kauno technologijos universitetas

INFORMATIKOS FAKULTETAS

Programų sistemų testavimas (T120B162)

Atliko:

IFF-8/5 gr. studentai

Paulius Balčiūnas

Kasparas Jasiūnas

Priėmė:

Eduardas Bareiša

KAUNAS 2021

Lab 2. Unit testing

Lab goals:

1. Unit tests creation in order to test software components functionality.
2. Tests generation using NUnit, Unit Test.

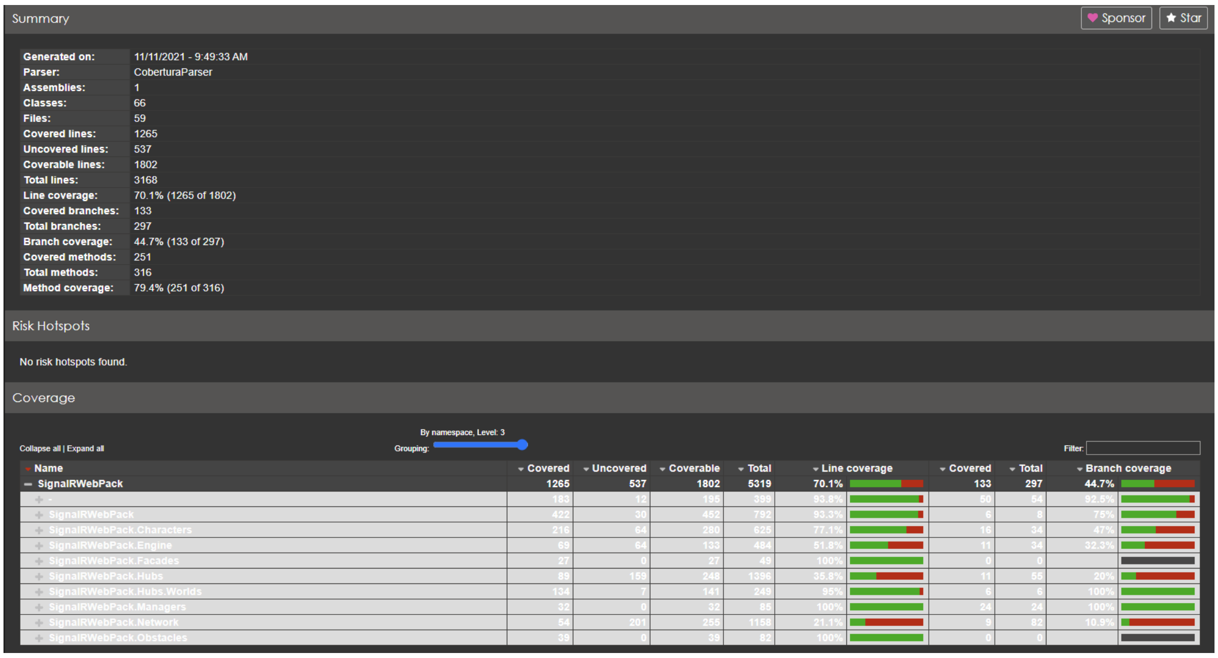
Lab work tasks:

1. Generate software unit tests in order to evaluate software quality for chosen software. The JTest, C++ Test or dot Tests programs could be used. Create some unit tests manually.

We generated test for the patterns that are used in the game. Patterns that were included: singleton, factory, abstract factory, builder, strategy, command, observer, adapter, façade.

Classes that were excluded: **network manager, program, startup, chathub**. They were all excluded because they use network connections.

2. Generate tests for a whole software application (100% code coverage)



Summary

Generated on:	11/11/2021 - 9:49:33 AM
Parser:	CoberturaParser
Assemblies:	1
Classes:	66
Files:	59
Covered lines:	1265
Uncovered lines:	537
Coverable lines:	1802
Total lines:	3168
Line coverage:	70.1% (1265 of 1802)
Covered branches:	133
Total branches:	297
Branch coverage:	44.7% (133 of 297)
Covered methods:	251
Total methods:	316
Method coverage:	79.4% (251 of 316)

Risk Hotspots

No risk hotspots found.

Coverage

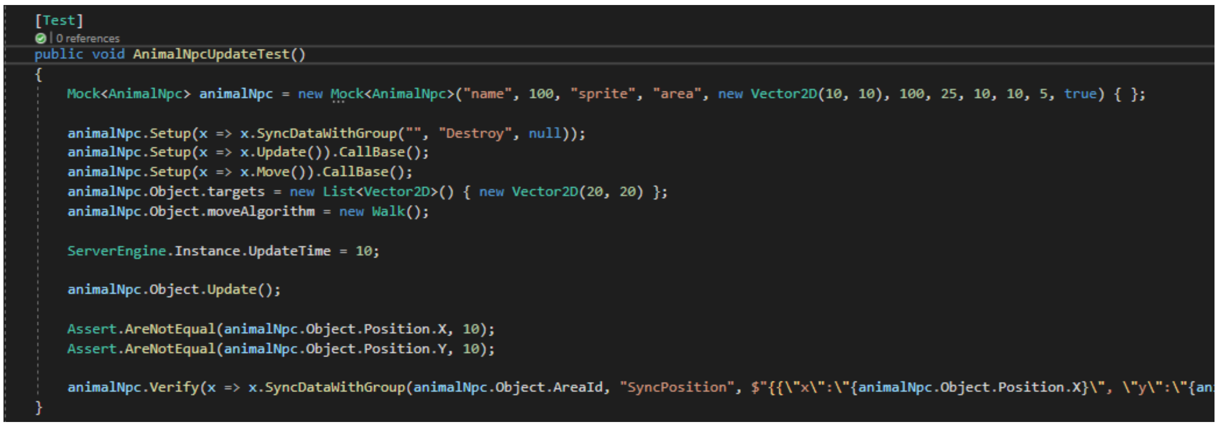
By namespace, Level: 3

Collapse all | Expand all — Grouping: — Filter:

Name	Covered	Uncovered	Coverable	Total	Line coverage		Covered	Total	Branch coverage	
SignalRWebPack	1265	537	1802	5319	70.1%		133	297	44.7%	
-	183	12	195	399	93.8%		50	54	92.5%	
SignalRWebPack	422	30	452	792	93.3%		6	8	75%	
SignalRWebPack.Characters	216	64	280	625	77.1%		16	34	47%	
SignalRWebPack.Engine	69	64	133	484	51.8%		11	34	32.3%	
SignalRWebPack.Facades	27	0	27	49	100%		0	0		
SignalRWebPack.Hubs	89	159	248	1396	35.8%		11	55	20%	
SignalRWebPack.Hubs.Worlds	134	7	141	249	95%		6	6	100%	
SignalRWebPack.Managers	32	0	32	85	100%		24	24	100%	
SignalRWebPack.Network	54	201	255	1158	21.1%		9	82	10.9%	
SignalRWebPack.Obstacles	39	0	39	82	100%		0	0		

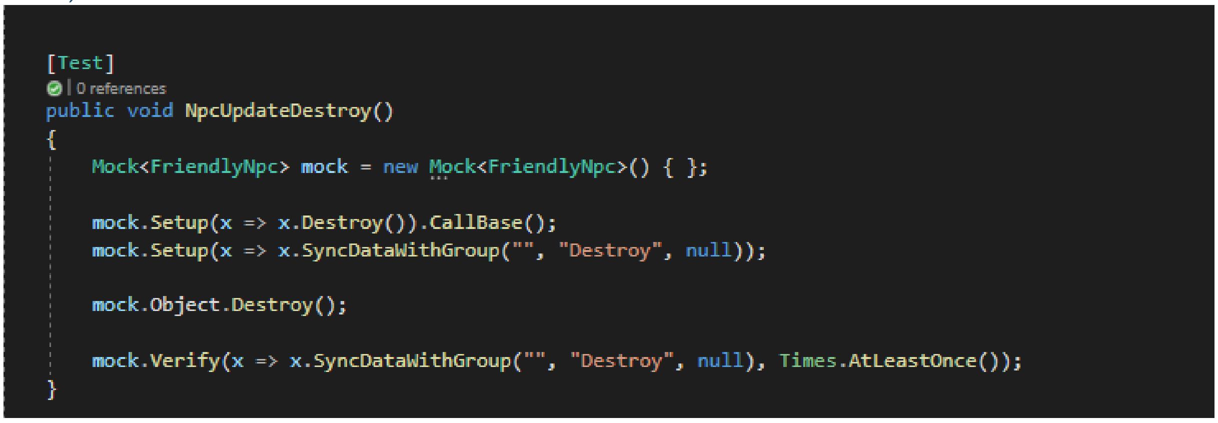
We couldn't reach the 100% coverage, because there were several scripts excluded.

3. Research mocks, stubs, drivers, use them while creating unit tests where appropriate.



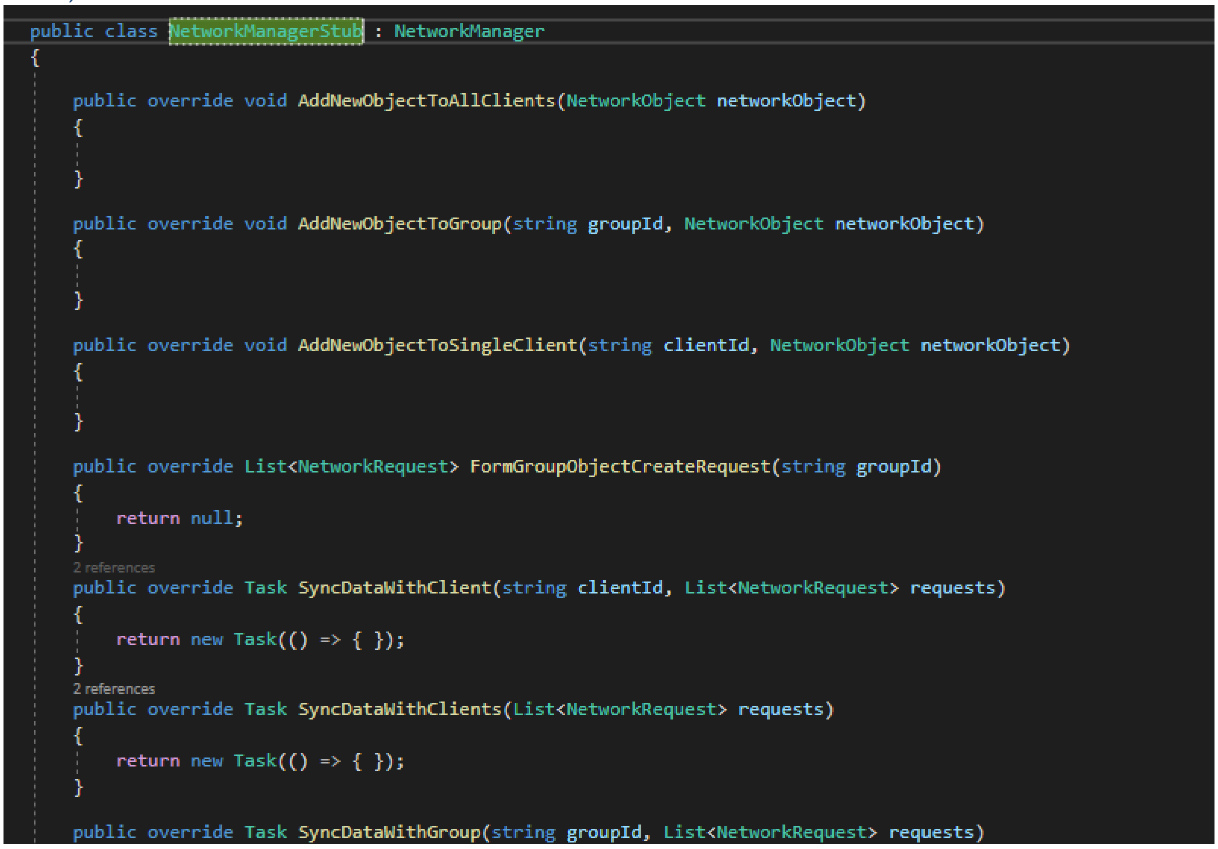
```
[Test]
0 references
public void AnimalNpcUpdateTest()
{
    Mock<AnimalNpc> animalNpc = new Mock<AnimalNpc>("name", 100, "sprite", "area", new Vector2D(10, 10), 100, 25, 10, 10, 5, true) { };

    animalNpc.Setup(x => x.SyncDataWithGroup("", "Destroy", null));
    animalNpc.Setup(x => x.Update()).CallBase();
    animalNpc.Setup(x => x.Move()).CallBase();
    animalNpc.Object.targets = new List<Vector2D>() { new Vector2D(20, 20) };
    animalNpc.Object.moveAlgorithm = new Walk();

    ServerEngine.Instance.UpdateTime = 10;

    animalNpc.Object.Update();

    Assert.AreNotEqual(animalNpc.Object.Position.X, 10);
    Assert.AreNotEqual(animalNpc.Object.Position.Y, 10);

    animalNpc.Verify(x => x.SyncDataWithGroup(animalNpc.Object.AreaId, "SyncPosition", $"{{\"x\":\"{animalNpc.Object.Position.X}\", \"y\":\"{an
}
```

Code 1, mock



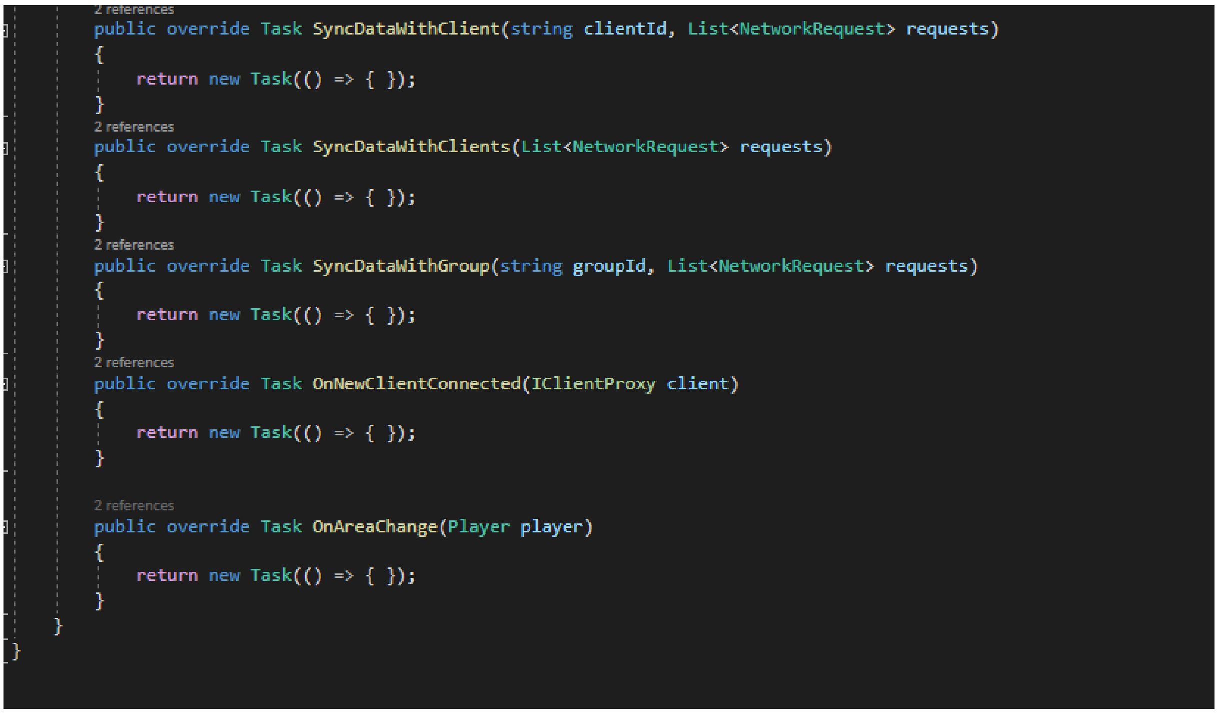
```
[Test]
0 references
public void NpcUpdateDestroy()
{
    Mock<FriendlyNpc> mock = new Mock<FriendlyNpc>() { };

    mock.Setup(x => x.Destroy()).CallBase();
    mock.Setup(x => x.SyncDataWithGroup("", "Destroy", null));

    mock.Object.Destroy();

    mock.Verify(x => x.SyncDataWithGroup("", "Destroy", null), Times.AtLeastOnce());
}
```

Code 2, mock



```
public class NetworkManagerStub : NetworkManager
{

    public override void AddNewObjectToAllClients(NetworkObject networkObject)
    {

    }

    public override void AddNewObjectToGroup(string groupId, NetworkObject networkObject)
    {

    }

    public override void AddNewObjectToSingleClient(string clientId, NetworkObject networkObject)
    {

    }

    public override List<NetworkRequest> FormGroupObjectCreateRequest(string groupId)
    {
        return null;
    }
    2 references
    public override Task SyncDataWithClient(string clientId, List<NetworkRequest> requests)
    {
        return new Task(() => { });
    }
    2 references
    public override Task SyncDataWithClients(List<NetworkRequest> requests)
    {
        return new Task(() => { });
    }

    public override Task SyncDataWithGroup(string groupId, List<NetworkRequest> requests)
```

Code 3, stub

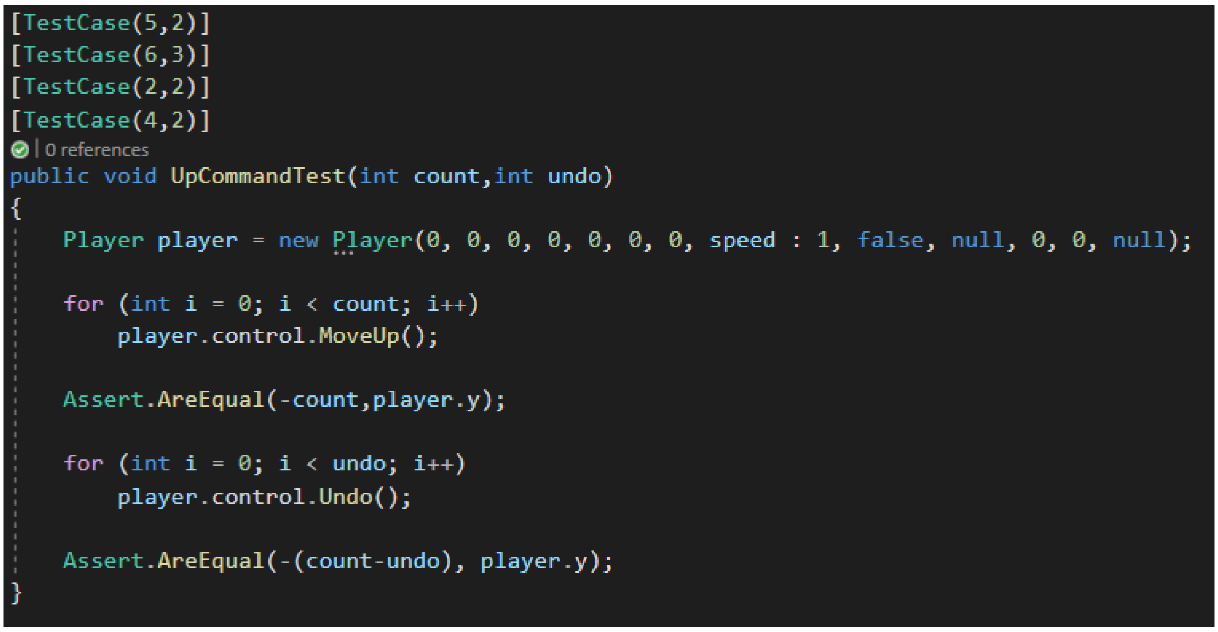


```
        2 references
        public override Task SyncDataWithClient(string clientId, List<NetworkRequest> requests)
        {
            return new Task(() => { });
        }
        2 references
        public override Task SyncDataWithClients(List<NetworkRequest> requests)
        {
            return new Task(() => { });
        }
        2 references
        public override Task SyncDataWithGroup(string groupId, List<NetworkRequest> requests)
        {
            return new Task(() => { });
        }
        2 references
        public override Task OnNewClientConnected(IClientProxy client)
        {
            return new Task(() => { });
        }

        2 references
        public override Task OnAreaChange(Player player)
        {
            return new Task(() => { });
        }
    }
}
```

Code 4, stub

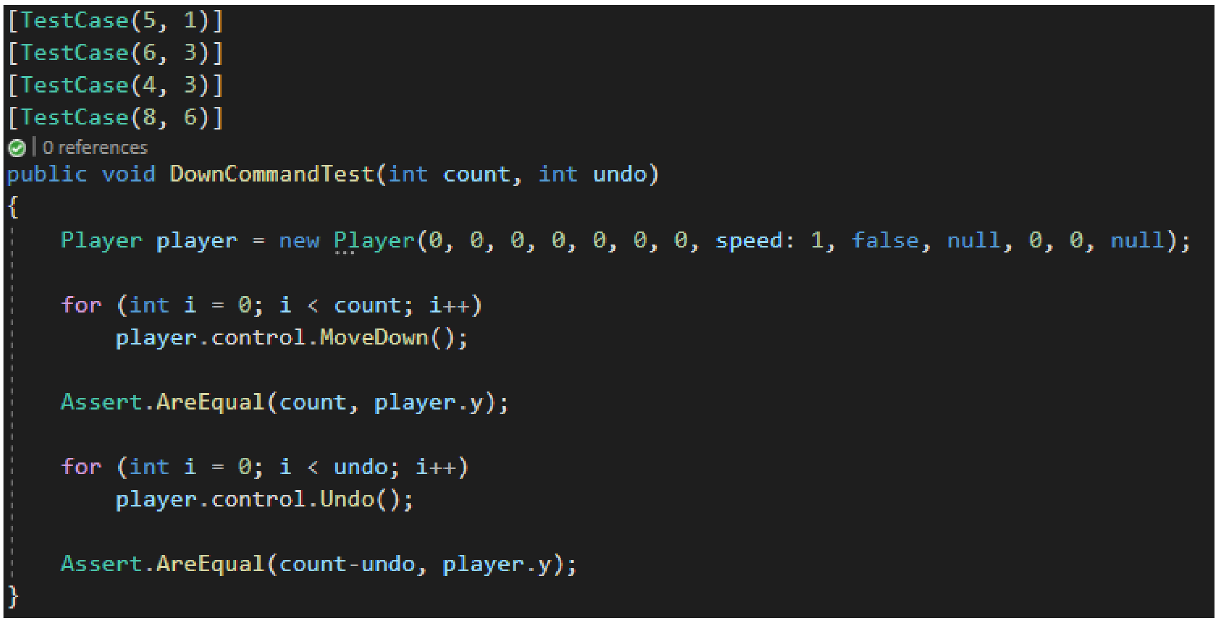
4. Research parametrized tests, use them while creating unit tests where appropriate.

One of the places where we used parametrized tests is command pattern, because it let's us know if the commands are working properly and if it's possible to undo them correctly.



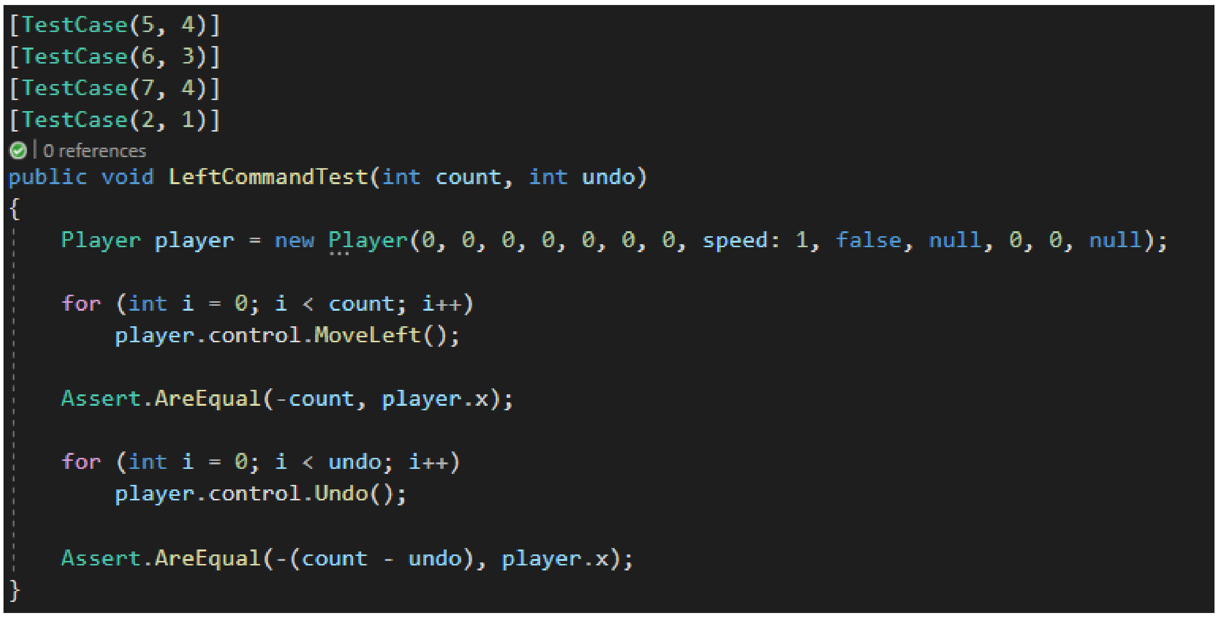
```
[TestCase(5,2)]
[TestCase(6,3)]
[TestCase(2,2)]
[TestCase(4,2)]
| 0 references
public void UpCommandTest(int count,int undo)
{
    Player player = new Player(0, 0, 0, 0, 0, 0, 0, speed : 1, false, null, 0, 0, null);

    for (int i = 0; i < count; i++)
        player.control.MoveUp();

    Assert.AreEqual(-count,player.y);

    for (int i = 0; i < undo; i++)
        player.control.Undo();

    Assert.AreEqual(-(count-undo), player.y);
}
```

Code 5, parametrized test



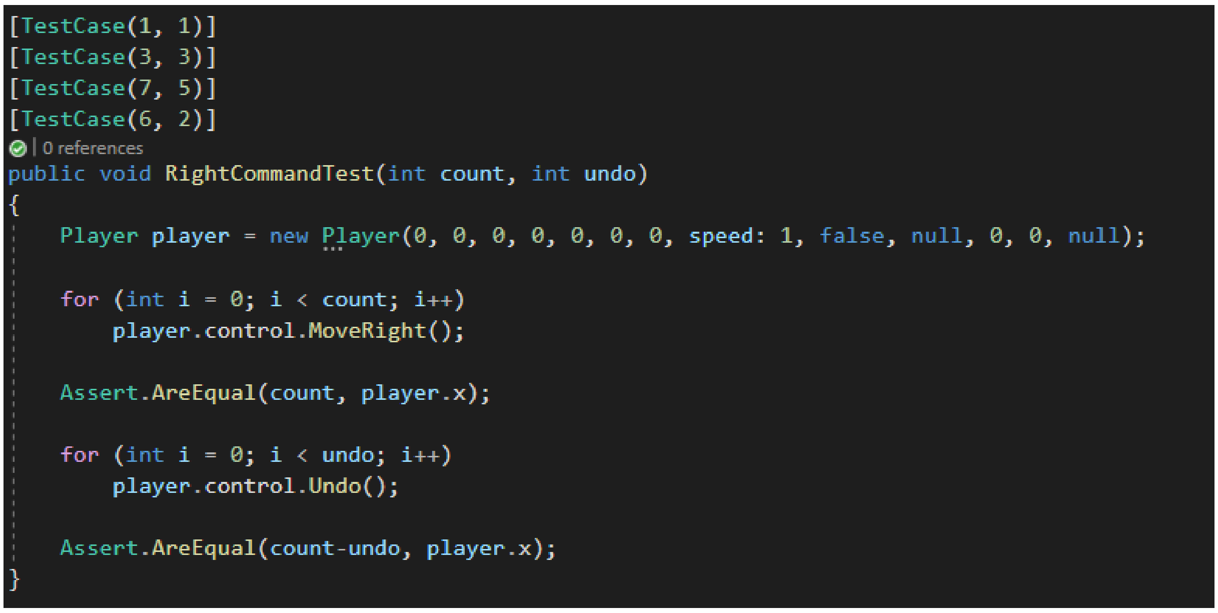
```
[TestCase(5, 1)]
[TestCase(6, 3)]
[TestCase(4, 3)]
[TestCase(8, 6)]
public void DownCommandTest(int count, int undo)
{
    Player player = new Player(0, 0, 0, 0, 0, 0, 0, speed: 1, false, null, 0, 0, null);

    for (int i = 0; i < count; i++)
        player.control.MoveDown();

    Assert.AreEqual(count, player.y);

    for (int i = 0; i < undo; i++)
        player.control.Undo();

    Assert.AreEqual(count-undo, player.y);
}
```

Code 6, parametrized test



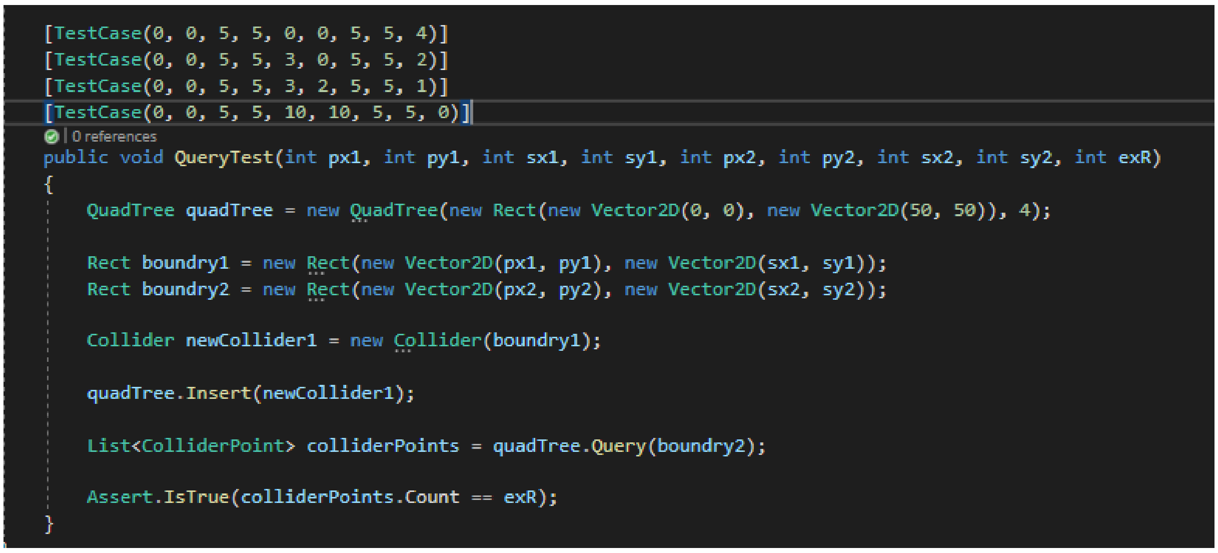
```
[TestCase(5, 4)]
[TestCase(6, 3)]
[TestCase(7, 4)]
[TestCase(2, 1)]
public void LeftCommandTest(int count, int undo)
{
    Player player = new Player(0, 0, 0, 0, 0, 0, 0, speed: 1, false, null, 0, 0, null);

    for (int i = 0; i < count; i++)
        player.control.MoveLeft();

    Assert.AreEqual(-count, player.x);

    for (int i = 0; i < undo; i++)
        player.control.Undo();

    Assert.AreEqual(-(count - undo), player.x);
}
```

Code 7, parametrized test



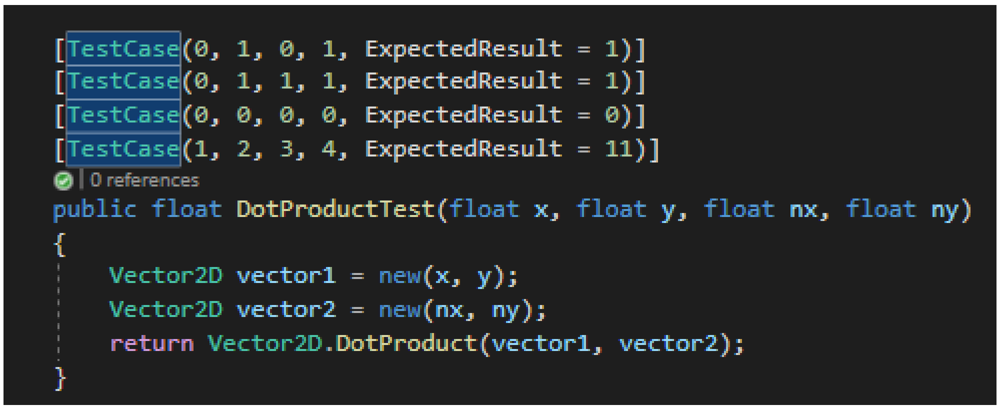
```
[TestCase(1, 1)]
[TestCase(3, 3)]
[TestCase(7, 5)]
[TestCase(6, 2)]
0 references
public void RightCommandTest(int count, int undo)
{
    Player player = new Player(0, 0, 0, 0, 0, 0, 0, speed: 1, false, null, 0, 0, null);

    for (int i = 0; i < count; i++)
        player.control.MoveRight();

    Assert.AreEqual(count, player.x);

    for (int i = 0; i < undo; i++)
        player.control.Undo();

    Assert.AreEqual(count-undo, player.x);
}
```

Code 8, parametrized test



```
[TestCase(0, 0, 5, 5, 0, 0, 5, 5, 4)]
[TestCase(0, 0, 5, 5, 3, 0, 5, 5, 2)]
[TestCase(0, 0, 5, 5, 3, 2, 5, 5, 1)]
[TestCase(0, 0, 5, 5, 10, 10, 5, 5, 0)]
0 references
public void QueryTest(int px1, int py1, int sx1, int sy1, int px2, int py2, int sx2, int sy2, int exR)
{
    QuadTree quadTree = new QuadTree(new Rect(new Vector2D(0, 0), new Vector2D(50, 50)), 4);

    Rect boundry1 = new Rect(new Vector2D(px1, py1), new Vector2D(sx1, sy1));
    Rect boundry2 = new Rect(new Vector2D(px2, py2), new Vector2D(sx2, sy2));

    Collider newCollider1 = new Collider(boundry1);

    quadTree.Insert(newCollider1);

    List<ColliderPoint> colliderPoints = quadTree.Query(boundry2);

    Assert.IsTrue(colliderPoints.Count == exR);
}
```

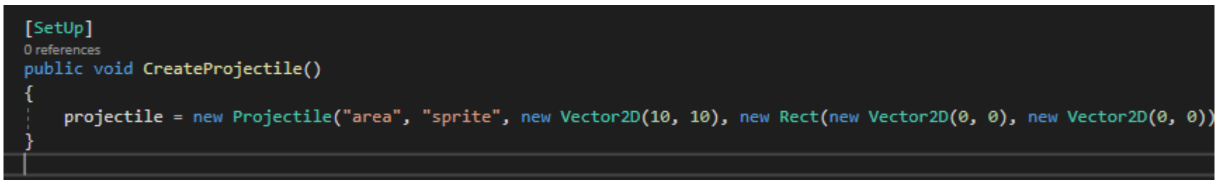
Code 9, parametrized test



```
[TestCase(0, 1, 0, 1, ExpectedResult = 1)]
[TestCase(0, 1, 1, 1, ExpectedResult = 1)]
[TestCase(0, 0, 0, 0, ExpectedResult = 0)]
[TestCase(1, 2, 3, 4, ExpectedResult = 11)]
0 references
public float DotProductTest(float x, float y, float nx, float ny)
{
    Vector2D vector1 = new(x, y);
    Vector2D vector2 = new(nx, ny);
    return Vector2D.DotProduct(vector1, vector2);
}
```

Code 10, parametrized test

5. Research tests set-up, tear-down phases, use them while creating unit tests where appropriate.

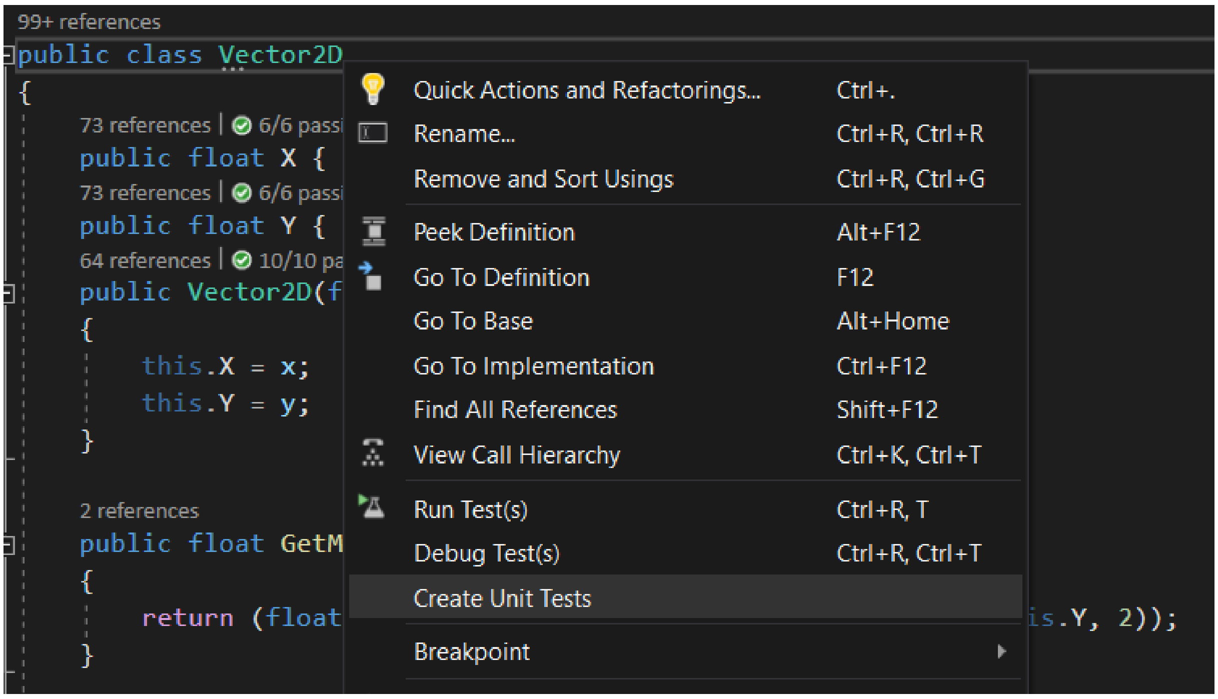


```
[SetUp]
0 references
public void CreateProjectile()
{
    projectile = new Projectile("area", "sprite", new Vector2D(10, 10), new Rect(new Vector2D(0, 0), new Vector2D(0, 0))
}
```

Code 11, set-up

Create tests using graphical editor for one chosen class to test.

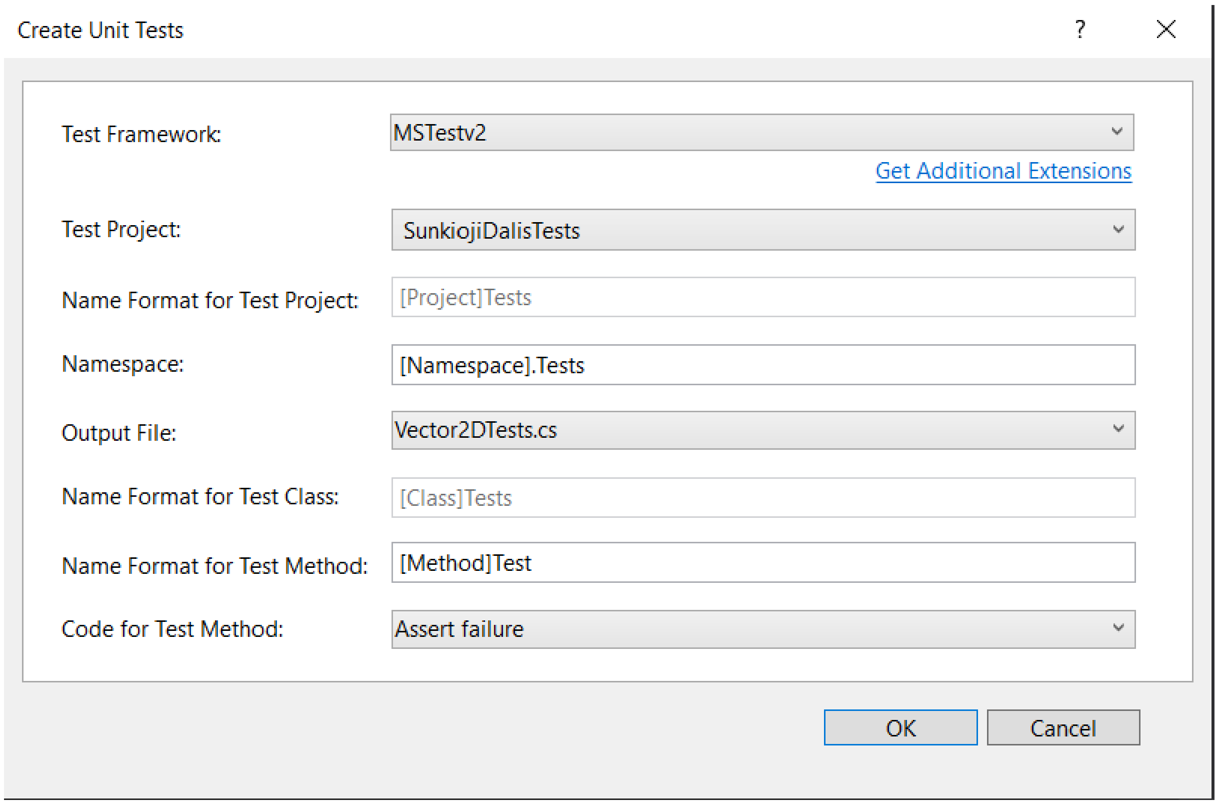
5.1. Our chosen class is **Vector2D.** To create an unit test for the whole class, you have to right-click on the class and select “Create Unit Tests”.



```
99+ references
public class Vector2D
{
    73 references | 6/6 passi
    public float X {
    73 references | 6/6 passi
    public float Y {
    64 references | 10/10 pa
    public Vector2D(f
    {
        this.X = x;
        this.Y = y;
    }

    2 references
    public float GetM
    {
        return (float                                          is.Y, 2));
    }
```

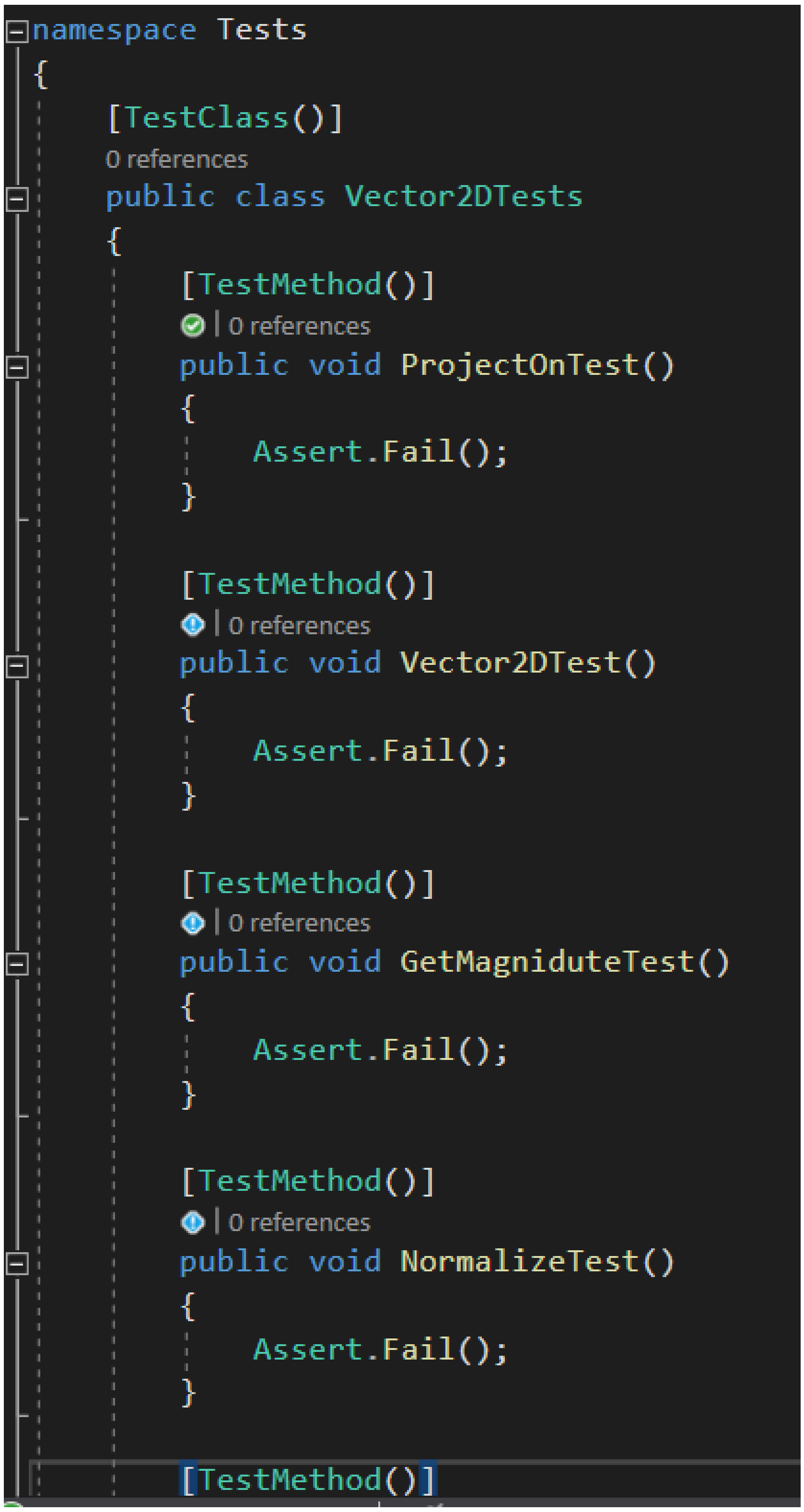
Quick Actions and Refactorings... Ctrl+.
Rename... Ctrl+R, Ctrl+R
Remove and Sort Usings Ctrl+R, Ctrl+G
Peek Definition Alt+F12
Go To Definition F12
Go To Base Alt+Home
Go To Implementation Ctrl+F12
Find All References Shift+F12
View Call Hierarchy Ctrl+K, Ctrl+T
Run Test(s) Ctrl+R, T
Debug Test(s) Ctrl+R, Ctrl+T
Create Unit Tests
Breakpoint

5.2. Select the test project and an output file name


Create Unit Tests

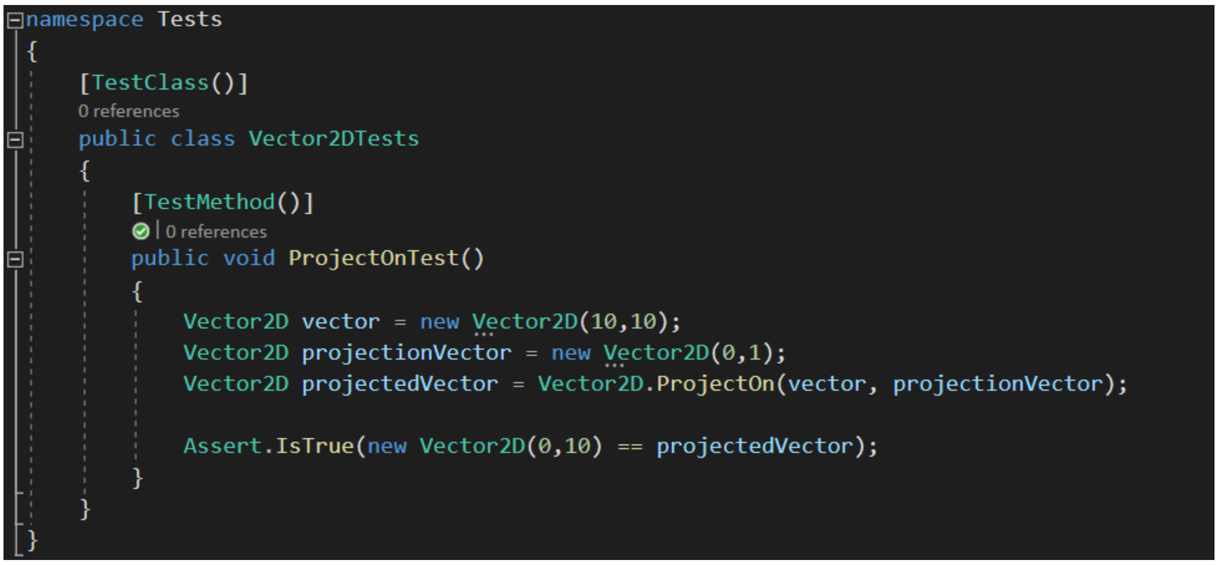
Test Framework:	MSTestv2
Test Project:	SunkiojiDalisTests
Name Format for Test Project:	[Project]Tests
Namespace:	[Namespace].Tests
Output File:	Vector2DTests.cs
Name Format for Test Class:	[Class]Tests
Name Format for Test Method:	[Method]Test
Code for Test Method:	Assert failure

5.3. Automatically generated code for all of the methods



```
namespace Tests
{
    [TestClass()]
    0 references
    public class Vector2DTests
    {
        [TestMethod()]
        | 0 references
        public void ProjectOnTest()
        {
            Assert.Fail();
        }

        [TestMethod()]
        | 0 references
        public void Vector2DTest()
        {
            Assert.Fail();
        }

        [TestMethod()]
        | 0 references
        public void GetMagniduteTest()
        {
            Assert.Fail();
        }

        [TestMethod()]
        | 0 references
        public void NormalizeTest()
        {
            Assert.Fail();
        }

        [TestMethod()]
```

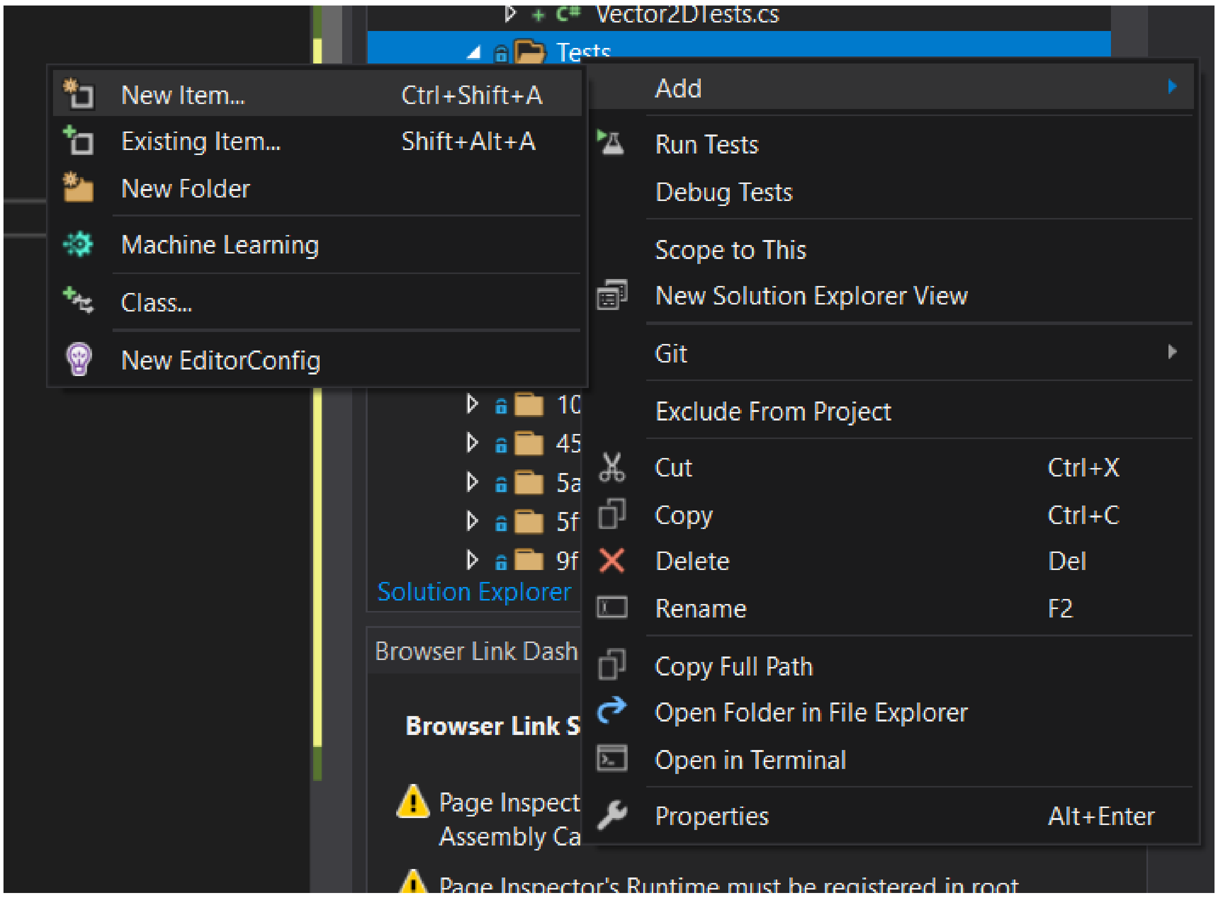
5.4. The methods that were generated are empty, that's why we have to fill in them ourselves. This is an example unit code for the **Vector2D.ProjectOn()** method.

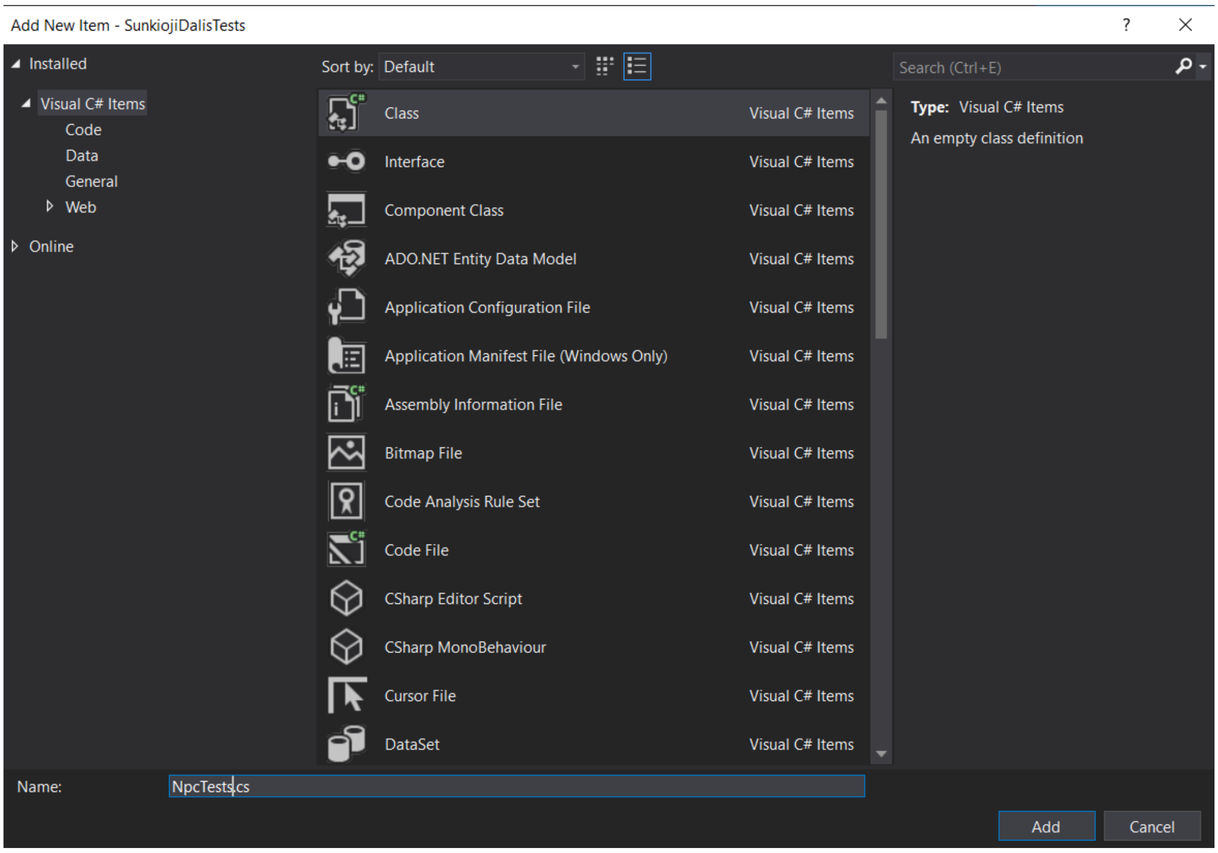


```
namespace Tests
{
    [TestClass()]
    0 references
    public class Vector2DTests
    {
        [TestMethod()]
        | 0 references
        public void ProjectOnTest()
        {
            Vector2D vector = new Vector2D(10,10);
            Vector2D projectionVector = new Vector2D(0,1);
            Vector2D projectedVector = Vector2D.ProjectOn(vector, projectionVector);

            Assert.IsTrue(new Vector2D(0,10) == projectedVector);
        }
    }
}
```

6. Create unit tests in code for one chosen class to test.

6.1. First we need to create a new class for testing



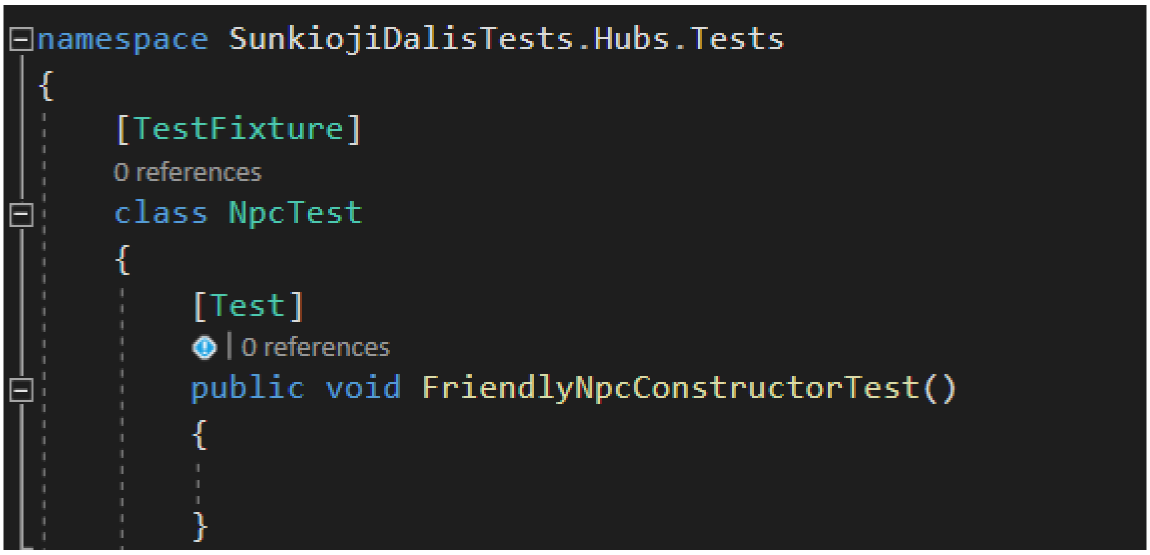

Add New Item - SunkiojiDalisTests

Class	Visual C# Items
Interface	Visual C# Items
Component Class	Visual C# Items
ADO.NET Entity Data Model	Visual C# Items
Application Configuration File	Visual C# Items
Application Manifest File (Windows Only)	Visual C# Items
Assembly Information File	Visual C# Items
Bitmap File	Visual C# Items
Code Analysis Rule Set	Visual C# Items
Code File	Visual C# Items
CSharp Editor Script	Visual C# Items
CSharp MonoBehaviour	Visual C# Items
Cursor File	Visual C# Items
DataSet	Visual C# Items

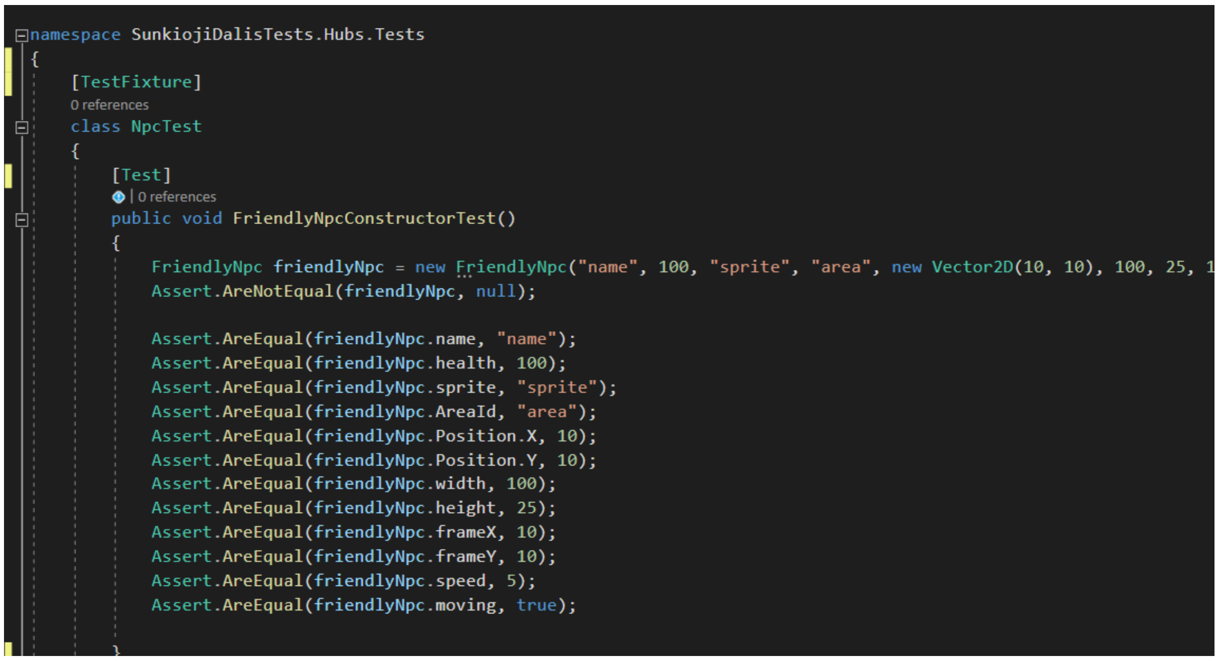
Type: Visual C# Items
An empty class definition

Name: NpcTests.cs

6.2. Then we add the necessary attributes to the class and methods.


```
namespace SunkiojiDalisTests.Hubs.Tests
{
    [TestFixture]
    0 references
    class NpcTest
    {
        [Test]
        | 0 references
        public void FriendlyNpcConstructorTest()
        {

        }
```

6.3. Last but not least, we add the testing code for the class methods



```
namespace SunkiojiDalisTests.Hubs.Tests
{
    [TestFixture]
    0 references
    class NpcTest
    {
        [Test]
        | 0 references
        public void FriendlyNpcConstructorTest()
        {
            FriendlyNpc friendlyNpc = new FriendlyNpc("name", 100, "sprite", "area", new Vector2D(10, 10), 100, 25, 1
            Assert.AreNotEqual(friendlyNpc, null);

            Assert.AreEqual(friendlyNpc.name, "name");
            Assert.AreEqual(friendlyNpc.health, 100);
            Assert.AreEqual(friendlyNpc.sprite, "sprite");
            Assert.AreEqual(friendlyNpc.AreaId, "area");
            Assert.AreEqual(friendlyNpc.Position.X, 10);
            Assert.AreEqual(friendlyNpc.Position.Y, 10);
            Assert.AreEqual(friendlyNpc.width, 100);
            Assert.AreEqual(friendlyNpc.height, 25);
            Assert.AreEqual(friendlyNpc.frameX, 10);
            Assert.AreEqual(friendlyNpc.frameY, 10);
            Assert.AreEqual(friendlyNpc.speed, 5);
            Assert.AreEqual(friendlyNpc.moving, true);

        }
```

7. Evaluate software tests coverage.

Method coverage: 79.4% (251 of 316)
Line coverage: 70.1 % (1265 of 1802)

Our goal was to have as much method testes as possible. Difficulties came from testing network, asynchronous methods and because of that, we lost a considerable amount of coverage.