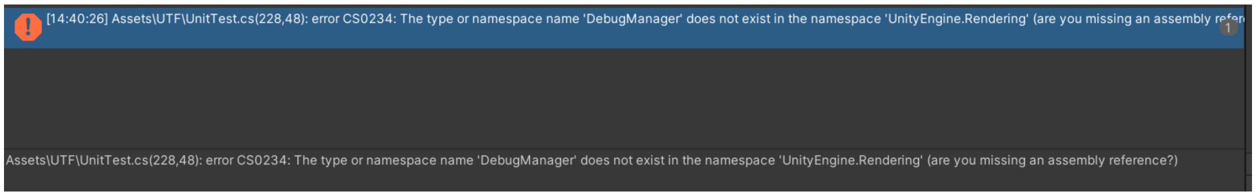
Approach: just try to use it naively like a user might. Reporting my thoughts here.

1. Run in Unity 2022.3.42f1 build-in render pipeline. Error on import:



```
[14:40:26] Assets\UTF\UnitTest.cs(228,48): error CS0234: The type or namespace name 'DebugManager' does not exist in the namespace 'UnityEngine.Rendering' (are you missing an assembly refer
Assets\UTF\UnitTest.cs(228,48): error CS0234: The type or namespace name 'DebugManager' does not exist in the namespace 'UnityEngine.Rendering' (are you missing an assembly reference?)
```

Commented it out so it can compile.

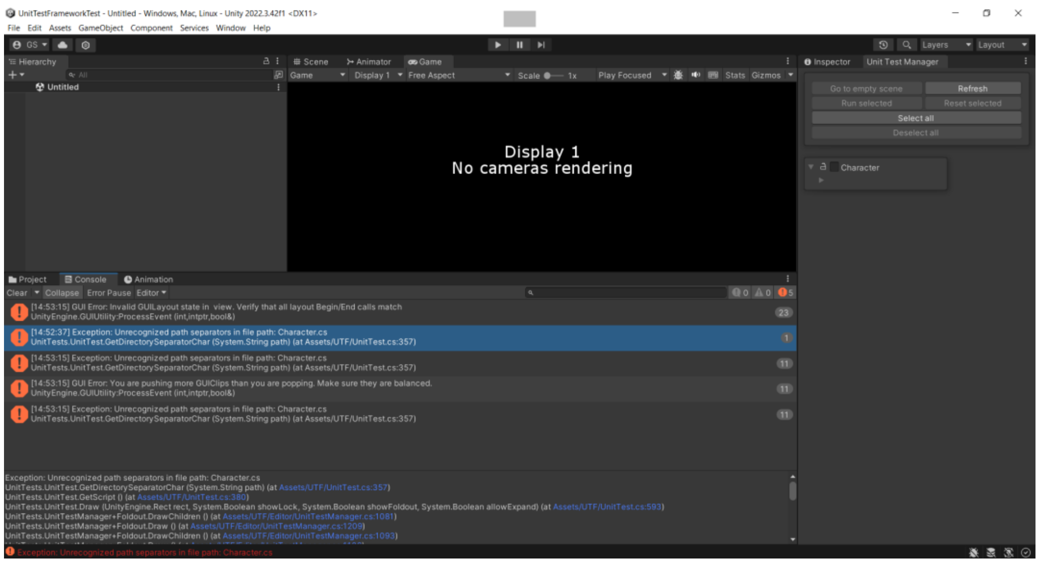
Also something about being unable to import from a similar namespace (also commented out).

2. Some sort of documentation/QuickStart guide would be nice.

3. The 'Run selected' button should be replaced with 'Run selected (enter playmode)' or some clear message that the user needs to be in play mode needs to be visible if the user needs to be in playmode to run unit tests. This is confusing/unclear otherwise.

4. Feedback when a test passed successfully or even runs in general would be preferred. I press run but am unsure if the result was a success or not (on the examples – assuming I'm doing it right).

5. Tried to write a custom test using a simple script copying off the example. It shows in the unit test manager, but when I expand to the test I get errors. See included project.



```
[14:53:15] GUI Error: Invalid GUILayout state in  view. Verify that all layout Begin/End calls match
UnityEngine.GUIUtility:ProcessEvent (int,intptr,bool&)                                                    23
[14:52:37] Exception: Unrecognized path separators in file path: Character.cs
UnitTests.UnitTest.GetDirectorySeparatorChar (System.String path) (at Assets/UTF/UnitTest.cs:357)         1
[14:53:15] Exception: Unrecognized path separators in file path: Character.cs
UnitTests.UnitTest.GetDirectorySeparatorChar (System.String path) (at Assets/UTF/UnitTest.cs:357)         11
[14:53:15] GUI Error: You are pushing more GUIClips than you are popping. Make sure they are balanced.
UnityEngine.GUIUtility:ProcessEvent (int,intptr,bool&)                                                    11
[14:53:15] Exception: Unrecognized path separators in file path: Character.cs
UnitTests.UnitTest.GetDirectorySeparatorChar (System.String path) (at Assets/UTF/UnitTest.cs:357)         11

Exception: Unrecognized path separators in file path: Character.cs
UnitTests.UnitTest.GetDirectorySeparatorChar (System.String path) (at Assets/UTF/UnitTest.cs:357)
UnitTests.UnitTest.GetScript () (at Assets/UTF/UnitTest.cs:380)
UnitTests.UnitTest.Draw (UnityEngine.Rect rect, System.Boolean showLock, System.Boolean showFoldout, System.Boolean allowExpand) (at Assets/UTF/UnitTest.cs:593)
UnitTests.UnitTestManager+Foldout.DrawChildren () (at Assets/UTF/Editor/UnitTestManager.cs:1081)
UnitTests.UnitTestManager+Foldout.Draw () (at Assets/UTF/Editor/UnitTestManager.cs:1209)
UnitTests.UnitTestManager+Foldout.DrawChildren () (at Assets/UTF/Editor/UnitTestManager.cs:1093)

Exception: Unrecognized path separators in file path: Character.cs
```

Looking into it a little bit more, perhaps I need to have a set up and tear down function like in the example?

If this is the case, I would prefer if I didn't have to write so much boilerplate code. Perhaps abstract classes or interfaces could somehow reduce this burden on me.

I tried to copy over the set up and tear down functions to see if I could get my character to work, but it didn't. Still got the same error.

Give up here. Will try again with updated version.

I didn't get the chance to evaluate it beyond the surface level, but:

If this is targeting gameplay programmers, I am mostly interested in testing for behavior.

- For example, if I spawn an enemy over here, it should be able to reach my player over there.
- Is my enemy speed the correct value all the time?
- Can I make that jump which is X units away?

Normally, I would create a test scene where I test out my various gameplay systems and develop my features. Automated testing could be useful in the event that I extend on that system and want confidence that I have not broken any previous behaviors. EX: I add air friction, but now my jump fails for the range I want! I want to be alerted of that and fix/respond.

Programmers that develop backend systems would likely feel this is closer to home. For example, this might be useful if I was developing Unity assets and wanted to verify the functionality of features.

- For example, I commonly find that I am double checking whether or not I reset values to the default values in the scene before releasing.
- I might want to verify that data that I am constructing is being correctly populated.

Overall I want to verify end-user functionality. Are my backend systems able to provide the features, data, and functionality I intend… This becomes more important for more complex projects where it is cumbersome to test everything each time.