (Directory.GetDirectories(targetPath).Length > 0)) and it contains directories ((Directory.GetFiles(targetPath).Length > 0).

Test 3: Check that OVR Avatar Settings exists in the project

- Obtain the absolute path to the directory “Luxonic Project” by calling the Directory.GetCurrentDirectory () function.
- Use the Path.Combine() function to add the path “Assets/Resources” to the absolute path.
- Assert that the directory exists (Directory.Exists()) and that it contains files (Directory.GetDirectories(targetPath).Length > 0)).

Test 4: Check that OVR Gamepad Bundle Plugin exists in the project

- Obtain the absolute path to the directory “Luxonic Project” by calling the Directory.GetCurrentDirectory () function.
- Use the Path.Combine() function to add the path “Assets/Plugins/OVRGamepad.bundle/Contents” to the absolute path.
- Assert that the directory exists (Directory.Exists()) and that it contains files (Directory.GetDirectories(targetPath).Length > 0)) and it contains directories ((Directory.GetFiles(targetPath).Length > 0).

Test 5: Check that ImageEffects standard asset exists in the project

- Obtain the absolute path to the directory “Luxonic Project” by calling the Directory.GetCurrentDirectory () function.
- Use the Path.Combine() function to add the path “Assets/Standard Assets/Editor/ImageEffects” to the absolute path.
- Assert that the directory exists (Directory.Exists()) and that it contains files (Directory.GetDirectories(targetPath).Length > 0)).

Test 6: Check that Effects standard asset exists in the project

- Obtain the absolute path to the directory “Luxonic Project” by calling the Directory.GetCurrentDirectory () function.
- Use the Path.Combine() function to add the path “Assets/Standard Assets/Effects” to the absolute path.

- Assert that the directory exists (Directory.Exists()) and that it contains files (Directory.GetDirectories(targetPath).Length > 0)) and it contains directories ((Directory.GetFiles(targetPath).Length > 0).

*NOTE: we are not checking for any testing framework dependencies as they are not critical to the system.

5.2 User gets into the default scene

Test1: Check that a default scene is present in the build

- Obtain the default scene from the list “Scenes in Build” in Unity by calling the EditorSceneManager.GetSceneByBuildIndex(0) function.
- Assert that returned scene isValid().

6 System Testing

The purpose of these tests is to ensure that the system is in compliance with the stakeholder’s requirements.The tests are all black-box and use the same options that the user would have in the user interface.

6.1 Copy Tests

6.1.1 Copy image

Testing to ensure that image copies are created properly.

- Set up by loading an image by calling Dashboard.Load(), where testImage is the test image.
- Create copy by calling Display.createCopy((Tray.getThumbnails())[0].image)
- Get the list of copies by calling GameObject.FindGameObjectsWithTag(“Copy”) and store it in the variable copiesList.

Test 1: Test that the copy exists

- Check that the copy exists in copyList by asserting that size is equal to one.
- Assert (copiesList.size() == 1)
- Assert (Sisplay.copies.size() == 1)

Test 2: Test that the copy is the same image in the Display

- Save the display image that corresponds to the copy in the variable image by calling Display.getImages()[0]
- Check that the copy GameObject is different to the corresponding image gameObject by asserting (copiesList[0] == image)

Test 3: Test that the copy is the same image in the tray

- Save the thumbnail that corresponds to the copy in the variable thumbnail by calling Tray.getThumbnails()[0]
- Check that the copy GameObject is different to the corresponding thumbnail gameObject by asserting (copiesList[0] == thumbnail)

6.1.2 Remove copy

Be sure that the copy and all related memory no longer exist but the originals remain.

- Create a copy by calling Display.CreateCopy(targetImage.getComponent<SpriteRenderer>().Sprite)
- Remove the copy by calling Copy.VRButtonClicked("Close")

Test 1: Test that the copy is no longer visible in the workspace

- Assert (copies[0].isActive() == false)

Test 2: Test that the original image is still in the display

- Get the list of images in the display by calling Display.getImages()
- Assert that the targetImage is still within the list.

Test 3: Test that the original image is still in the tray

- Get the list of thumbnails by calling Tray.getThumbnails()
- Assert that the targetImage is still in the list.

6.1.3 Change contrast

Test 1: Test that changing the contrast of the copy does not change the contrast of the display image or the tray image

- Create a copy and use the contrast function in copy to set a new contrast value
- Check that copy.renderer.material.GetFloat("_ContrastAmount") is not default value.
- Check that displayrenderer.material.HasProperty("_ContrastAmount") is false.
- Check that trayImage.renderer.material.HasProperty("_ContrastAmount") is false.

Test 2: Test that changing the contrast to the least possible value works

- Create a copy and call copy.brightness(MIN_VALUE)
- Assert that copy.renderer.material.GetFloat("_ContrastAmount") = MIN_VALUE, the lower contrast limit.

Test 3: Test that changing the contrast to the greatest possible value works

- Create a copy and call copy.brightness(MAX_VALUE)
- Assert that copy.renderer.material.GetFloat("_ContrastAmount") = MAX_VALUE, the upper contrast limit.

6.1.4 Change brightness

Test 1: Test that changing the brightness of the copy does not change the brightness of the display image or the tray image

- Create a copy and use the brightness function in copy to set a new brightness value
- Check that copy.renderer.material.GetFloat("_BrightnessAmount") is not default value.
- Check that displayrenderer.material.HasProperty("_BrightnessAmount") is false.
- Check that trayImage.renderer.material.HasProperty("_BrightnessAmount") is false.

Test 2: Test that changing the brightness to the least possible value works

- Create a copy and call copy.brightness(MIN_VALUE)
- Assert that copy.renderer.material.GetFloat("_BrightnessAmount") = MIN_VALUE, the lower contrast limit.

Test 3: Test that changing the brightness to the greatest possible value works

- Create a copy and call copy.brightness(MAX_VALUE)
- Assert that copy.renderer.material.GetFloat("_BrightnessAmount") = MAX_VALUE, the upper contrast limit.

6.1.5 Change filter

**NOTE: Change filter has not been implemented yet and cannot be tested until there is at least a stub
Test that the filter changes for the copy

Test 1: Test that changing the filter of the copy does not change the filter/color of the original
Test 2: Test that there can be multiple filters applied to a copy

6.2 FileBrowser Tests

6.2.1 Load from file browser

**NOTE: these tests will not be implemented until File Browser is implemented as at least a stub
Test that the loaded images have been added to the tray in the correct manner

- Load the image file from the file browser by calling
  LoadBar.convertAndSendImage(targetImage)
  OR Create an event to select an image file??? **Can we create this event?**

Test 1: Test that the image thumbnail has been added to the tray

- Get the list of thumbnails by calling Tray.getThumbnails()
- Assert the list size has increased by one after the image was loaded

- Assert the targetImage is in the list.

Test 2: Test that no image has been added to the tray on cancel

- Create/call an event to select cancel
- Get the list of thumbnails by calling Tray.getThumbnails()
- Assert the list size has not changed after cancel was selected.

Test 3: Test that the image has been added to the end of the tray?
Test 4: Test that the image has been added to the display
Test 5: Test that the image has not been added to the display when cancel is pressed
Test 6: Test that the image has been added to the appropriate spot in the display

6.3 Tray Tests

6.3.1 Select Image in tray

**NOTE: these tests will not be implemented until select is implemented as at least a stub
Test that the display is always displaying what is selected in the tray.

Test 1: Test that the thumbnail in the tray is selected

- Call UpdateTray() with a parameter of the thumbnail that you would like selected
- Assert that the thumbnails list is positioned at the object selected

Test 2: Test that the image is added to the display if it is not currently added to the display

- Assert that an image is not in the Tray
- Call UpdateTray() with a parameter of the image that you would like added
- Assert that the image is in the display
- Assert that the display is positioned at the image

Test 3: Test that the display snaps to the selected image if the image is already in the display

- Assert that an image is in the Tray
- Call UpdateTray() with a parameter of the image that you would like added
- Assert that there is only one of the image in the display
- Assert that the display is positioned at the image that it just tried to add

6.3.2 Remove from tray

**NOTE: these tests will not be implemented until Thumbnail Tray remove is implemented as at least a stub
Test that changing the tray properly updates the display.

- Load an image into the tray using the file browser
- Create a copy by calling Display.CreateCopy(targetImage)

Test 1: Test that the selected image is removed from the tray and from the display if the display contains it, but copies still remain

Test 2: Test that the selected image is removed from the tray and not from the display if the display does not contains it, but copies still remain

Test 3: Test that if no image is selected that nothing is removed from the tray or display and that copies still remain

Test 4: Test that remove all removes all images from the display and form the tray but not the workspace

6.4 Display Tests

6.4.1 Next/Previous image in display

Test that updating the display also updates the tray

- Load thumbnails into the tray

Test 1: Test that going to the next image in the display updates the displayImages order and the selected thumbnail in the tray.

- Save the first and second elements of the linked list displayImages located in the Display class
- Go to the next image by calling Display.ScrollLeft()
- Assert that the saved first image is now at the end of the linked list displayImages
- Assert that the saved second image is now at the beginning of the linked list displayImages

Test 2: Test that going to the previous(right) image in the display updates the displayImages order and the selected image in the tray

- Save the last and first elements of the linked list displayImages located in the Display class
- Go to the previous image by calling Display.ScrollRight()
- Assert that the saved last image is now at the beginning of the linked list displayImages
- Assert that the saved first image is now at the second position of the linked list displayImages

Test 3: Test that pressing next(left) does nothing when there are no images in the display and tray

- Set up the test by having no images within the display and tray
- Using a try catch block, try to go to the next image by calling Display.ScrollLeft() catching any possible exceptions.
- Assert exceptions == null
- Assert that the linked list displayImages has a size of zero
- Get the list of thumbnails by calling Tray.getThumbnails()

- Assert that the list of thumbnails has a size of zero

Test 4: Test that pressing prev does nothing when there are no images in the display and tray
- Set up the test by having no images within the display and tray
- Using a try catch block, try to go to the previous image by calling Display.ScrollRight() catching any possible exceptions.
- Assert exceptions == null
- Assert that the linked list displayImages has a size of zero
- Get the list of thumbnails by calling Tray.getThumbnails()
- Assert that the list of thumbnails has a size of zero

6.5 Dashboard Tests

Dashboard tests will not be implemented yet as a result of possible changes that are incoming to the Display

7 Integration testing

Involves testing the interactions between integrated components to ensure that their behaviours are proper. These tests a should cover the interactions of functions that call each other from the same class or separate classes via white-box testing and without the use of mocks. The UML diagram should help show some of the relationships between components.

7.1 Dashboard and Display

7.1.1 Minimize/maximize the buttons

Test 1 - “Minimize”, adding buttons/display back.
- Make sure all the buttons are set active
- Call Dashboard.minimize()
- Assert that all buttons are set false except for minimize
    - loadButton.gameObject.setActive == false
    - minimizeButton.gameObject.setActive == true

Test 2 - “Minimize”, adding buttons/display back.
- Set up all buttons to be inactive
- Call Dashboard.minimize()
- Assert that all the buttons are not active
    - Assert that loadButton.gameObject.setActive == true

7.1.2 Load adds image

- Set up Dashboard and display to active.

- imagesList = Display.Images.size(); // Dummy variable -- Size of List
- displayImagesList = Display.displayImages.size(); // Dummy variable -- Size of List
- Assert (imagesList == displayImagesList)
- Call Dashboard.load()
- Assert (Display.Images.size() = imagesList + 1); // Checking that image has been loaded.
- Assert (Display.displayImages.size() = displayImagesList + 1); // Checking that image has been loaded.

7.2 Dashboard and File Browser

**NOTE: file browser not yet implemented and therefore cannot be tested

7.3 Display and Tray

7.3.1 Add images

Test 1: Test that images are loaded into display and tray properly

- Set up display and tray with zero images loaded in
- imagesList = Display.Images.size();
- thumbnailsList = Tray.thumbnails.size();
- Assert (imagesList == 0)
- Assert (thumbnailList == 0)
- Call Dashboard.load()
- Assert (imagesList == 1)
- Assert (thumbnailsList == 1)

Test 2: Test with one more image

- Set up display and tray with one image loaded in
- Assert (imagesList == 1)
- Assert (thumbnailList == 1)
- Call Dashboard.load()
- Assert (imagesList == 2)
- Assert (thumbnailsList == 2)

7.3.2 Remove Image when the tray and display contain 1 image

//remove has not yet been implemented

7.3.3 Remove Image when the tray and display contain 2 or more images

//remove has not yet been implemented

7.3.4 Remove Image when more than one exists in the tray and display

//remove has not yet been implemented

7.3.5 Add Tons of images

```
for(int i=0; i < x; i++){
        Dashboard.load()
}
```

Where x = 100,1000,10000,1000000,1000000

See how many we can add and try to break everything.
Radiologists will need to be handling potentially thousands of images

7.3.6 Remove image when there is nothing to be removed

//remove has not yet been implemented for tray

7.4 Display and Copy

7.4.1 Copy Created

Test 1:create a single copy

- copyNumber = amount of copies by using tag “copies”
- Call Display.createCopy()
- Assert that (amount of copies by using tag “copies” = copyNumber + 1)
- Assert that (image created copy is the same as one in display)

Test 2:create multiple copies

- copyNumber = amount of copies by using tag “copies”
- Call Display.createCopy()
- Assert that (amount of copies by using tag “copies” = copyNumber + 1)
- Assert that (image created copy is the same as one in display)

7.4.2 Copy Deleted

- //TODO

8 Regression testing

These tests will be added to this document as bugs/errors arise. Once a bug report is made, the appropriate tests will be added here to ensure that the bug does not arise again anywhere in the code.

8.1 Buttons should not be activated more than once per press

See issue #37

- Simulate collision
- Time collision for 1 second
- Collide with load
- Ensure that only one image was brought into workspace
- Repeat the following steps with a longer timer and ensure that 1 image is loaded

8.2 Check positions of instantiated Copy objects in Display.cs to prevent overlap

- Create multiple copies
- As each copy is being created loop through current list of copies
- Ensure the location of the new copy is not already being occupied by another copy.

8.3 Double hand grasp-and-release bug

- Simulate grab with right hand
- Simulate grab with left hand which right hand is still holding on
- Ensure that the copy is still in right hand
- Repeat following steps with left hand

8.4 Unhandled case in next/prev buttons

Assert that something useful is outputted for the user

8.5 Can grab the tray images

Assert that tray images are not ‘grabbable’

9 User testing

User testing will require pre-test and post-test documentation to ensure the user understands the purpose of their testing of our software. All questions will be optional for the user to answer. These questions are formatted and explained in the “User Testing Form”. Users are ideally people without experience in computer science as we are ideally looking for user interaction glitches, not testing for bugs, though we will still get some computer science users to test for us to add robustness to our tests.

- Before The users show up the lab should be properly set up with the equipment ready to go and at least 2 team members should be present for the test.
- When the user arrives, explain to them our project using the following key points:
    - It is a Virtual reality application
    - It is meant to be used by radiologists
    - The purpose is for the viewing of DICOM images
    - The purpose of them being here testing is to help us ensure that our program flows smoothly when it is being used by somebody who is unfamiliar with it
- Take the user's Pre documentation as per the ‘User Testing Form”
- Inform the user that:
    - they must use hand sanitizer that is supplied
    - they must accept assistance in taking the headset and controllers on and off
    - We will want their first impressions when the system starts up
    - We want them to walk us through their actions and what they think is going to happen before and after they do something
    - Let them know that if at anytime they need out of the headset to say so and the tester will come help them out
- During the testing the tester is to:
    - Ask the user questions
    - Navigate the user to missed material once the user thinks that they are done
    - Take notes on their responses and comments
- Take the users post documentation
- Show the user out
- Discuss the results and take notes
- Clean the headset and controllers
- The results of the user tests will live in text files in our repository
- After each series of user testing, the notable results will be appended as a new section under user testing for the purpose of review as a potential change to our system

10 Acceptance Testing

10.1 Internal Acceptance Testing

Manual tests to be done with the Project Manager once development is near completion to ensure that the system is meeting the project’s requirements, as well as by testers and developers at regular intervals.
Acceptance tests will be performed:

- By developers after working on an affected feature.
- By testers after a merge of a feature branch into the development branch
- By testers after a development freeze
- By the Project Manager just prior to turning in a deliverable build

Record of the test being performed, and its results, will be recorded in the project wiki.

10.1.1 Quit Workspace

Test to ensure that clicking on the “Quit” button closes the workspace at anytime.

10.1.2 Show/Hide Dashboard

Test that when the “Show Dashboard” button is pressed that the dashboard is displayed with all elements which include:

- “Quit” button
- “Load” from File browser button
- The Tray with any images it had on it still loaded and highlighted
- The Display with any images it had still on it that match the highlighted images in the Tray
- “Hide Dashboard” button

Test that when “Hide Dashboard” button is pressed that all of the above elements are hidden and the only element that remains is the “Show Dashboard” button

10.1.3 Copy Image

Test that when the user presses “A” or “X” on an image in a displaying dashboard that the image is pulled out and represented as a new unity game object in front of the user separate from the dashboard. Ensure that “A” and “X” cannot be used to copy an image when the dashboard is hidden

10.1.4 Remove Image

Test that when the user presses “B” or “Y” on a copied image displaying in the workspace that the image is removed and no longer represented. Ensure that “B” and “Y” cannot be used to remove any object from the dashboard.

10.1.5 Show file browser

Test that when the “Load” button is clicked that the file browser is displayed. Ensure that the file browser exists over top of everything else in the workspace.

10.1.6 Close filebrowser

Test that when the “Close” button is pressed that the file browser completely closes and no images are loaded

10.1.7 Load from file browser

Test that when the “Load” button is click and a directory/images are highlighted that all the images that are in the directory are loaded to the tray. Ensure that “Load” does nothing when nothing is selected.

10.1.8 Select image in tray

Test that when an icon in the tray is selected that it is shown on the display along with the icons around it.

10.1.9 Remove from tray

Test that when “Remove” is clicked that the current item that is being displayed is removed from the tray and the display is updated to reflect this.

10.1.10 Remove all from tray

Test that when “Remove All” is clicked that all items are removed from the tray and the display is updated to reflect this.

10.1.11 Slide through images in display

Test that the user can grab the slider and move it back and forth to browse through images in the tray. Ensure that the tray’s selection indicator is updated to reflect this.

10.1.12 Next/previous image in display

Test that the user can press the next and previous arrow to browse through images in the tray. Ensure that the tray’s selection indicator is updated to reflect this.

10.2 External Acceptance Testing

To be done by the Stakeholder to ensure that the system is meeting the requirements.

11 Coverage Testing

In order to monitor how thorough the above mentioned testing is, we require Coverage Testing. Challenges arise when attempting to run tests outside of the engine, one of the most common of which is the error "**System.Security.SecurityException : ECall methods must be packaged into a system module.**". This creates a serious issue, where tools external to the Unity 5 engine are unavailable for our use, and where no tools internal to Unity are present to achieve a the same/similar result.
There has been a forum post made very recently about this issue, which has been responded to by a Unity Developer, who essentially stated that Unity does not see this as an important issue and that it would not be fixed/developed for anytime soon. A link to the forum post can be found here:

https://forum.unity3d.com/threads/test-runner-editormode-playmode-tests-and-code-coverage-with-nunit.447941/#post-2942085

However, after much research there appears to be a workaround to the issue. We have attempted to implement a method of manual coverage testing (meaning it would have to be run manually, rather than with each build in Unity Cloud Build), following the steps outlined in the following guides:
http://codingdebauchery.blogspot.ca/2014/03/code-coverage-for-unityc.html
https://www.snip2code.com/Snippet/17788/How-to-programmatically-convert-the-Visu

When attempting to implement the above guides, we ran into conflicts with those solutions due to the VR libraries we are using in our project. Unfortunately, we have not yet found a solution to this conflict.

12 Testing Matrix

With the shift in our testing, we do not currently have a Testing Matrix. However, even though we are not doing automated testing, we will still be making a testing matrix based on our pseudocode tests, the pair reviews that we do, our acceptance tests, and using information that we get from resolving bugs.