GENERAL ROUGH OUTLINE

(ADJUST AS NEEDED)

How to create an edit mode unit test:

1. **Go to your project folders and select Assets→Tests→EditMode→*YourName***

2. **Once in your personal EditMode folder, right click in the folder, hover over "Create," then hover over "Testing" and click "C# Test Script"**

3. **Give the script a name to indicate what is being tested**

4. **In the script, delete the "UnityTest" and all lines below it unless you need it for the edit mode test (this is rare, and if you need to skip frames, then a play mode test might be a better option)**

5. **Change the name of the function under the [Test] tag to indicate what you are testing**

6. **If more than one type of test would be related to the general test name of the script, then each test can be implemented as a separate function with the [Test] tag above them.**

7. **If performing more than one test in a single script, then any setup/teardown code (creating and destroying the object to perform the tests on) that is reused in all functions should be placed in setup and teardown functions; the setup code should be placed in a function at the top labeled Setup() with the [SetUp] tag above it,**

```
[SetUp]

public void Setup()

{

   gameObject = new GameObject();

   player = gameObject.AddComponent<PlayerScript>();

}
```

and the teardown code should be placed in a function at the bottom labeled Teardown() with the [TearDown] tag above it

```
[TearDown]
```

```
public void Teardown()
{
   GameObject.DestroyImmediate(gameObject);
}
```

If doing this, then make sure to declare those variables at the top of the class outside of any functions

8. Write your unit/boundary test code, making sure to add assert family functions to test your results.

```
[Test]

public void DamageTest()

{

        // Arrange

        player.setHealth(51);



        // Act

        player.takeDamage();



        // Assert

        Assert.AreEqual(26, player.getHealth());



        // Act

        player.takeDamage();



        // Assert

        Assert.AreEqual(1, player.getHealth());

        // Act
        player.takeDamage();



        // Assert

        Assert.AreEqual(0, player.getHealth());



        // Act

        player.takeDamage();



        // Assert

        Assert.AreEqual(0, player.getHealth());

}
```

How to create a play mode unit test:

1. **Go to your project folders and select Assets→Tests→PlayMode→*YourName***

2. **Once in your personal PlayMode folder, right click in the folder, hover over "Create," then hover over "Testing" and click "C# Test Script"**

3. **Give the script a name to indicate what is being tested**

4. **Delete the "[Test]" and all lines below it until the [UnityTest] tag**

5. **Change the name of the function under the [UnityTest] tag to indicate what you are testing**

6. **If more than one type of test would be related to the general test name of the script, then each test can be implemented as a separate function with the [UnityTest] tag above them.**

7. If performing more than one test in a single script, then any setup/teardown code (creating and destroying the object to perform the tests on) that is reused in all functions should be placed in setup and teardown functions; the setup code should be placed in a function at the top labeled Setup() with the [UnitySetUp] tag above it,

```
[UnitySetUp]

public void Setup()

{

        gameObject = new GameObject();

        player = gameObject.AddComponent<PlayerScript>();

}
```

and the teardown code should be placed in a function at the bottom labeled Teardown() with the [UnityTearDown] tag above it

```
[UnityTearDown]
public void Teardown()
{
   GameObject.Destroy(gameObject);
}
```

8. Write your unit/boundary test code, making sure to add assert family functions to test your results.

```
public class MovementTest

{

        // A UnityTest behaves like a coroutine in Play Mode. In Edit Mode you can use

        // `yield return null;` to skip a frame.

        [UnityTest]

        public IEnumerator DistanceTest()
```

```
{

// Arrange

// Create object and get it's initial x position

GameObject gameObject = new GameObject();

float initXPos = gameObject.transform.position.x;

gameObject.AddComponent<PlayerScript>();



// Act

// Force player to move for 1 second

MoveHelper mover = gameObject.AddComponent<MoveHelper>();

yield return new WaitForSeconds(1);

// Stop player movement

MoveHelper.Destroy(mover);



// Assert

// Check distance traveled in 1 second

float distance = gameObject.transform.position.x - initXPos;



// Distance should be 10 units while allowing room for fluctuation due to run speed

if (distance > 10f && distance < 10.1f)

{

// If movement was within range, then round to even 10

distance = 10f;

}
```

```
        // Unit was supposed to travel very near to 10 units

        Assert.AreEqual(10f, distance);

        }

}
```

How to run your unit tests:

1. If Unity’s test runner is not already open, then go “Window” tab at the very top of Unity (if on mac, then this tab will be on your system’s taskbar)

2. After clicking on the Window tab at the top, hover over “General” and select Test Runner

3. Drag the test runner and place it wherever you like.

4. At the top of the test runner, select PlayMode or EditMode depending on which type of test you want to run

5. If you want to run everyone’s tests, click “Run All” at the top of the test runner. If you want to run all your tests, select *YourName*Edit.dll or *YourName*Play.dll and then select “Run Selected”

6. If you want to run only certain tests, then select the specific test you want to run and click “Run Selected”

7. If the test gets a checkmark, then that means that the test passed successfully

8. If the test fails, it will show a red symbol. And when selecting the specific test, it will show a message that will look like this:

It tells you what the expected result was, the result it got, as well as a path showing exactly what part of the test failed, so you know where to look.