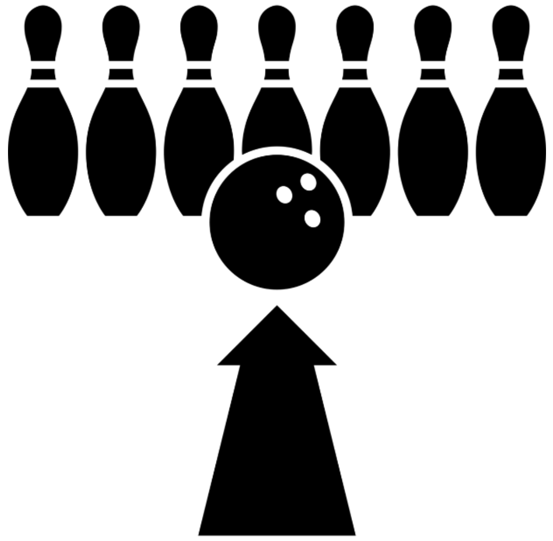
Bowling Game Kata

.NET



Scoring Bowling

10
Frames

- Frame = two opportunities (roll) to knock down 10 pins
- Frame score = roll1 + roll2 + bonuses

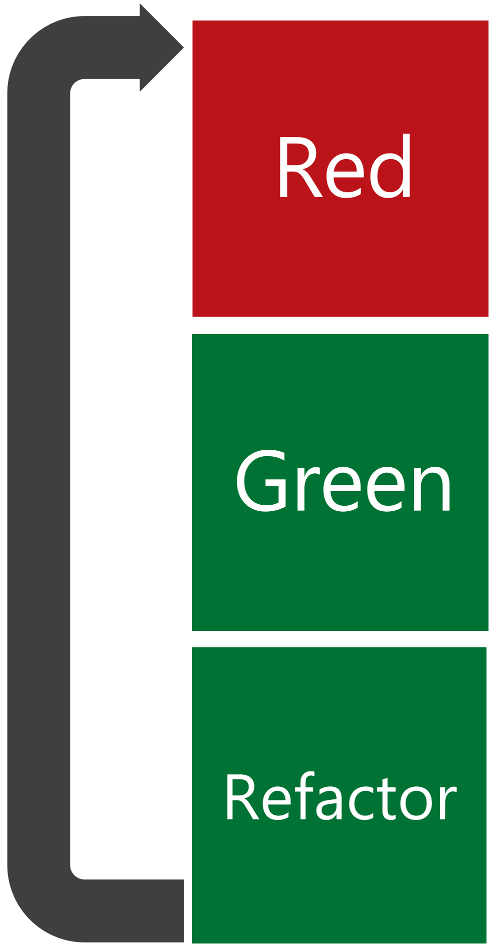
Strikes and spares gives you bonuses

- Spare score: 10 + score of next roll
- Strike score: 10 + score of next TWO rolls

Requirements

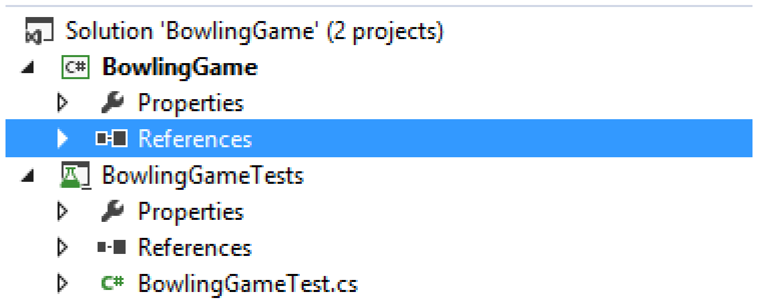
- Game class
  - *void roll(int pins)*
    - Called each time the player rolls a ball
  - *int score()*
    - Called at the end of the game
    - Returns the total score for that game

TDD cycle



Create the VS solution

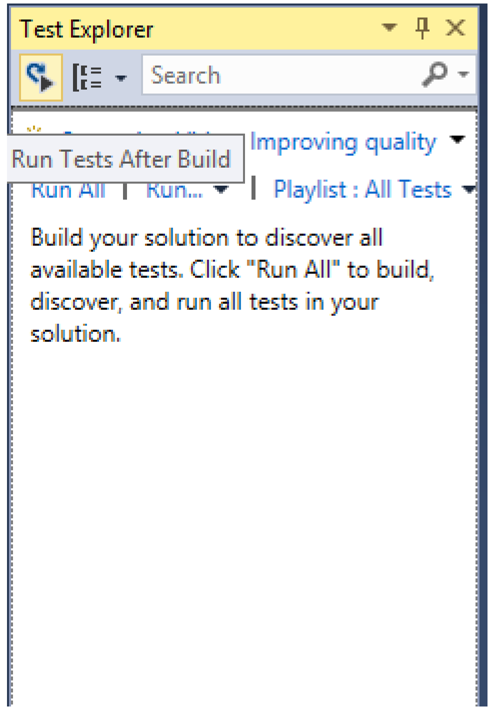
- Create a class project named BowlingGame
  - Delete Class1
- Create a test project named BowlingGameTest
  - Rename OOB Test class to *BowlingGameTest*



```
Solution 'BowlingGame' (2 projects)
◢ BowlingGame
    ▷ Properties
    ▷ References
◢ BowlingGameTests
    ▷ Properties
    ▷ References
    ▷ BowlingGameTest.cs
```

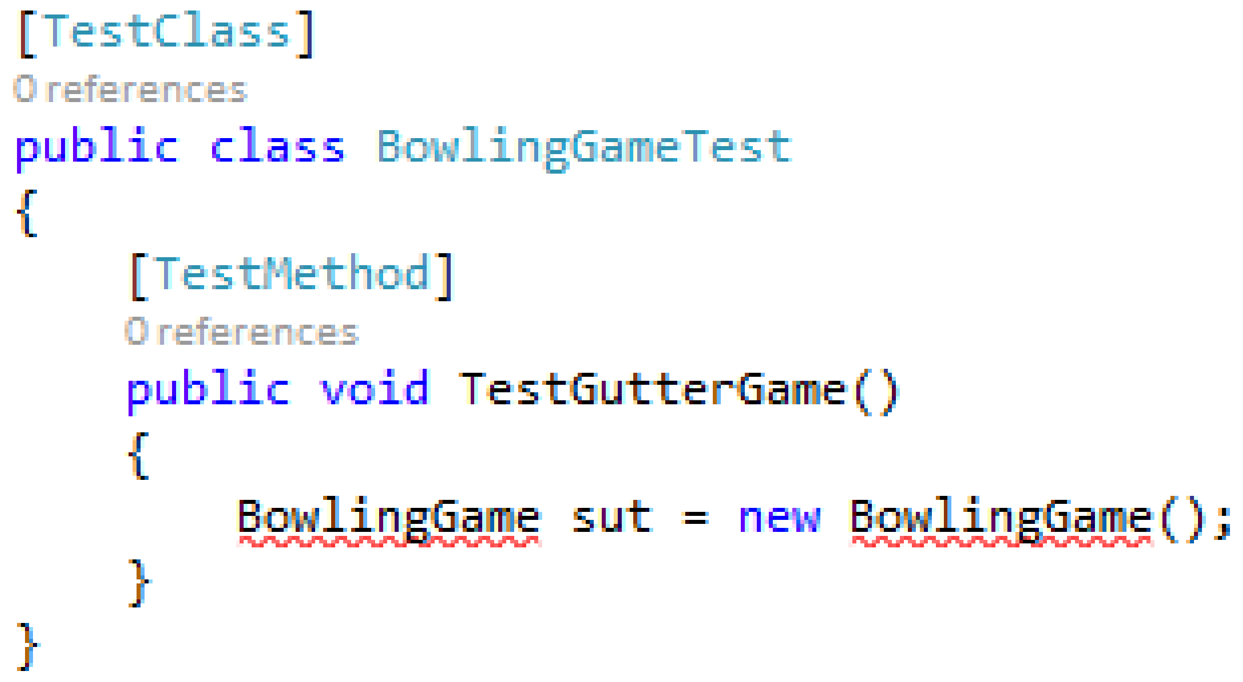
Begin

- Open Test Explorer
- Activate the “Run tests after build” feature



Build your solution to discover all available tests. Click "Run All" to build, discover, and run all tests in your solution.

The first test

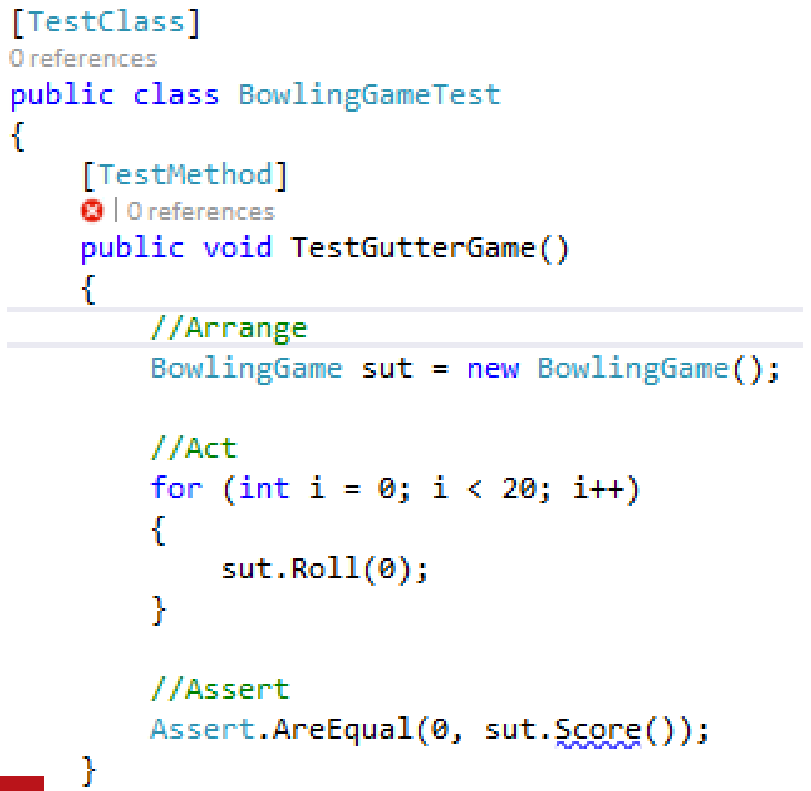


```
[TestClass]
0 references
public class BowlingGameTest
{
    [TestMethod]
    0 references
    public void TestGutterGame()
    {
        BowlingGame sut = new BowlingGame();
    }
}
```

Red

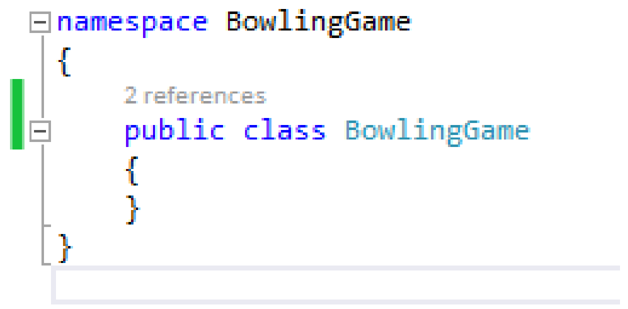
The first Test

Test



```
[TestClass]
0 references
public class BowlingGameTest
{
    [TestMethod]
    | 0 references
    public void TestGutterGame()
    {
        //Arrange
        BowlingGame sut = new BowlingGame();

        //Act
        for (int i = 0; i < 20; i++)
        {
            sut.Roll(0);
        }

        //Assert
        Assert.AreEqual(0, sut.Score());
    }
```

Code

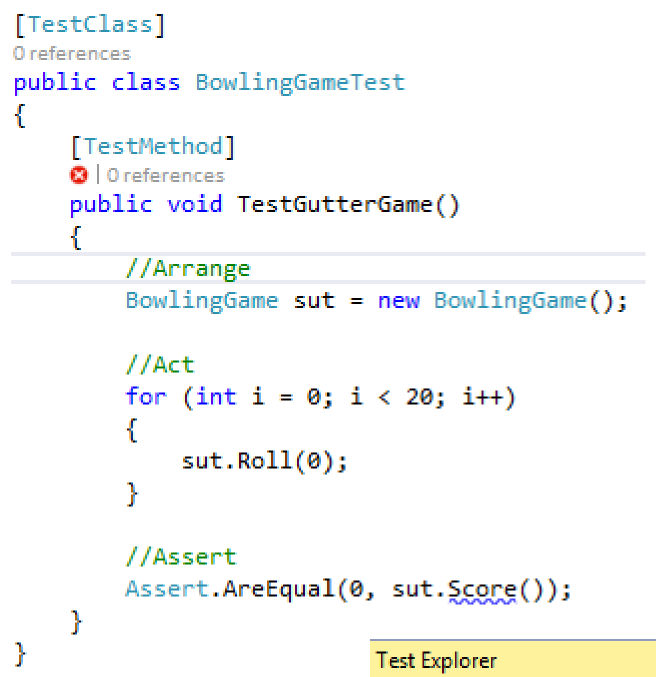


```
namespace BowlingGame
{
    2 references
    public class BowlingGame
    {
    }
}
```

Red

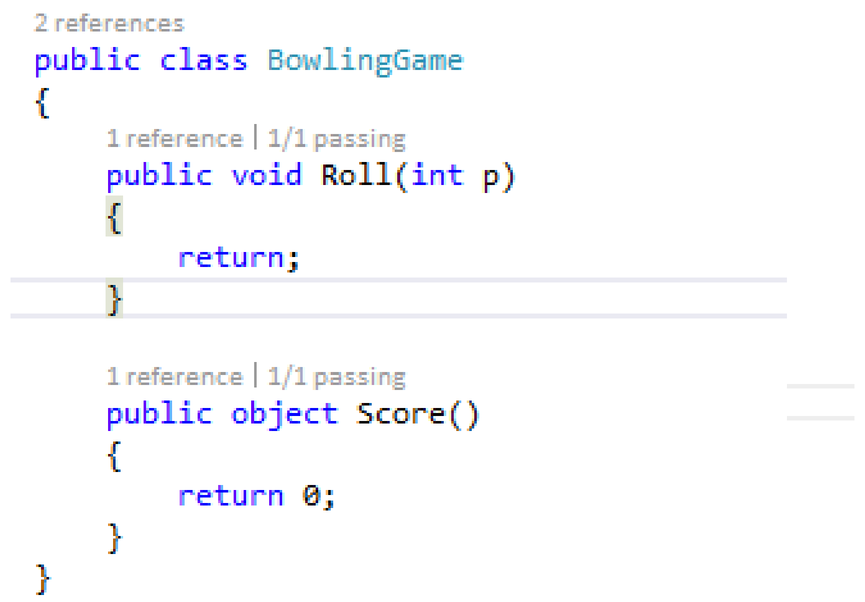
The first Test: gutter game

Test

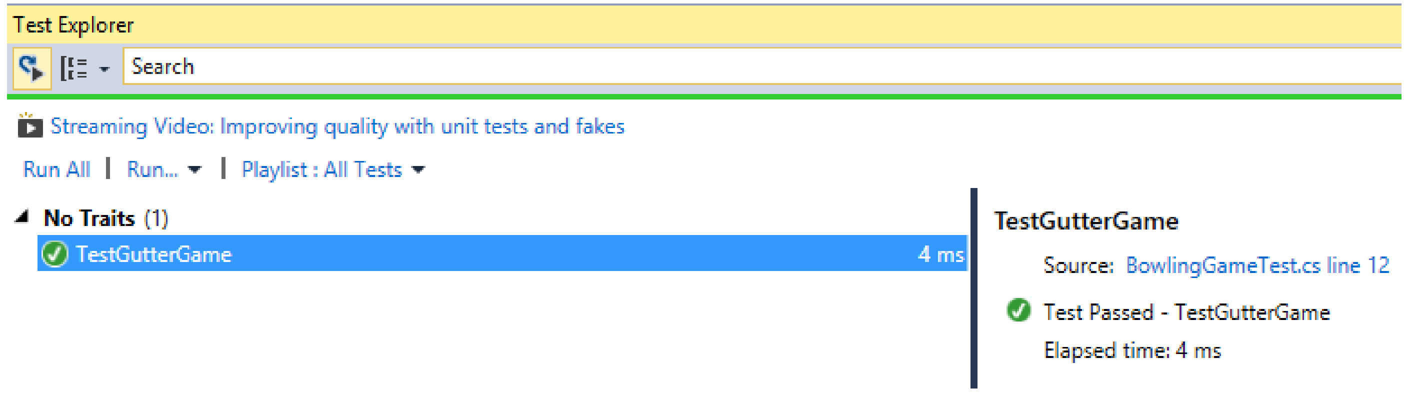


```
[TestClass]
0 references
public class BowlingGameTest
{
    [TestMethod]
    | 0 references
    public void TestGutterGame()
    {
        //Arrange
        BowlingGame sut = new BowlingGame();

        //Act
        for (int i = 0; i < 20; i++)
        {
            sut.Roll(0);
        }

        //Assert
        Assert.AreEqual(0, sut.Score());
    }
}
```

Code



```
2 references
public class BowlingGame
{
    1 reference | 1/1 passing
    public void Roll(int p)
    {
        return;
    }

    1 reference | 1/1 passing
    public object Score()
    {
        return 0;
    }
}
```



Test Explorer

Search

Streaming Video: Improving quality with unit tests and fakes

Run All | Run... | Playlist : All Tests

No Traits (1)

TestGutterGame 4 ms

TestGutterGame

Source: BowlingGameTest.cs line 12

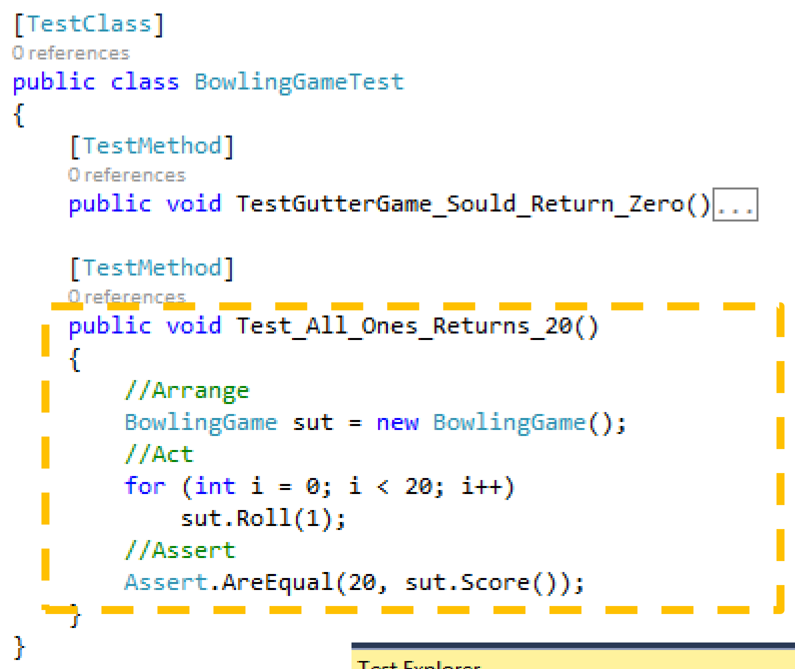
Test Passed - TestGutterGame

Elapsed time: 4 ms

Green

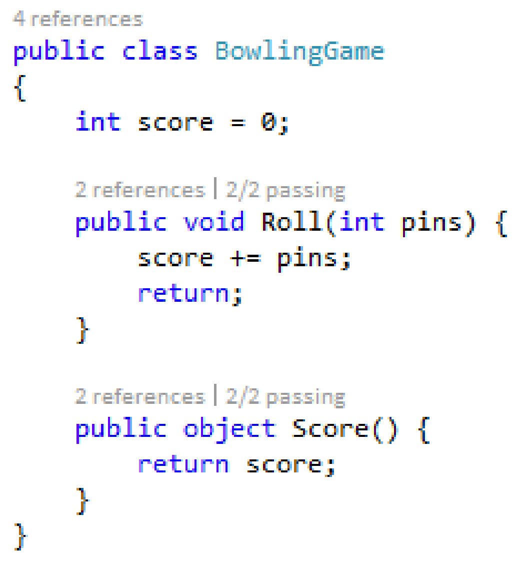
The second test: Game with no bonus

Test

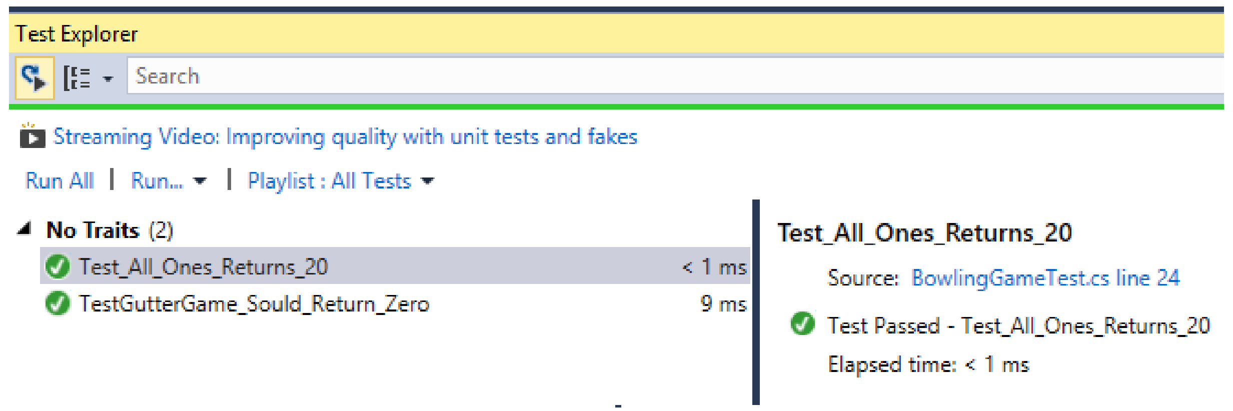


```
[TestClass]
0 references
public class BowlingGameTest
{
    [TestMethod]
    0 references
    public void TestGutterGame_Sould_Return_Zero()...

    [TestMethod]
    0 references
    public void Test_All_Ones_Returns_20()
    {
        //Arrange
        BowlingGame sut = new BowlingGame();
        //Act
        for (int i = 0; i < 20; i++)
            sut.Roll(1);
        //Assert
        Assert.AreEqual(20, sut.Score());
    }
}
```

Code



```
4 references
public class BowlingGame
{
    int score = 0;

    2 references | 2/2 passing
    public void Roll(int pins) {
        score += pins;
        return;
    }

    2 references | 2/2 passing
    public object Score() {
        return score;
    }
}
```



Test Explorer

Search

Streaming Video: Improving quality with unit tests and fakes

Run All | Run... ▾ | Playlist : All Tests ▾

No Traits (2)

✓ Test_All_Ones_Returns_20	< 1 ms
✓ TestGutterGame_Sould_Return_Zero	9 ms

Test_All_Ones_Returns_20

Source: BowlingGameTest.cs line 24

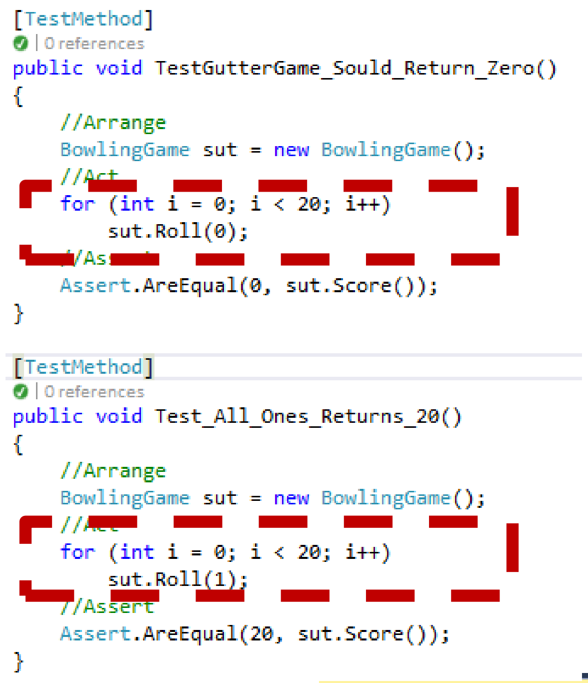
✓ Test Passed - Test_All_Ones_Returns_20

Elapsed time: < 1 ms

Green

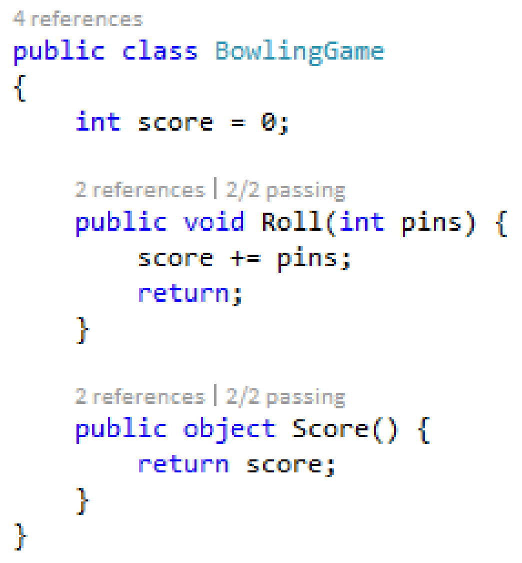
Time to refactor (DRY violation)

Test

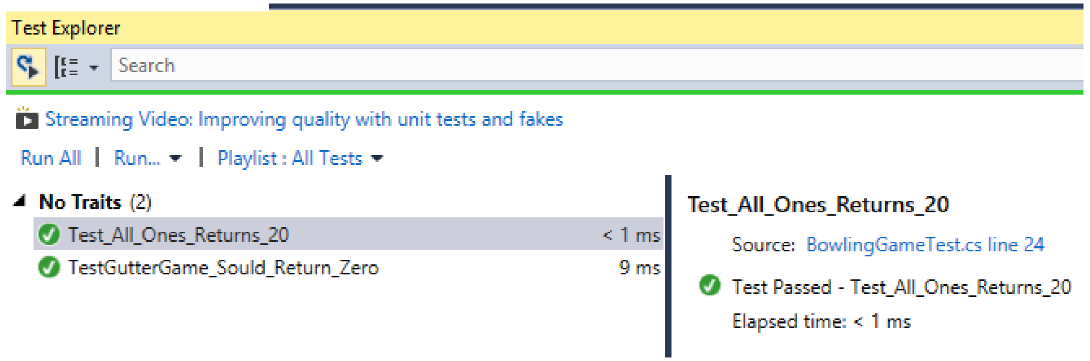


```
[TestMethod]
0 references
public void TestGutterGame_Sould_Return_Zero()
{
    //Arrange
    BowlingGame sut = new BowlingGame();
    //Act
    for (int i = 0; i < 20; i++)
        sut.Roll(0);
    //Assert
    Assert.AreEqual(0, sut.Score());
}

[TestMethod]
0 references
public void Test_All_Ones_Returns_20()
{
    //Arrange
    BowlingGame sut = new BowlingGame();
    //Act
    for (int i = 0; i < 20; i++)
        sut.Roll(1);
    //Assert
    Assert.AreEqual(20, sut.Score());
}
```

Code



```
4 references
public class BowlingGame
{
    int score = 0;

    2 references | 2/2 passing
    public void Roll(int pins) {
        score += pins;
        return;
    }

    2 references | 2/2 passing
    public object Score() {
        return score;
    }
}
```



```
Test Explorer
Search
Streaming Video: Improving quality with unit tests and fakes
Run All | Run... | Playlist : All Tests
No Traits (2)
  Test_All_Ones_Returns_20          < 1 ms
  TestGutterGame_Sould_Return_Zero    9 ms

Test_All_Ones_Returns_20
  Source: BowlingGameTest.cs line 24
  Test Passed - Test_All_Ones_Returns_20
  Elapsed time: < 1 ms
```

Refactor

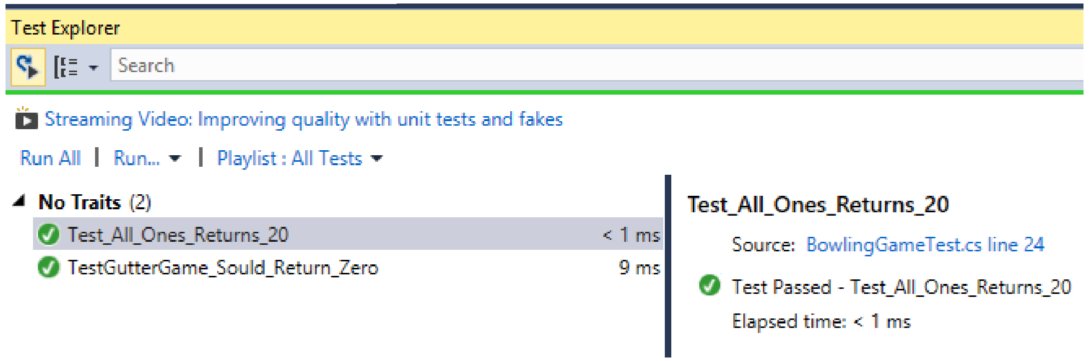
Time to refactor (DRY violation)

Test

```
[TestMethod]
✓ | 0 references
public void TestGutterGame_Sould_Return_Zero()
{
    int rolls = 20;
    int pinsKnocked = 0;

    //Arrange
    BowlingGame sut = new BowlingGame();
    //Act
    for (int i = 0; i < rolls; i++)
        sut.Roll(pinsKnocked);
    //Assert
    Assert.AreEqual(0, sut.Score());
}

[TestMethod]
✓ | 0 references
public void Test_All_Ones_Returns_20()...
```

Code

```
4 references
public class BowlingGame
{
    int score = 0;

    2 references | 2/2 passing
    public void Roll(int pins) {
        score += pins;
        return;
    }

    2 references | 2/2 passing
    public object Score() {
        return score;
    }
}
```


Test Explorer

Search

Streaming Video: Improving quality with unit tests and fakes

Run All | Run... ▾ | Playlist : All Tests ▾

```
◢ No Traits (2)
  ✓ Test_All_Ones_Returns_20                  < 1 ms
  ✓ TestGutterGame_Sould_Return_Zero            9 ms
```

Test_All_Ones_Returns_20

Source: BowlingGameTest.cs line 24

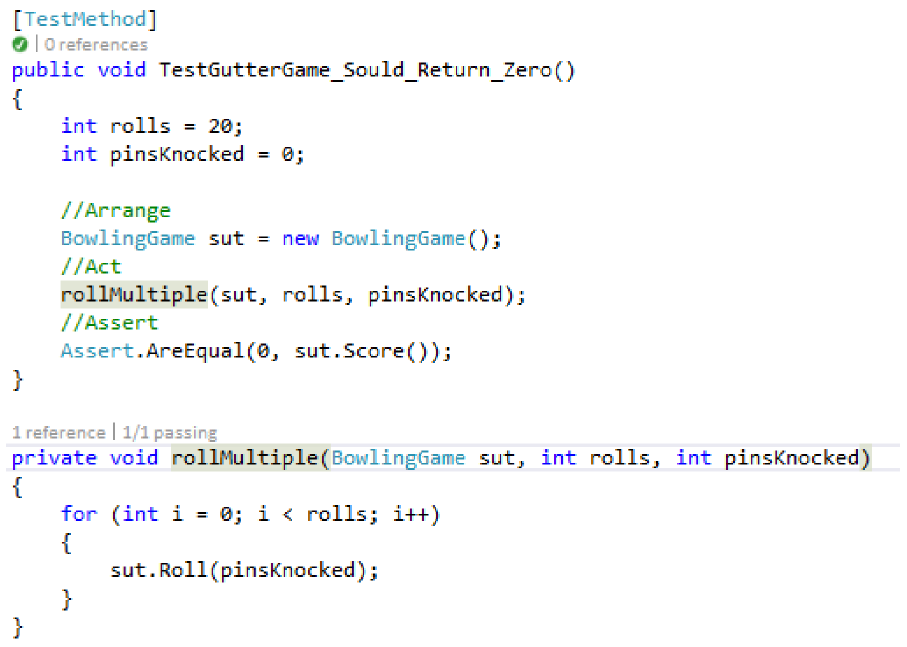
✓ Test Passed - Test_All_Ones_Returns_20

Elapsed time: < 1 ms

Refactor

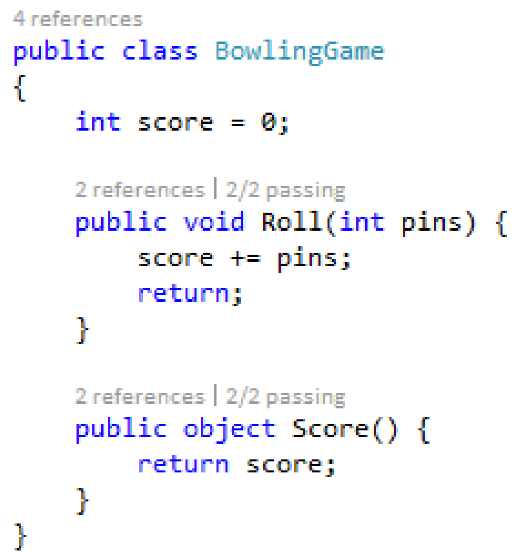
Time to refactor (DRY violation)

Test

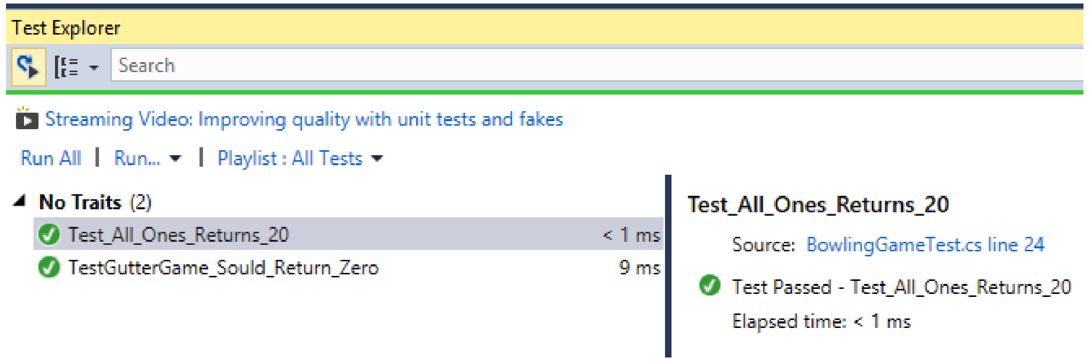


```
[TestMethod]
✓ | 0 references
public void TestGutterGame_Sould_Return_Zero()
{
    int rolls = 20;
    int pinsKnocked = 0;

    //Arrange
    BowlingGame sut = new BowlingGame();
    //Act
    rollMultiple(sut, rolls, pinsKnocked);
    //Assert
    Assert.AreEqual(0, sut.Score());
}

1 reference | 1/1 passing
private void rollMultiple(BowlingGame sut, int rolls, int pinsKnocked)
{
    for (int i = 0; i < rolls; i++)
    {
        sut.Roll(pinsKnocked);
    }
}
```

Code



```
4 references
public class BowlingGame
{
    int score = 0;

    2 references | 2/2 passing
    public void Roll(int pins) {
        score += pins;
        return;
    }

    2 references | 2/2 passing
    public object Score() {
        return score;
    }
}
```



Test Explorer

Search

Streaming Video: Improving quality with unit tests and fakes

Run All | Run... | Playlist : All Tests

```
◢ No Traits (2)
  ✓ Test_All_Ones_Returns_20             < 1 ms
  ✓ TestGutterGame_Sould_Return_Zero       9 ms
```

Test_All_Ones_Returns_20

Source: BowlingGameTest.cs line 24

✓ Test Passed - Test_All_Ones_Returns_20

Elapsed time: < 1 ms

Refactor

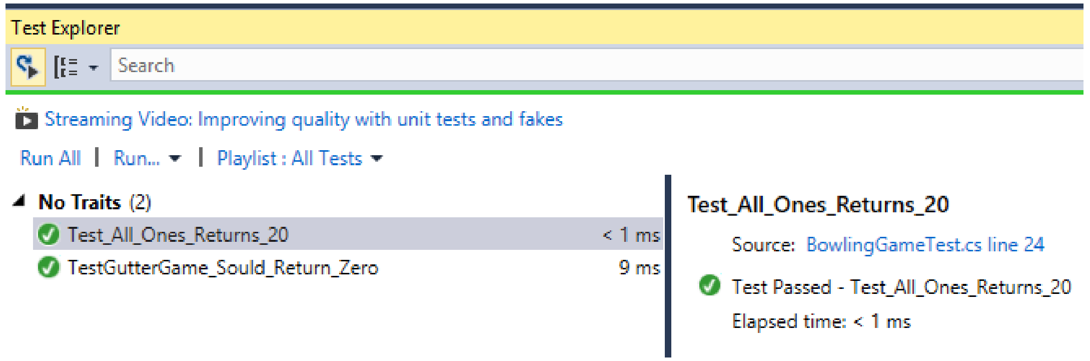
Time to refactor (DRY violation)

Test

```
[TestMethod]
✓ | 0 references
public void TestGutterGame_Sould_Return_Zero()...

2 references | 2/2 passing
private void rollMultiple(BowlingGame sut, int rolls, int pinsKnocked)...

[TestMethod]
✓ | 0 references
public void Test_All_Ones_Returns_20()
{
    int rolls = 20;
    int pinsKnocked = 1;

    //Arrange
    BowlingGame sut = new BowlingGame();
    //Act
    rollMultiple(sut, rolls, pinsKnocked);
    //Assert
    Assert.AreEqual(20, sut.Score());
}
```

Code

```
4 references
public class BowlingGame
{
    int score = 0;

    2 references | 2/2 passing
    public void Roll(int pins) {
        score += pins;
        return;
    }

    2 references | 2/2 passing
    public object Score() {
        return score;
    }
}
```


Test Explorer

Search

Streaming Video: Improving quality with unit tests and fakes

Run All | Run... ▾ | Playlist : All Tests ▾

```
◢ No Traits (2)
   ✓ Test_All_Ones_Returns_20          < 1 ms
   ✓ TestGutterGame_Sould_Return_Zero    9 ms
```

Test_All_Ones_Returns_20

Source: BowlingGameTest.cs line 24

✓ Test Passed - Test_All_Ones_Returns_20

Elapsed time: < 1 ms

Refactor

Are we meeting DRY?

What about initializing the SUT?

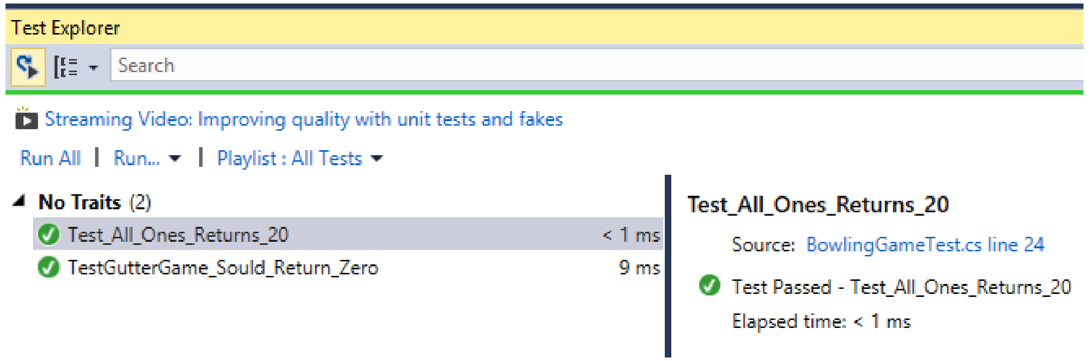
Using Test Initialization methods

Test

```
[TestMethod]
✓ | 0 references
public void TestGutterGame_Sould_Return_Zero()...

2 references | 2/2 passing
private void rollMultiple(BowlingGame sut, int rolls, int pinsKnocked)...

[TestMethod]
✓ | 0 references
public void Test_All_Ones_Returns_20()
{
    int rolls = 20;
    int pinsKnocked = 1;

    //Arrange
    BowlingGame sut = new BowlingGame();
    //Act
    rollMultiple(sut, rolls, pinsKnocked);
    //Assert
    Assert.AreEqual(20, sut.Score());
}
```

Code

```
4 references
public class BowlingGame
{
    int score = 0;

    2 references | 2/2 passing
    public void Roll(int pins) {
        score += pins;
        return;
    }

    2 references | 2/2 passing
    public object Score() {
        return score;
    }
}
```


Test Explorer

Search

Streaming Video: Improving quality with unit tests and fakes

Run All | Run... | Playlist : All Tests

No Traits (2)

✓ Test_All_Ones_Returns_20	< 1 ms
✓ TestGutterGame_Sould_Return_Zero	9 ms

Test_All_Ones_Returns_20

Source: BowlingGameTest.cs line 24

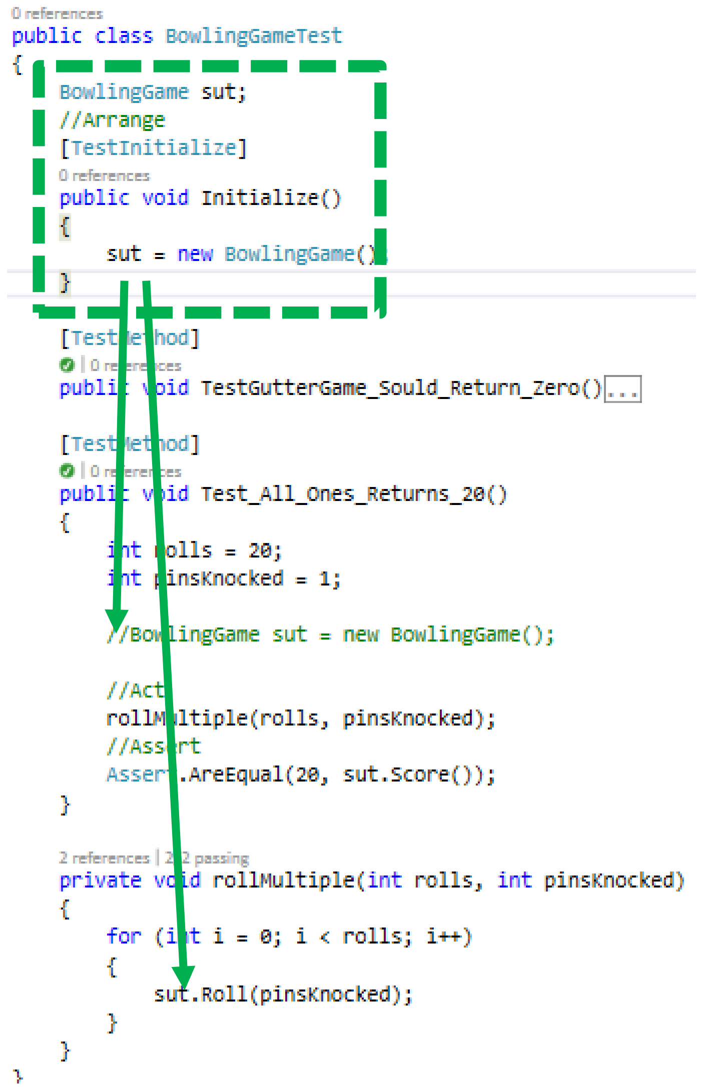
✓ Test Passed - Test_All_Ones_Returns_20

Elapsed time: < 1 ms

Refactor

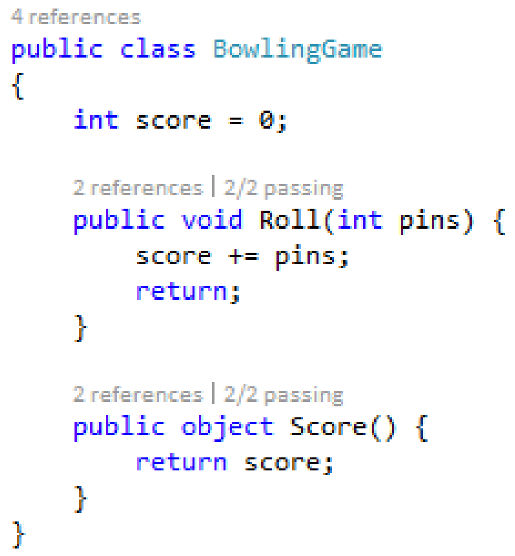
Time to refactor (DRY violation)

Test

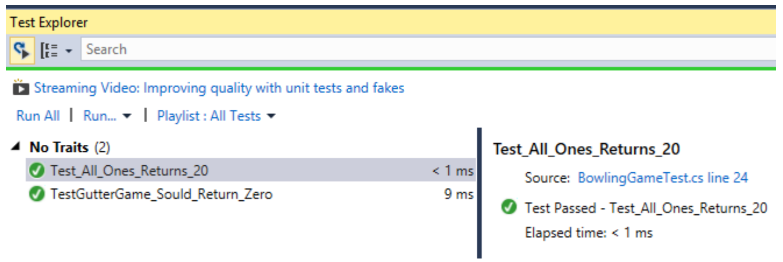


```
0 references
public class BowlingGameTest
{
    BowlingGame sut;
    //Arrange
    [TestInitialize]
    0 references
    public void Initialize()
    {
        sut = new BowlingGame();
    }
    [TestMethod]
    0 references
    public void TestGutterGame_Sould_Return_Zero()[...]

    [TestMethod]
    0 references
    public void Test_All_Ones_Returns_20()
    {
        int rolls = 20;
        int pinsKnocked = 1;

        //BowlingGame sut = new BowlingGame();

        //Act
        rollMultiple(rolls, pinsKnocked);
        //Assert
        Assert.AreEqual(20, sut.Score());
    }

    2 references | 2/2 passing
    private void rollMultiple(int rolls, int pinsKnocked)
    {
        for (int i = 0; i < rolls; i++)
        {
            sut.Roll(pinsKnocked);
        }
    }
}
```

Code



```
4 references
public class BowlingGame
{
    int score = 0;

    2 references | 2/2 passing
    public void Roll(int pins) {
        score += pins;
        return;
    }

    2 references | 2/2 passing
    public object Score() {
        return score;
    }
}
```



Test Explorer

Search

Streaming Video: Improving quality with unit tests and fakes

Run All | Run... ▾ | Playlist : All Tests ▾

```
▲ No Traits (2)
  ✓ Test_All_Ones_Returns_20                < 1 ms
  ✓ TestGutterGame_Sould_Return_Zero          9 ms
```

Test_All_Ones_Returns_20

Source: BowlingGameTest.cs line 24

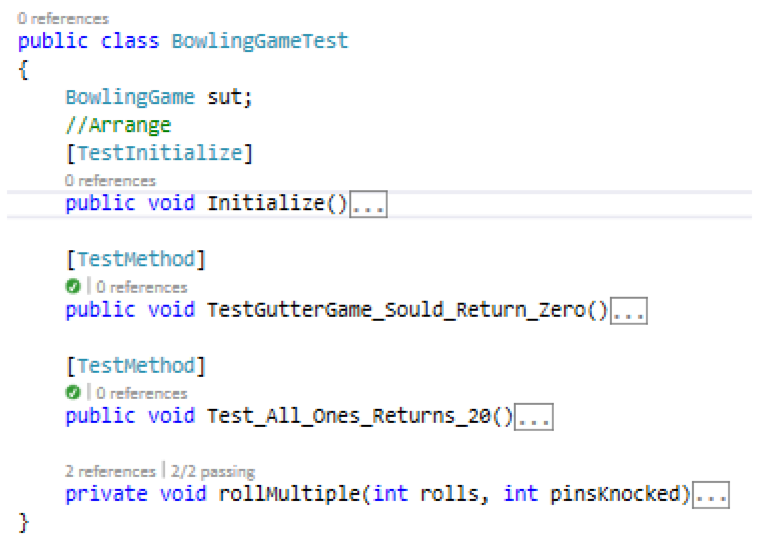
✓ Test Passed - Test_All_Ones_Returns_20

Elapsed time: < 1 ms

Refactor

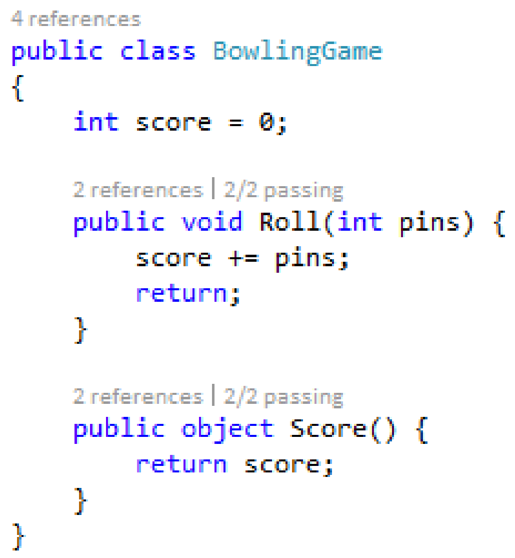
Time to refactor (DRY violation)

Test

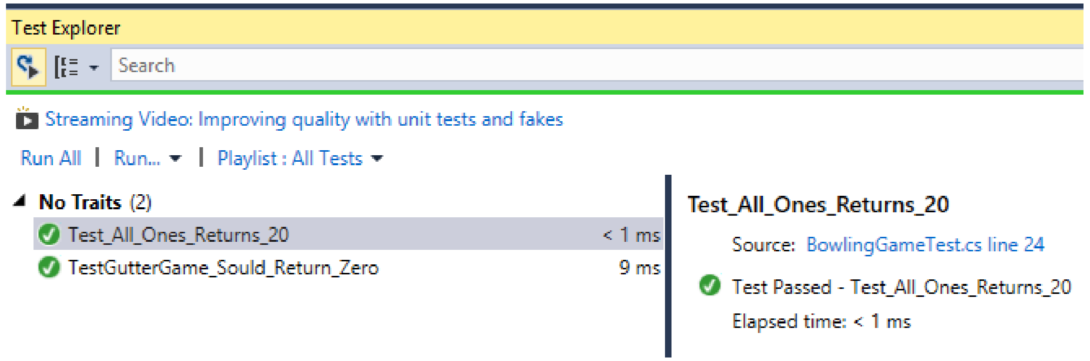


```
0 references
public class BowlingGameTest
{
    BowlingGame sut;
    //Arrange
    [TestInitialize]
    0 references
    public void Initialize()...

    [TestMethod]
    | 0 references
    public void TestGutterGame_Sould_Return_Zero()...

    [TestMethod]
    | 0 references
    public void Test_All_Ones_Returns_20()...

    2 references | 2/2 passing
    private void rollMultiple(int rolls, int pinsKnocked)...
}
```

Code



```
4 references
public class BowlingGame
{
    int score = 0;

    2 references | 2/2 passing
    public void Roll(int pins) {
        score += pins;
        return;
    }

    2 references | 2/2 passing
    public object Score() {
        return score;
    }
}
```



```
Test Explorer
Search
Streaming Video: Improving quality with unit tests and fakes
Run All | Run... | Playlist : All Tests
No Traits (2)
  Test_All_Ones_Returns_20                 < 1 ms
  TestGutterGame_Sould_Return_Zero           9 ms

Test_All_Ones_Returns_20
  Source: BowlingGameTest.cs line 24
  Test Passed - Test_All_Ones_Returns_20
  Elapsed time: < 1 ms
```

Refactor

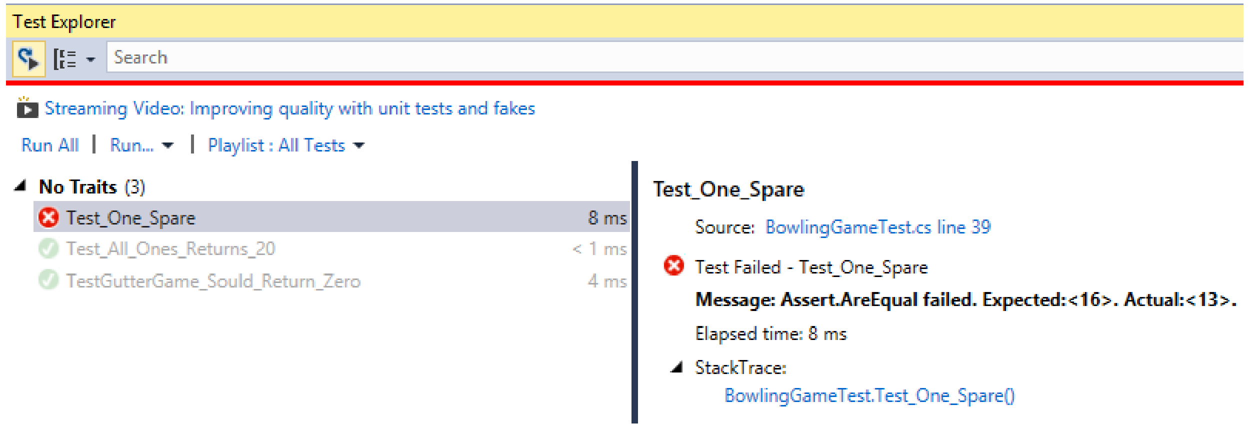
The third test: One Spare

Test

```
[TestMethod]
 | 0 references
public void Test_One_Spare()
{
    //Act
    sut.Roll(5);
    sut.Roll(5); //Spare here
    sut.Roll(3);

    rollMultiple(17, 0);

    //Assert
    Assert.AreEqual(16, sut.Score());
}
```

Code

```
4 references
public class BowlingGame
{
    int score = 0;

    2 references | 2/2 passing
    public void Roll(int pins) {
        score += pins;
        return;
    }

    2 references | 2/2 passing
    public object Score() {
        return score;
    }
}
```



Test Explorer

Search

Streaming Video: Improving quality with unit tests and fakes

Run All | Run... | Playlist : All Tests

```
No Traits (3)
  Test_One_Spare                          8 ms
  Test_All_Ones_Returns_20              < 1 ms
  TestGutterGame_Sould_Return_Zero        4 ms
```

Test_One_Spare

Source: BowlingGameTest.cs line 39

Test Failed - Test_One_Spare

Message: Assert.AreEqual failed. Expected:<16>. Actual:<13>.

Elapsed time: 8 ms

StackTrace:
BowlingGameTest.Test_One_Spare()

Red

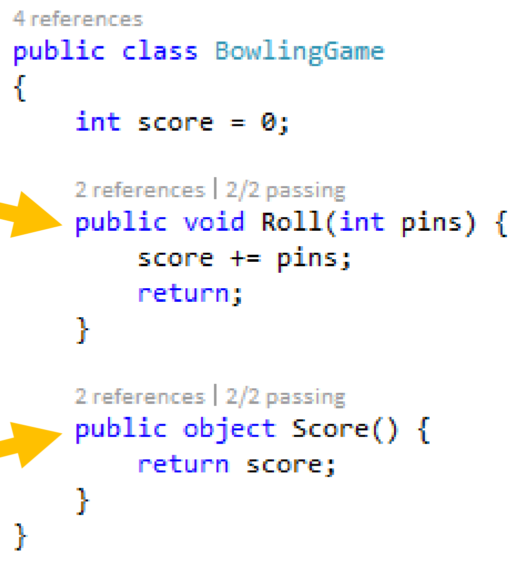
Time to rethink our design…

The third test: One Spare

Roll calculates the score. Should?

Score implies calculation. But nothing is done.

Code



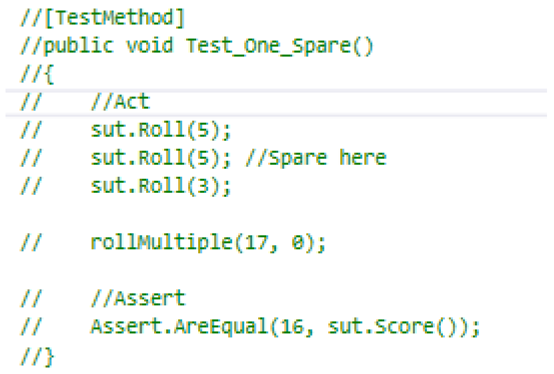
```
4 references
public class BowlingGame
{
    int score = 0;

    2 references | 2/2 passing
    public void Roll(int pins) {
        score += pins;
        return;
    }

    2 references | 2/2 passing
    public object Score() {
        return score;
    }
}
```

Let’s refactor….

Remember: keep your test suite green while refactoring.

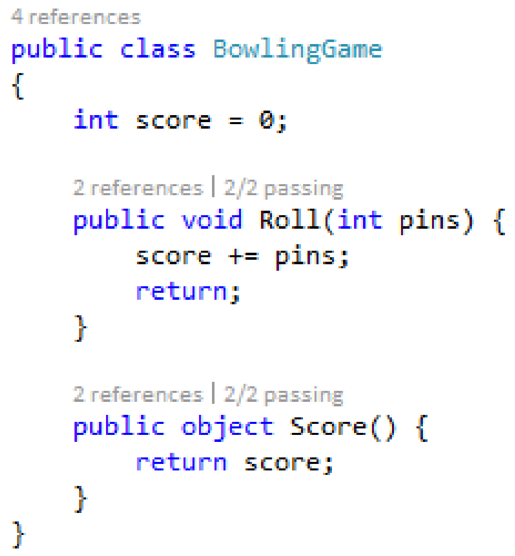
Refactoring current functionality

Test

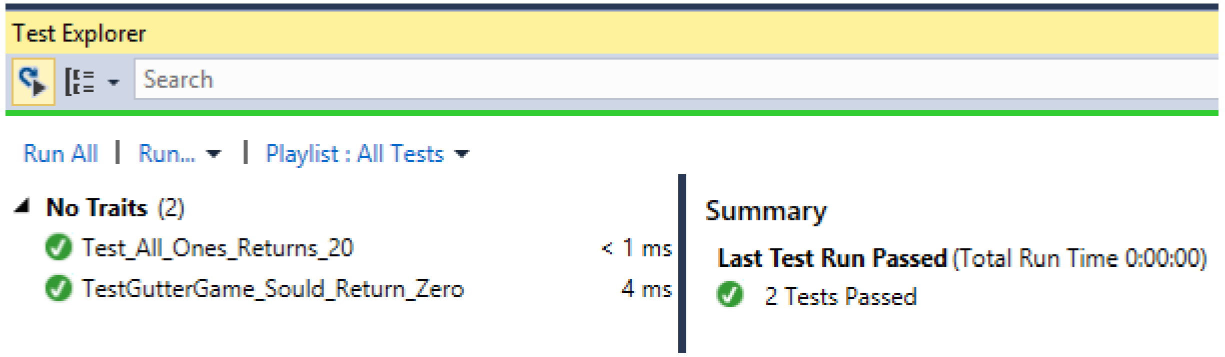


```
//[TestMethod]
//public void Test_One_Spare()
//{
//    //Act
//    sut.Roll(5);
//    sut.Roll(5); //Spare here
//    sut.Roll(3);

//    rollMultiple(17, 0);

//    //Assert
//    Assert.AreEqual(16, sut.Score());
//}
```

Code



```
4 references
public class BowlingGame
{
    int score = 0;

    2 references | 2/2 passing
    public void Roll(int pins) {
        score += pins;
        return;
    }

    2 references | 2/2 passing
    public object Score() {
        return score;
    }
}
```

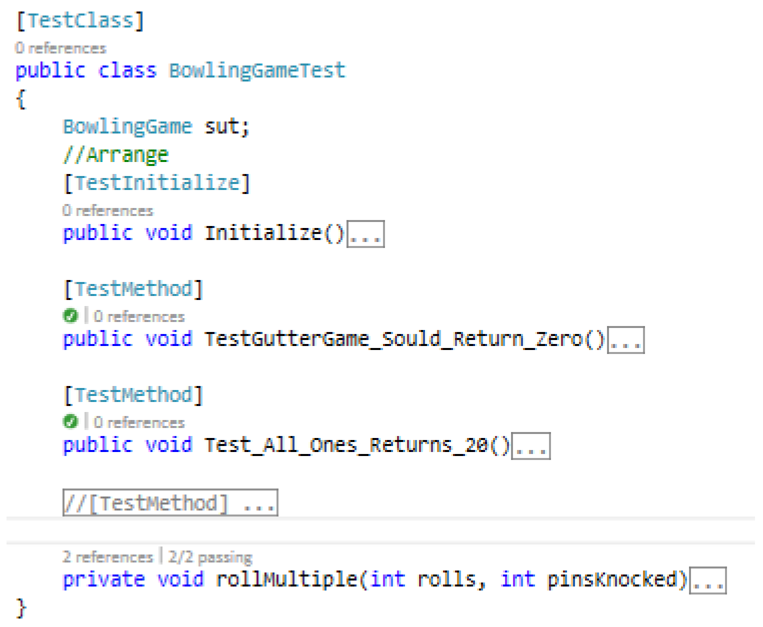


```
Test Explorer
Search
Run All | Run... ▾ | Playlist : All Tests ▾

◢ No Traits (2)                                  Summary
  ✓ Test_All_Ones_Returns_20            < 1 ms    Last Test Run Passed (Total Run Time 0:00:00)
  ✓ TestGutterGame_Sould_Return_Zero      4 ms    ✓ 2 Tests Passed
```

Refactor

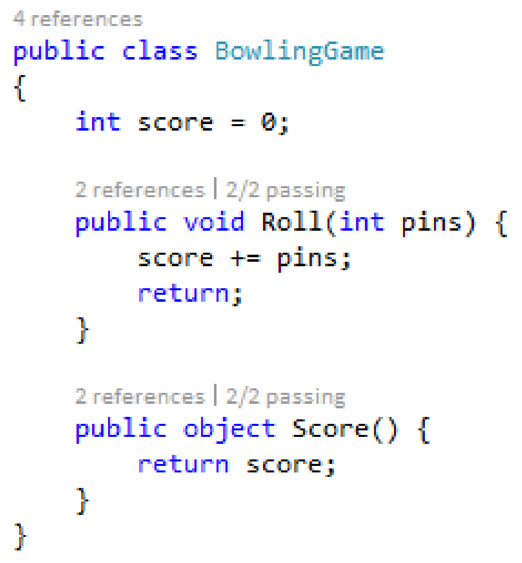
Refactoring current functionality

Test

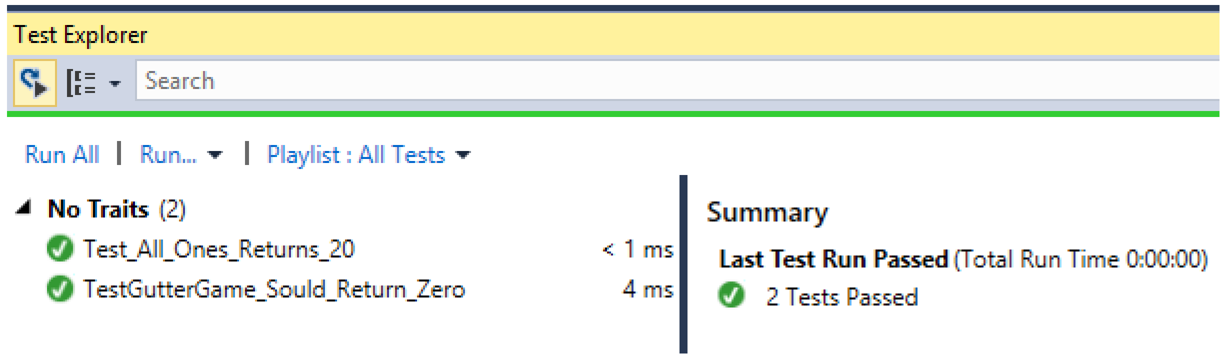


```
[TestClass]
0 references
public class BowlingGameTest
{
    BowlingGame sut;
    //Arrange
    [TestInitialize]
    0 references
    public void Initialize()[...]

    [TestMethod]
    ✓ | 0 references
    public void TestGutterGame_Sould_Return_Zero()[...]

    [TestMethod]
    ✓ | 0 references
    public void Test_All_Ones_Returns_20()[...]

    //[TestMethod] ...

    2 references | 2/2 passing
    private void rollMultiple(int rolls, int pinsKnocked)[...]
}
```

Code



```
4 references
public class BowlingGame
{
    int score = 0;

    2 references | 2/2 passing
    public void Roll(int pins) {
        score += pins;
        return;
    }

    2 references | 2/2 passing
    public object Score() {
        return score;
    }
}
```



Test Explorer

Search

Run All | Run... ▾ | Playlist : All Tests ▾

```
◢ No Traits (2)
  ✓ Test_All_Ones_Returns_20              < 1 ms
  ✓ TestGutterGame_Sould_Return_Zero        4 ms
```

Summary

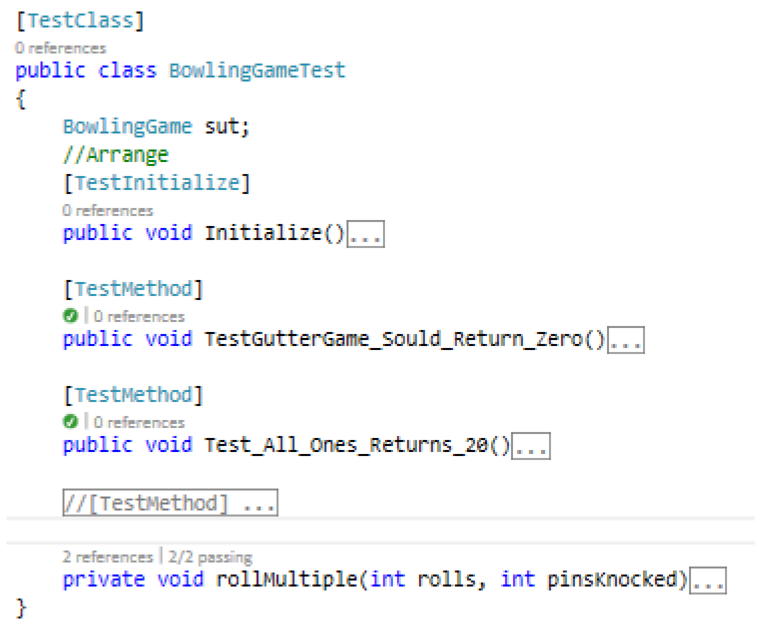
Last Test Run Passed (Total Run Time 0:00:00)

✓ 2 Tests Passed

Refactor

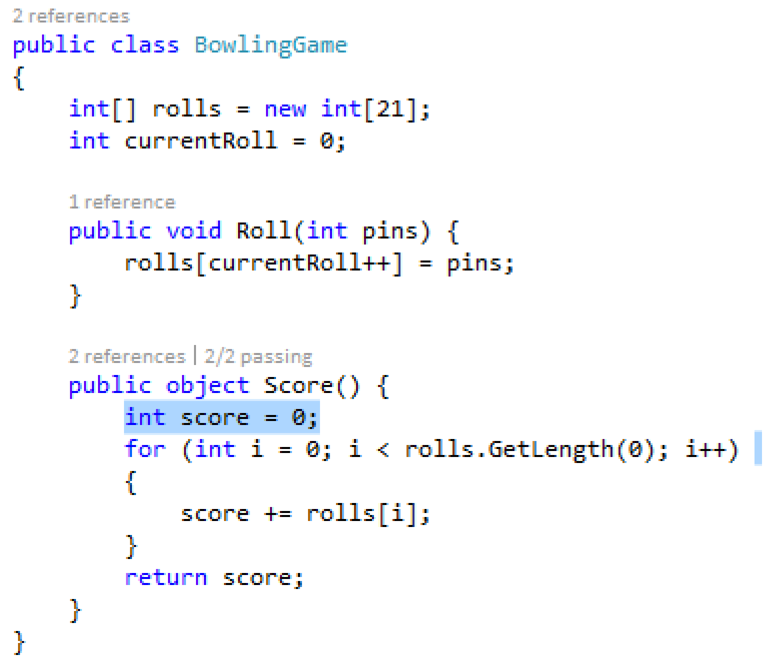
Refactoring current functionality

Test

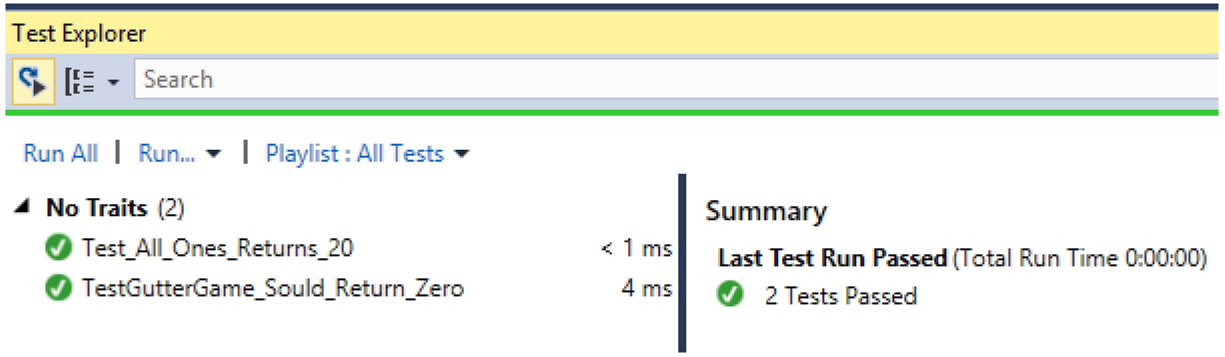


```
[TestClass]
0 references
public class BowlingGameTest
{
    BowlingGame sut;
    //Arrange
    [TestInitialize]
    0 references
    public void Initialize()[...]

    [TestMethod]
    ✓ | 0 references
    public void TestGutterGame_Sould_Return_Zero()[...]

    [TestMethod]
    ✓ | 0 references
    public void Test_All_Ones_Returns_20()[...]

    //[TestMethod] ...

    2 references | 2/2 passing
    private void rollMultiple(int rolls, int pinsKnocked)[...]
}
```

Code



```
2 references
public class BowlingGame
{
    int[] rolls = new int[21];
    int currentRoll = 0;

    1 reference
    public void Roll(int pins) {
        rolls[currentRoll++] = pins;
    }

    2 references | 2/2 passing
    public object Score() {
        int score = 0;
        for (int i = 0; i < rolls.GetLength(0); i++)
        {
            score += rolls[i];
        }
        return score;
    }
}
```



Test Explorer

Search

Run All | Run... | Playlist : All Tests

```
No Traits (2)
  ✓ Test_All_Ones_Returns_20          < 1 ms
  ✓ TestGutterGame_Sould_Return_Zero    4 ms
```

Summary

Last Test Run Passed (Total Run Time 0:00:00)

✓ 2 Tests Passed

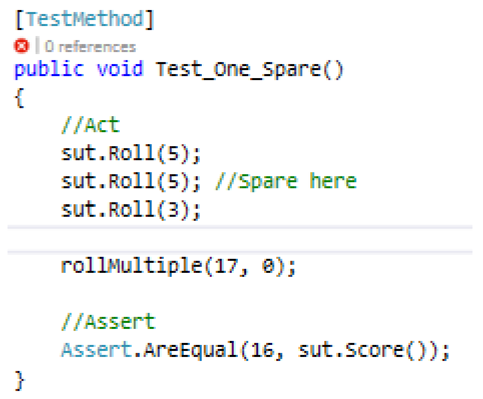
Refactor

Done refactoring….

We can move on with our next test.

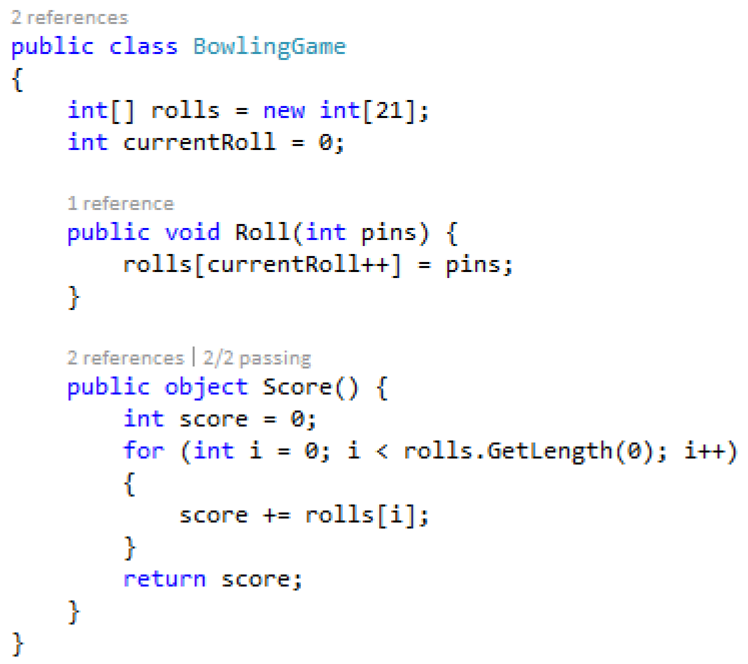
The third test: One Spare

Test

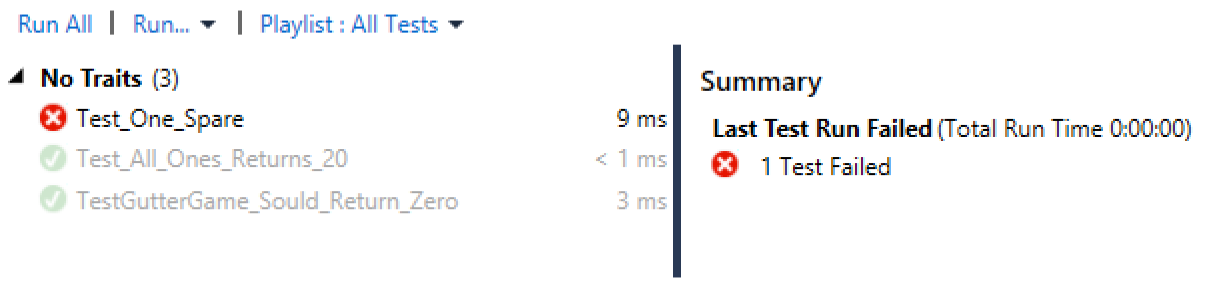


```
[TestMethod]
| 0 references
public void Test_One_Spare()
{
    //Act
    sut.Roll(5);
    sut.Roll(5); //Spare here
    sut.Roll(3);

    rollMultiple(17, 0);

    //Assert
    Assert.AreEqual(16, sut.Score());
}
```

Code



```
2 references
public class BowlingGame
{
    int[] rolls = new int[21];
    int currentRoll = 0;

    1 reference
    public void Roll(int pins) {
        rolls[currentRoll++] = pins;
    }

    2 references | 2/2 passing
    public object Score() {
        int score = 0;
        for (int i = 0; i < rolls.GetLength(0); i++)
        {
            score += rolls[i];
        }
        return score;
    }
}
```



```
◢ No Traits (3)
  ✗ Test_One_Spare                        9 ms
  ✓ Test_All_Ones_Returns_20            < 1 ms
  ✓ TestGutterGame_Sould_Return_Zero      3 ms
```

Summary

Last Test Run Failed (Total Run Time 0:00:00)

✗ 1 Test Failed

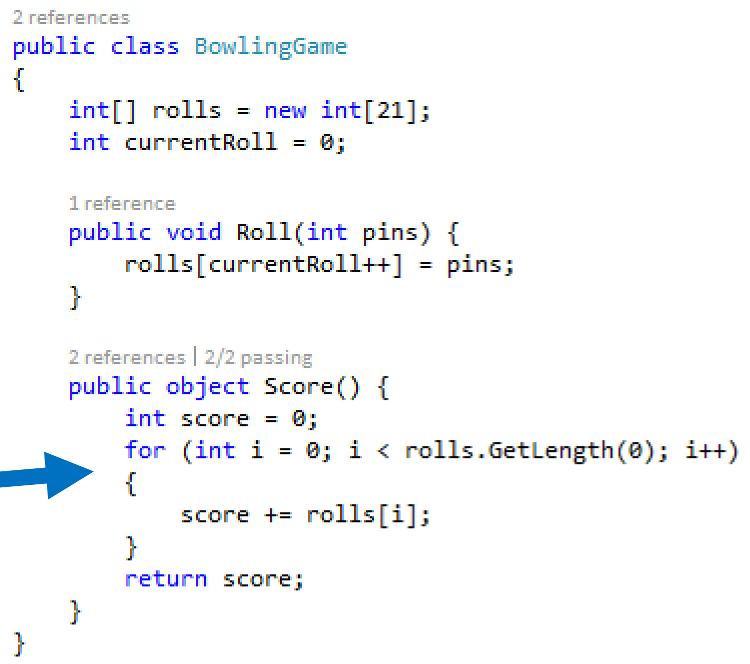
Red

The third test: One Spare

Current implementation is unaware of “frame” concept.

Difficult to calculate spare and strike bonus

Code



```
2 references
public class BowlingGame
{
    int[] rolls = new int[21];
    int currentRoll = 0;

    1 reference
    public void Roll(int pins) {
        rolls[currentRoll++] = pins;
    }

    2 references | 2/2 passing
    public object Score() {
        int score = 0;
        for (int i = 0; i < rolls.GetLength(0); i++)
        {
            score += rolls[i];
        }
        return score;
    }
}
```

Time to refactor (again)….

Remember: keep your test suite green while refactoring.

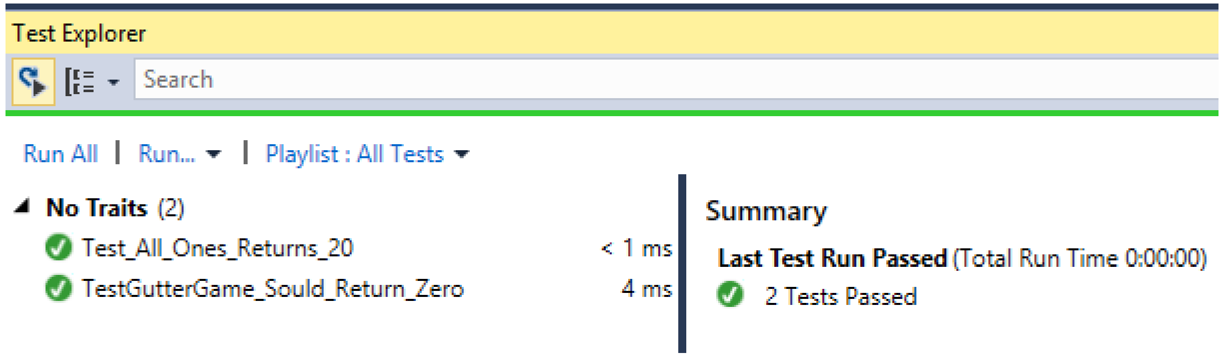
Refactoring current functionality

Test

```
//[TestMethod]
//public void Test_One_Spare()
//{
//    //Act
//    sut.Roll(5);
//    sut.Roll(5); //Spare here
//    sut.Roll(3);

//    rollMultiple(17, 0);

//    //Assert
//    Assert.AreEqual(16, sut.Score());
//}
```

Code

```
2 references
public class BowlingGame
{
    int[] rolls = new int[21];
    int currentRoll = 0;

    1 reference
    public void Roll(int pins) {
        rolls[currentRoll++] = pins;
    }

    2 references | 2/2 passing
    public object Score() {
        int score = 0;
        for (int i = 0; i < rolls.GetLength(0); i++)
        {
            score += rolls[i];
        }
        return score;
    }
}
```



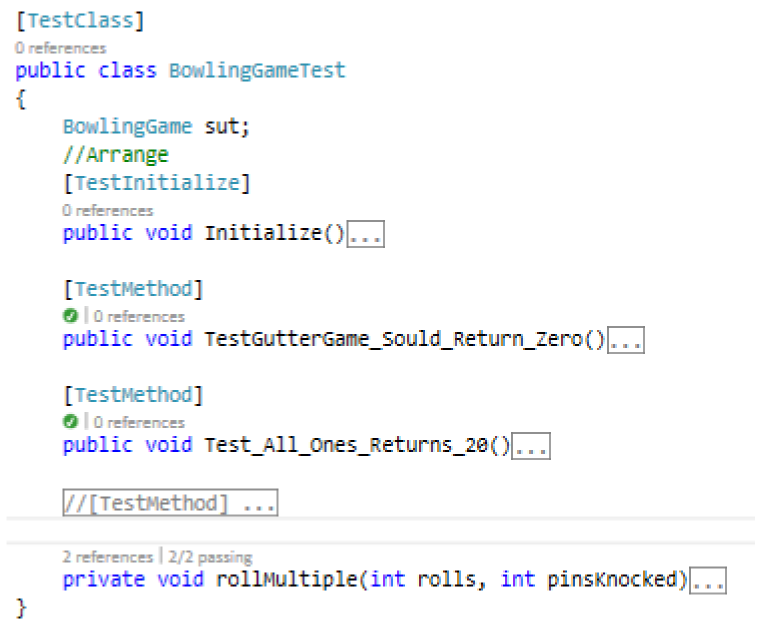
Test Explorer

```
No Traits (2)
  Test_All_Ones_Returns_20           < 1 ms
  TestGutterGame_Sould_Return_Zero     4 ms

Summary
Last Test Run Passed (Total Run Time 0:00:00)
  2 Tests Passed
```

Refactor

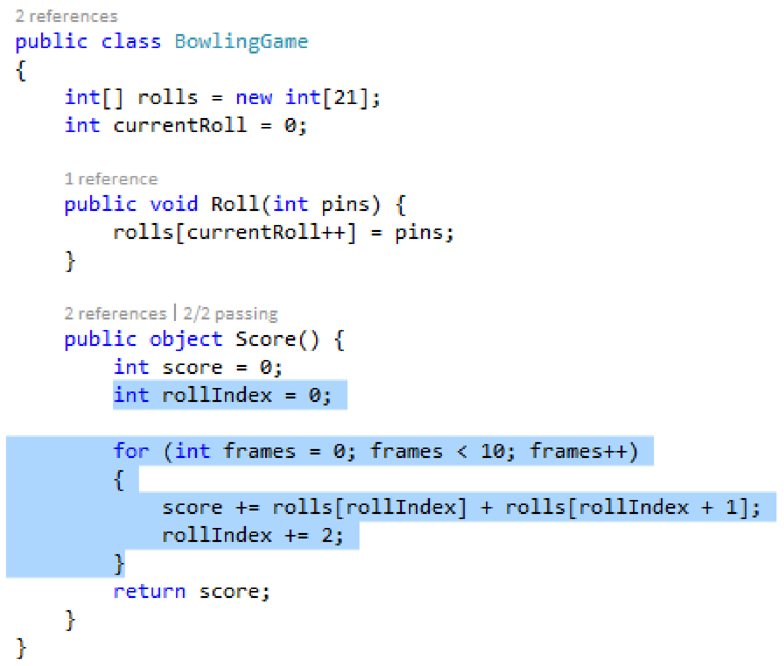
Refactoring current functionality

Test

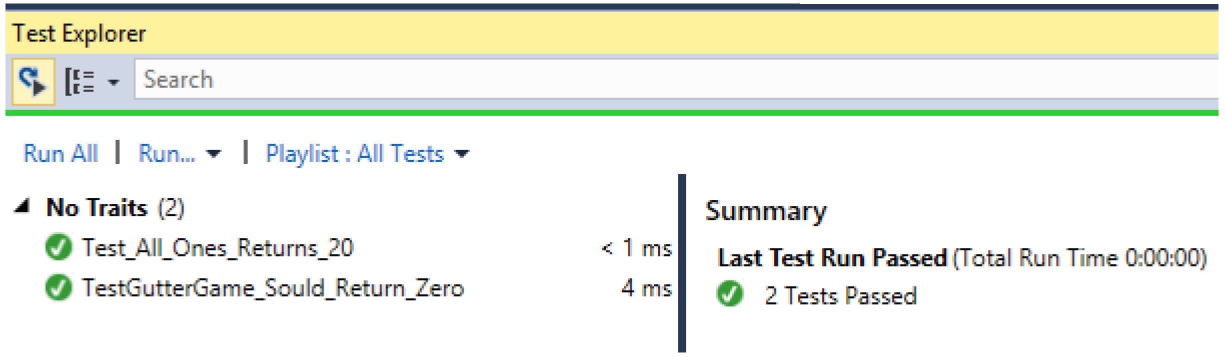


```
[TestClass]
0 references
public class BowlingGameTest
{
    BowlingGame sut;
    //Arrange
    [TestInitialize]
    0 references
    public void Initialize()...

    [TestMethod]
    ✓ | 0 references
    public void TestGutterGame_Sould_Return_Zero()...

    [TestMethod]
    ✓ | 0 references
    public void Test_All_Ones_Returns_20()...

    //[TestMethod] ...

    2 references | 2/2 passing
    private void rollMultiple(int rolls, int pinsKnocked)...
}
```

Code



```
2 references
public class BowlingGame
{
    int[] rolls = new int[21];
    int currentRoll = 0;

    1 reference
    public void Roll(int pins) {
        rolls[currentRoll++] = pins;
    }

    2 references | 2/2 passing
    public object Score() {
        int score = 0;
        int rollIndex = 0;

        for (int frames = 0; frames < 10; frames++)
        {
            score += rolls[rollIndex] + rolls[rollIndex + 1];
            rollIndex += 2;
        }
        return score;
    }
}
```



```
No Traits (2)
  ✓ Test_All_Ones_Returns_20            < 1 ms
  ✓ TestGutterGame_Sould_Return_Zero      4 ms

Summary
Last Test Run Passed (Total Run Time 0:00:00)
✓ 2 Tests Passed
```

Refactor

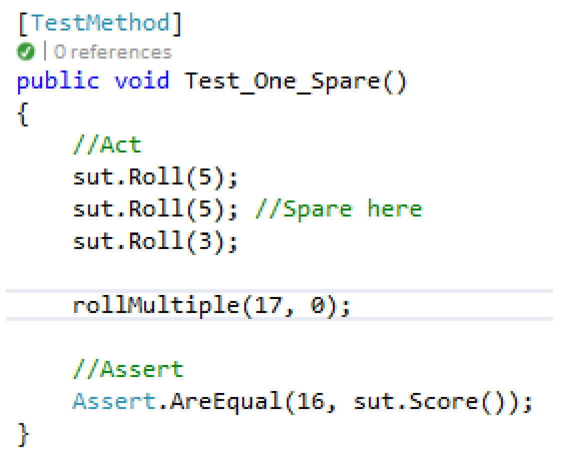
Done refactoring….

Everything is still green.

We can move on with Test 3.

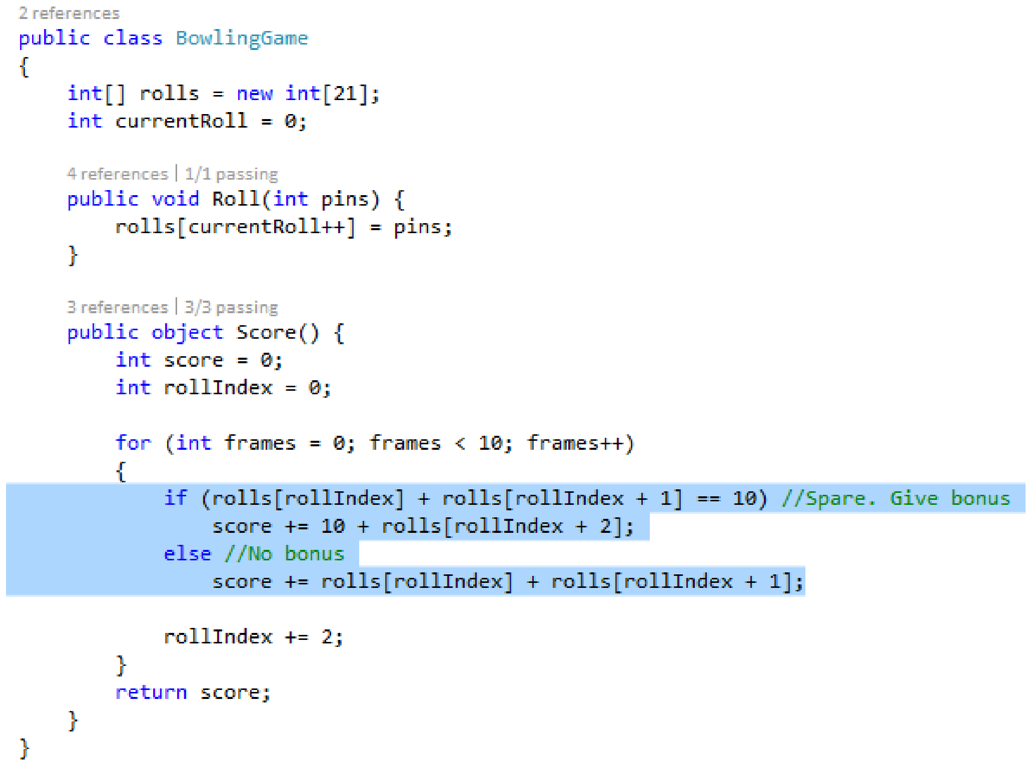
The third test: One Spare

Test

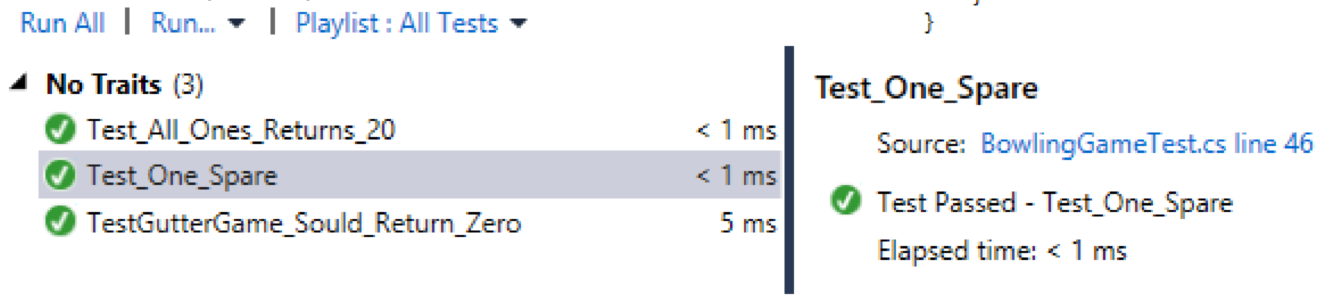


```
[TestMethod]
✓ | 0 references
public void Test_One_Spare()
{
    //Act
    sut.Roll(5);
    sut.Roll(5); //Spare here
    sut.Roll(3);

    rollMultiple(17, 0);

    //Assert
    Assert.AreEqual(16, sut.Score());
}
```

Code



```
2 references
public class BowlingGame
{
    int[] rolls = new int[21];
    int currentRoll = 0;

    4 references | 1/1 passing
    public void Roll(int pins) {
        rolls[currentRoll++] = pins;
    }

    3 references | 3/3 passing
    public object Score() {
        int score = 0;
        int rollIndex = 0;

        for (int frames = 0; frames < 10; frames++)
        {
            if (rolls[rollIndex] + rolls[rollIndex + 1] == 10) //Spare. Give bonus
                score += 10 + rolls[rollIndex + 2];
            else //No bonus
                score += rolls[rollIndex] + rolls[rollIndex + 1];

            rollIndex += 2;
        }
        return score;
    }
}
```



```
No Traits (3)
  ✓ Test_All_Ones_Returns_20              < 1 ms
  ✓ Test_One_Spare                        < 1 ms
  ✓ TestGutterGame_Sould_Return_Zero        5 ms
```

Test_One_Spare

Source: BowlingGameTest.cs line 46

✓ Test Passed - Test_One_Spare

Elapsed time: < 1 ms

Green

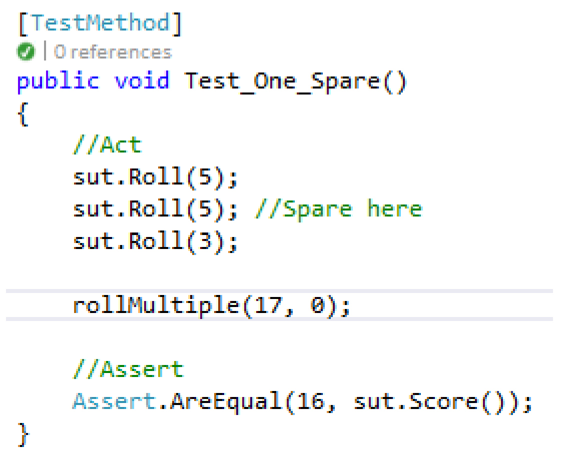
Time to put our refactoring hat….

Pending Design Concerns

- Way we generate spares in test (//spare here comment)
- Way we check for spare existence (//Spare. Give bonus)

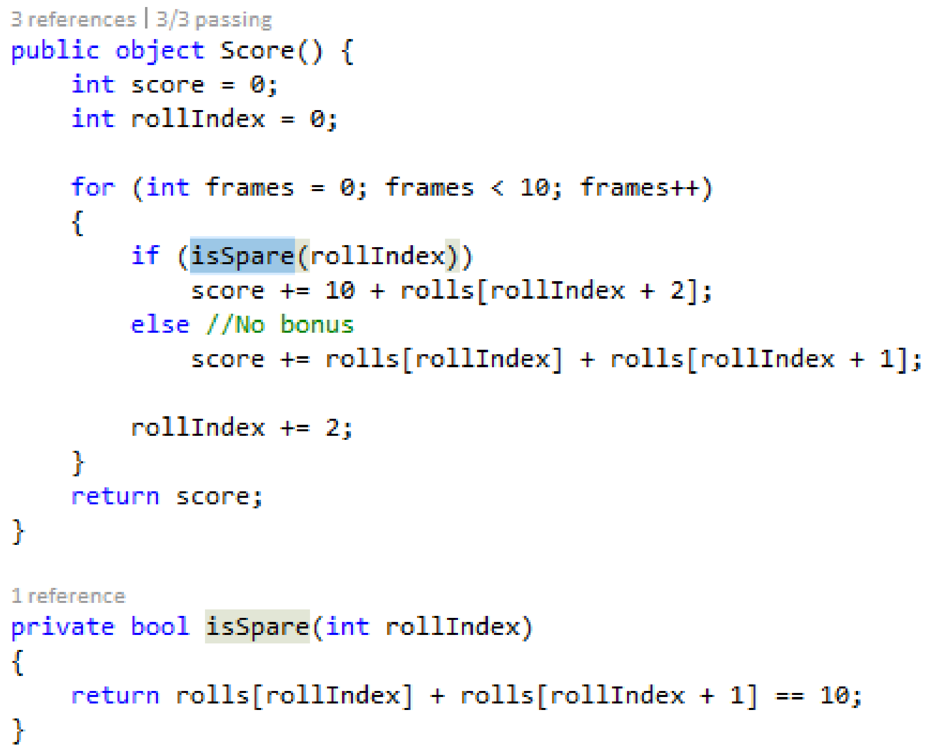
Refactoring spare calculation

Test

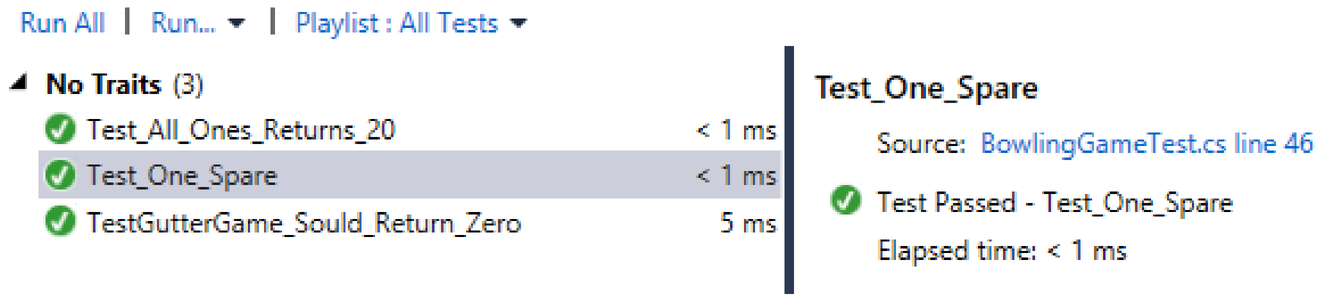


```
[TestMethod]
✓ | 0 references
public void Test_One_Spare()
{
    //Act
    sut.Roll(5);
    sut.Roll(5); //Spare here
    sut.Roll(3);

    rollMultiple(17, 0);

    //Assert
    Assert.AreEqual(16, sut.Score());
}
```

Code



```
3 references | 3/3 passing
public object Score() {
    int score = 0;
    int rollIndex = 0;

    for (int frames = 0; frames < 10; frames++)
    {
        if (isSpare(rollIndex))
            score += 10 + rolls[rollIndex + 2];
        else //No bonus
            score += rolls[rollIndex] + rolls[rollIndex + 1];

        rollIndex += 2;
    }
    return score;
}

1 reference
private bool isSpare(int rollIndex)
{
    return rolls[rollIndex] + rolls[rollIndex + 1] == 10;
}
```



Run All | Run... ▾ | Playlist : All Tests ▾

◢ **No Traits** (3)

✓ Test_All_Ones_Returns_20	< 1 ms
✓ Test_One_Spare	< 1 ms
✓ TestGutterGame_Sould_Return_Zero	5 ms

Test_One_Spare

Source: BowlingGameTest.cs line 46

✓ Test Passed - Test_One_Spare

Elapsed time: < 1 ms

Refactor

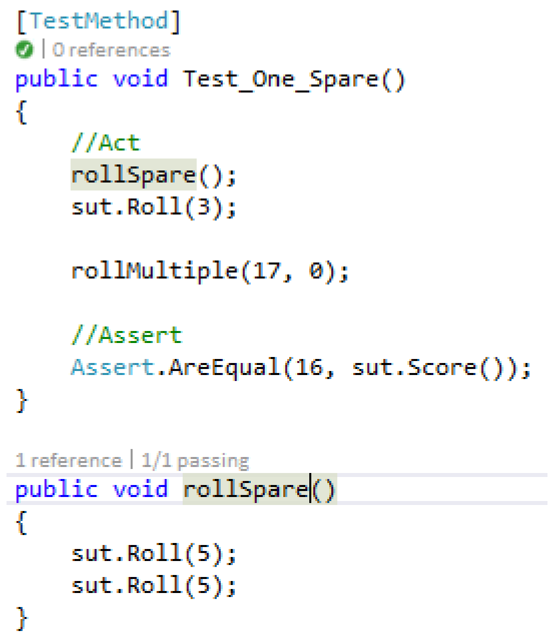
Time to put our refactoring hat….

Pending Design Concerns

- Way we generate spares in test (//spare here comment)
- ~~Way we check for spare existence (//Spare. Give bonus)~~

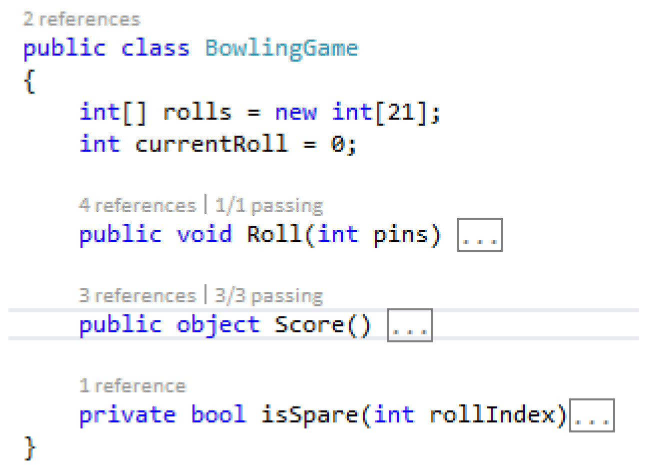
Refactor 3rd Test

Test

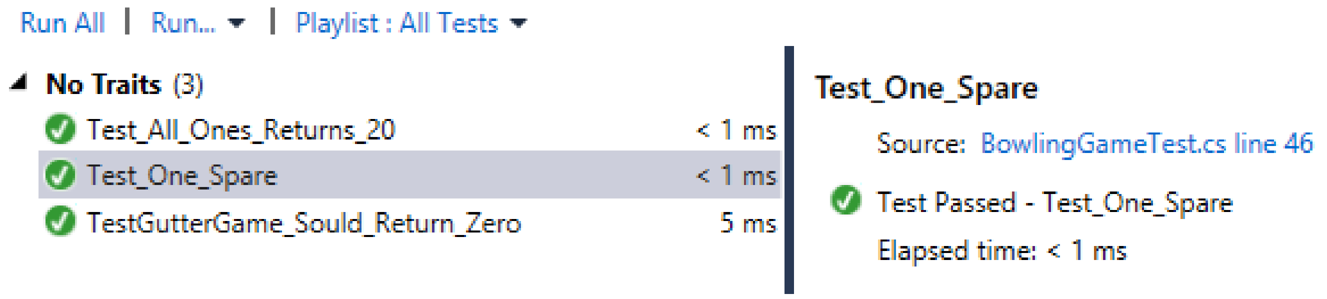


```
[TestMethod]
✓ | 0 references
public void Test_One_Spare()
{
    //Act
    rollSpare();
    sut.Roll(3);

    rollMultiple(17, 0);

    //Assert
    Assert.AreEqual(16, sut.Score());
}

1 reference | 1/1 passing
public void rollSpare()
{
    sut.Roll(5);
    sut.Roll(5);
}
```

Code



```
2 references
public class BowlingGame
{
    int[] rolls = new int[21];
    int currentRoll = 0;

    4 references | 1/1 passing
    public void Roll(int pins) ...

    3 references | 3/3 passing
    public object Score() ...

    1 reference
    private bool isSpare(int rollIndex)...
}
```



```
◢ No Traits (3)
  ✓ Test_All_Ones_Returns_20            < 1 ms
  ✓ Test_One_Spare                      < 1 ms
  ✓ TestGutterGame_Sould_Return_Zero      5 ms

Test_One_Spare
  Source: BowlingGameTest.cs line 46
  ✓ Test Passed - Test_One_Spare
  Elapsed time: < 1 ms
```

Refactor

We addressed pending design concerns….

Pending Design Concerns

- ~~Way we generate spares in test (//spare here comment)~~
- ~~Way we check for spare existence (//Spare. Give bonus)~~

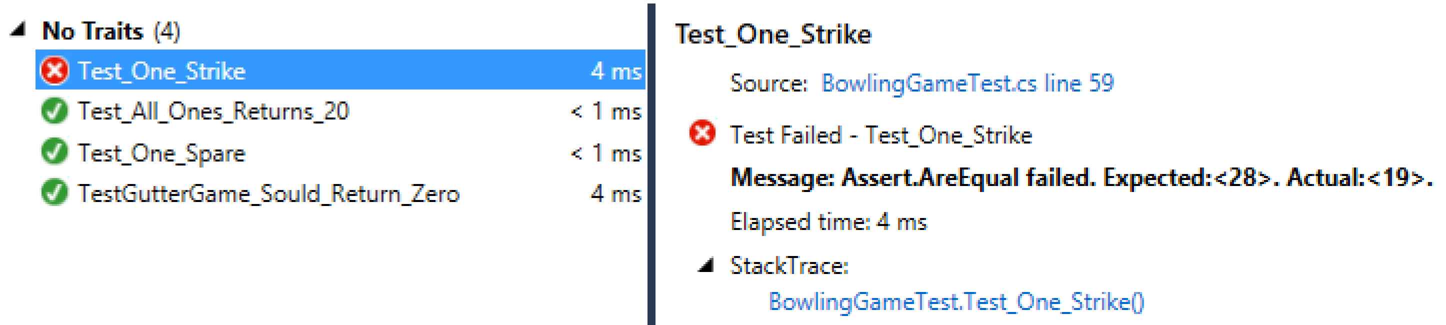
The 4th test: Strike bonus

Test

```
[TestMethod]
| 0 references
public void Test_One_Strike()
{
    //Act
    sut.Roll(10); //Rolls a strike
    sut.Roll(4);
    sut.Roll(5);
    rollMultiple(16, 0);

    //Assert
    //10 points for the strike, plus 9 points bonus,
    //plus 9 points from the additional rolls
    Assert.AreEqual(28, sut.Score());
}
```

Code

```
2 references
public class BowlingGame
{
    int[] rolls = new int[21];
    int currentRoll = 0;

    4 references | 1/1 passing
    public void Roll(int pins) ...

    3 references | 3/3 passing
    public object Score() ...

    1 reference
    private bool isSpare(int rollIndex)...
}
```



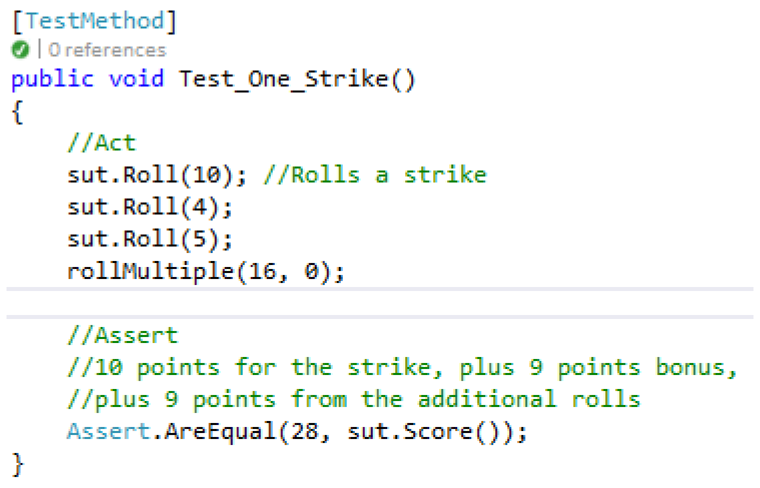
```
No Traits (4)
  Test_One_Strike                      4 ms
  Test_All_Ones_Returns_20           < 1 ms
  Test_One_Spare                     < 1 ms
  TestGutterGame_Sould_Return_Zero     4 ms
```

```
Test_One_Strike
  Source: BowlingGameTest.cs line 59
  Test Failed - Test_One_Strike
  Message: Assert.AreEqual failed. Expected:<28>. Actual:<19>.
  Elapsed time: 4 ms
  StackTrace:
    BowlingGameTest.Test_One_Strike()
```

Red

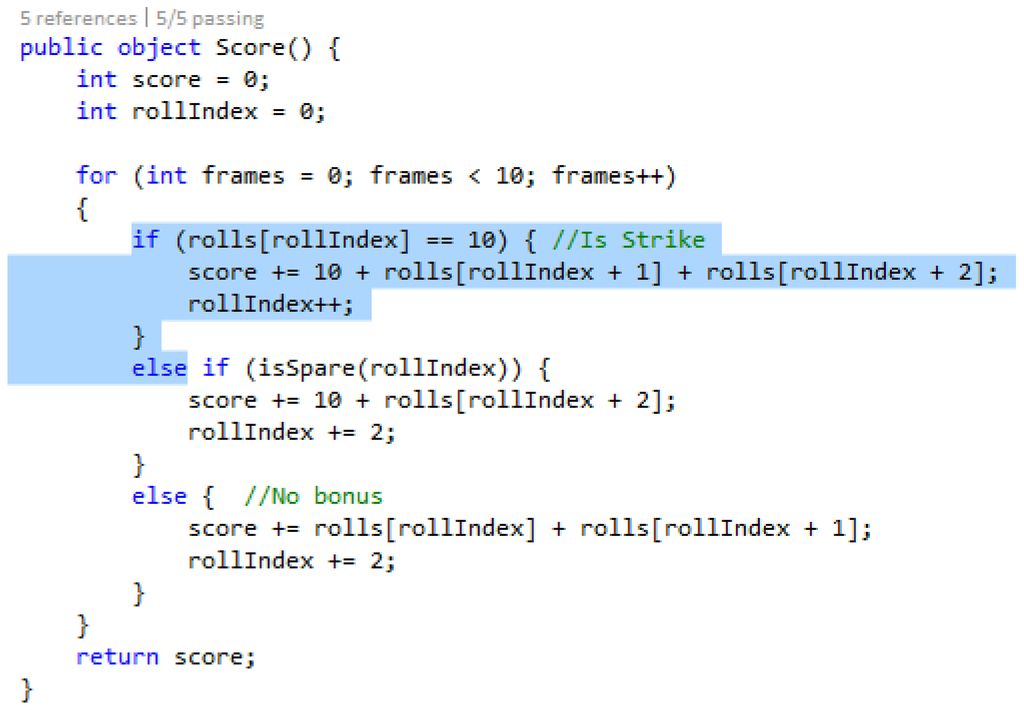
The 4th test: Strike bonus

Test



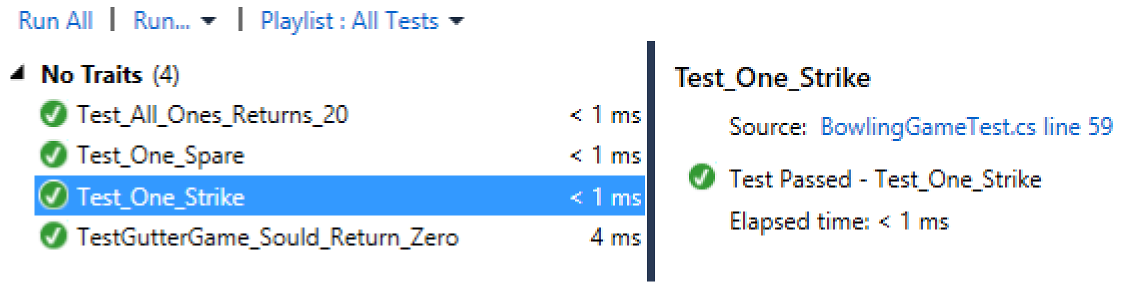
```
[TestMethod]
✓ | 0 references
public void Test_One_Strike()
{
    //Act
    sut.Roll(10); //Rolls a strike
    sut.Roll(4);
    sut.Roll(5);
    rollMultiple(16, 0);

    //Assert
    //10 points for the strike, plus 9 points bonus,
    //plus 9 points from the additional rolls
    Assert.AreEqual(28, sut.Score());
}
```

Code



```
5 references | 5/5 passing
public object Score() {
    int score = 0;
    int rollIndex = 0;

    for (int frames = 0; frames < 10; frames++)
    {
        if (rolls[rollIndex] == 10) { //Is Strike
            score += 10 + rolls[rollIndex + 1] + rolls[rollIndex + 2];
            rollIndex++;
        }
        else if (isSpare(rollIndex)) {
            score += 10 + rolls[rollIndex + 2];
            rollIndex += 2;
        }
        else {  //No bonus
            score += rolls[rollIndex] + rolls[rollIndex + 1];
            rollIndex += 2;
        }
    }
    return score;
}
```

Green



```
Run All | Run... ▾ | Playlist : All Tests ▾

◢ No Traits (4)                                Test_One_Strike
  ✓ Test_All_Ones_Returns_20          < 1 ms     Source: BowlingGameTest.cs line 59
  ✓ Test_One_Spare                    < 1 ms     ✓ Test Passed - Test_One_Strike
  ✓ Test_One_Strike                   < 1 ms     Elapsed time: < 1 ms
  ✓ TestGutterGame_Sould_Return_Zero    4 ms
```

Time to put our refactoring hat….

Pending Design Concerns

- Encapsulate generation of strikes in test
- Method to if roll is a strike
- Strike and spare calculation

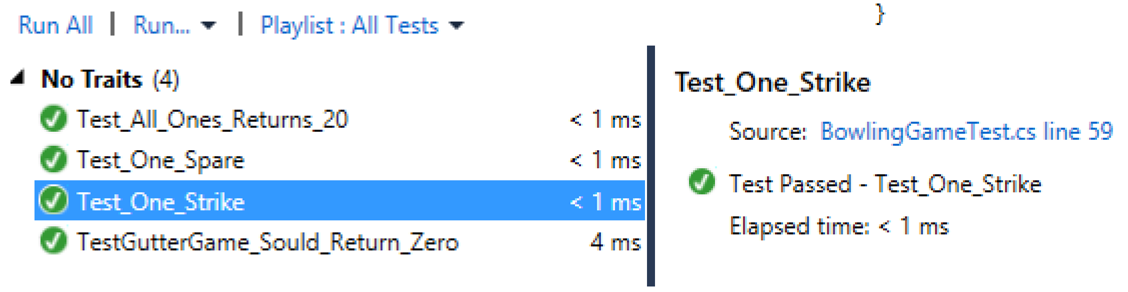
The 4th test: refactoring

Test

```
[TestMethod]
✓ | 0 references
public void Test_One_Strike()
{
    //Act
    rollStrike();
    sut.Roll(4);
    sut.Roll(5);
    rollMultiple(16, 0);

    //Assert
    //10 points for the strike, plus 9 points bonus,
    //plus 9 points from the additional rolls
    Assert.AreEqual(28, sut.Score());
}

1 reference | 1/1 passing
public void rollStrike()
{
    sut.Roll(10);
}
```

Code

```
5 references | 5/5 passing
public object Score() {
    int score = 0;
    int rollIndex = 0;

    for (int frames = 0; frames < 10; frames++)
    {
        if (isStrike(rollIndex)) { //Is Strike
            score += 10 + strikeBonus(rollIndex);
            rollIndex++;
        }
        else if (isSpare(rollIndex)) {
            score += 10 + spareBonus(rollIndex);
            rollIndex += 2;
        }
        else {  //No bonus
            score += rolls[rollIndex] + rolls[rollIndex + 1];
            rollIndex += 2;
        }
    }
    return score;
}

1 reference
private int spareBonus(int rollIndex)
{
    return rolls[rollIndex + 2];
}
```



```
◢ No Traits (4)
  ✓ Test_All_Ones_Returns_20            < 1 ms
  ✓ Test_One_Spare                      < 1 ms
  ✓ Test_One_Strike                     < 1 ms
  ✓ TestGutterGame_Sould_Return_Zero      4 ms

Test_One_Strike
  Source: BowlingGameTest.cs line 59
  ✓ Test Passed - Test_One_Strike
  Elapsed time: < 1 ms
```

Refactor

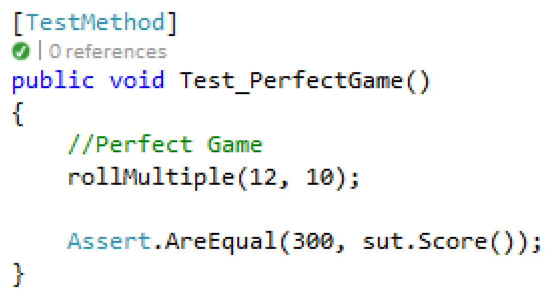
Time to put our refactoring hat….

Pending Design Concerns

- ~~Encapsulate generation of strikes in test~~
- ~~Method to if roll is a strike~~
- ~~Strike and spare calculation~~

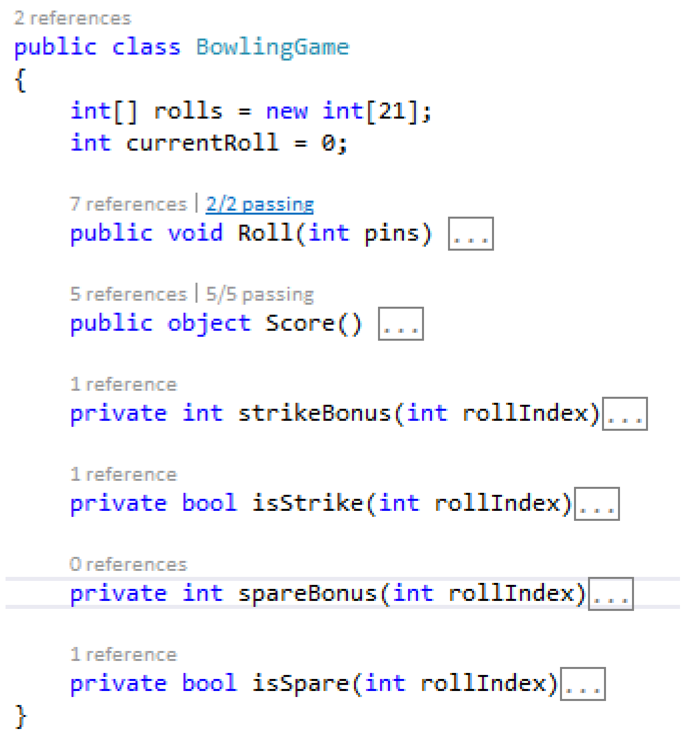
The 5th test: perfect game

Test

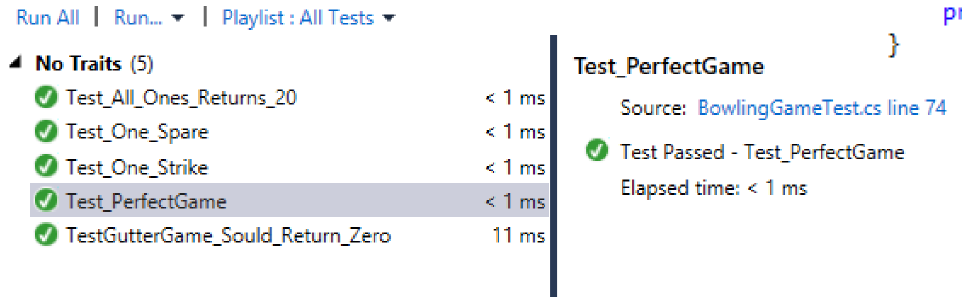


```
[TestMethod]
✓ | 0 references
public void Test_PerfectGame()
{
    //Perfect Game
    rollMultiple(12, 10);

    Assert.AreEqual(300, sut.Score());
}
```

Code



```
2 references
public class BowlingGame
{
    int[] rolls = new int[21];
    int currentRoll = 0;

    7 references | 2/2 passing
    public void Roll(int pins) ...

    5 references | 5/5 passing
    public object Score() ...

    1 reference
    private int strikeBonus(int rollIndex)...

    1 reference
    private bool isStrike(int rollIndex)...

    0 references
    private int spareBonus(int rollIndex)...

    1 reference
    private bool isSpare(int rollIndex)...
}
```



```
Run All | Run... ▾ | Playlist : All Tests ▾

◢ No Traits (5)
  ✓ Test_All_Ones_Returns_20          < 1 ms
  ✓ Test_One_Spare                    < 1 ms
  ✓ Test_One_Strike                   < 1 ms
  ✓ Test_PerfectGame                  < 1 ms
  ✓ TestGutterGame_Sould_Return_Zero   11 ms

Test_PerfectGame
  Source: BowlingGameTest.cs line 74
  ✓ Test Passed - Test_PerfectGame
  Elapsed time: < 1 ms
```

Green

Outline

- First test: gutter game
- Second test: score with no bonus
  - Refactor tests (DRY violation)
- Third test: spare bonus
  - Refactor SUT design
  - Refactor for DAMP
- Fourth test: strike bonus
- Fifth test: perfect game

Contact

Mauricio Aviles
Test Consultant
mavil@microsoft.com
www.microsoft.com/services

© 2013 Microsoft Corporation. All rights reserved. Microsoft, Windows, Windows Vista and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries. The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION

References

- Logos:
  - Bowling by Juan Pablo Bravo from The Noun Project