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 document 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.

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.

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 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.

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).

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.

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.

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.

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.

Issue #8: Permissions for 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.

Issue #9: we cannot test the methods of 2 GameObjects against each other, as that can only be done in the Unity Editor

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.

The following E-mail Was sent to Unity Customer support:

"Dear Unity,

Currently, I am creating an application with Unity for the Oculus Rift to allow the user a workspace in which they can use to perform particular tasks. This project has a fair bit of data management that I am structuring, manipulating, and storing. Normally, for writing such an application, a full set of end-to-end tests would make the project go smoother and catch errors sooner. However, while using Unity, it has been difficult to test to the calibre in which I would desire.

To the best of my knowledge and everybody around me, I cannot perform proper automated system and integration tests for my project. This stems from the fact that Unity will not allow me to use an outside editor (Visual Studio or Monodevelop) to create Unity GameObjects and interact them with one another. I have tried using the Unity Test Tools Editor Test Runner and I was able to create Unit Tests where a single script can be tested by creating the game object it needs and calling the scripts functions from that object. System and Integration testing get exponentially more complicated because I need to interact these objects with each other outside of pressing play in the Unity Editor, which does not appear to be possible.

Being able to run automated tests is critical to my project and many others, I am sure. My solution has been to create test scenes that automate parts of the tests, but I still have to manually perform them myself by clicking play and performing a few actions which takes a lot more time than if I could automate the tests completely every time that I build with Unity Cloud Build.

In short, I would like to know If Unity has any tools in place, or plans to put tools in place that allow me to create and interact Game Objects with one another outside of the Unity Editor so that we can properly automate testing.”

We will update this document with the response once it has been received