# Carrying out During Development Testing in Visual Studio

It makes sense to utilise the facilities for unit testing within Visual Studio to create test code and run the tests.

## Creating the Test project within your solution for your game

The steps are:

1) Create a new project “File, Add, New Project”   
2) Choose “Installed, Other Languages, Visual C#, Test” then select “Unit Test Project”. (If Unit Test Project doesn’t show up you’ll need to change to .NET Framework 4.5.x on the dropdown).  
3) Give the project a Name – I’ve called mine NewBreakOutTests  
4) Click OK.

This creates the Test project within your solution for your game.

Now before any tests can be added we need to add a reference for any classes we wish to write test code for.  
  
5) Right click on NewBreakOutTests in the solution explorer and select “Add Reference”.

6) Select “Projects, Solution” and tick BreakoutGameDemo so you can write test code for it.

7) Also while on this screen add a reference for the Assembly System.Drawing (needed to access Brush colours).

8) Optionally write “using BreakoutGameDemo;” at the top of the code for the test class file (UnitTest1.cs). This saves having to refer to your classes using fully qualified names e.g. BreakoutGameDemo.GameObject and just refer directly to them e.g. GameObject. You can also add “using System.Drawing;”.

## The AAA Pattern to writing unit tests

* The AAA (Arrange, Act, Assert) pattern is a common way of writing unit tests for a method under test.
* The Arrange section of a unit test method initializes objects and sets the value of the data that is passed to the method under test.
* The Act section invokes the method under test with the arranged parameters.
* The Assert section verifies that the action of the method under test behaves as expected.

## Kinds of Asserts

### Assert

In your test method, you can call any number of methods of the Assert class, such as Assert.AreEqual(). The Assert class has many methods to choose from, and many of those methods have several overloads.

### StringAssert

Use the StringAssert class to compare strings. This class contains a variety of useful methods such as StringAssert.Contains, StringAssert.Matches, and StringAssert.StartsWith.

***There are more kinds but this is out of the scope of this tutorial and is not required for A Level.***

## Here is an example of Unit Test Code for the BreakOutGameDemo game’s GameObject Class

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using BreakoutGameDemo;

using System.Drawing;

namespace NewBreakOutTests

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestGetBlnIsVisible()

{

GameObject brick = new GameObject(41, 100, 80, 20, true, Brushes.Red);

Assert.AreEqual(true, brick.GetBlnIsVisible());

}

[TestMethod]

public void TestPosition()

{

GameObject brick = new GameObject(41, 100, 80, 20, true, Brushes.Red);

Rectangle rec = brick.GetRecPosition();

Rectangle testRec = new Rectangle(41, 100, 80, 20);

Assert.AreEqual(rec, testRec);

}

[TestMethod]

public void TestBrushColour()

{

GameObject brick = new GameObject(41, 100, 80, 20, true, Brushes.Red);

Assert.AreEqual(System.Drawing.Brushes.Red, brick.GetDrawingBrushColour());

brick.SetDrawingBrushColour(System.Drawing.Brushes.Yellow);

Assert.AreEqual(System.Drawing.Brushes.Yellow, brick.GetDrawingBrushColour());

}

[TestMethod]

public void TestVisibility()

{

GameObject brick = new GameObject(41, 100, 80, 20, true, Brushes.Red);

brick.MakeInvisible();

Assert.AreEqual(false, brick.GetBlnIsVisible());

brick.MakeVisible();

Assert.AreEqual(true, brick.GetBlnIsVisible());

}

[TestMethod]

public void TestCheckForIntersect()

{

GameObject brick = new GameObject(41, 100, 80, 20, true, Brushes.Red);

GameObject brickOther = new GameObject(41, 100, 80, 20, true, Brushes.Red);

Assert.AreEqual(true, brick.CheckForIntersect(brickOther.GetRecPosition()));

}

[TestMethod]

public void TestFailedCheckForIntersect()

{

GameObject brick = new GameObject(41, 100, 80, 20, true, Brushes.Red);

GameObject brickOther = new GameObject(0, 0, 80, 20, true, Brushes.Red);

Assert.AreEqual(false, brick.CheckForIntersect(brickOther.GetRecPosition()));

}

[TestMethod]

public void TestBorderlineCheckForIntersect()

{

GameObject brick = new GameObject(41, 100, 80, 20, true, Brushes.Red);

GameObject brickOther = new GameObject(120, 119, 80, 20, true, Brushes.Red);

Assert.AreEqual(true, brick.CheckForIntersect(brickOther.GetRecPosition()));

}

}

## }

## Running tests

In order to run the tests you have coded the project you must first select Build, Build Solution (Ctrl+Shift+B). Once this is complete you can select Test, Run, All Tests. The Test Explorer window should appear (if not already showing) with the results of your tests.

## Playlists

It helps to organise your tests into playlists so when it comes to screenshotting evidence of during development testing results you can easily just show the results for the class you are testing and not the whole project.

After running the tests as shown above you can simply right click on a test to add it to a playlist (either existing or new playlist). If you can’t see your test make sure you are not already displaying a playlist (go to PlayList and choose All Tests on the dropdown list).

