BUG03 Debug Log

# Bug:

**Bug 3:** Odds in the game do not appear to be correct.  
Crown and Anchor games have an approximate 8% bias to the house. So the win : (win+lose) ratio  
should approximately equal 0.42. This does not appear to be the case.

# Assumptions:

* None

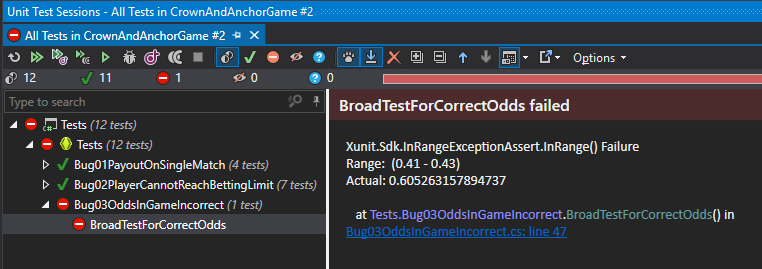
# Produce UAT Tests for each Bug

Done

# Changes to Base Code

A slight change to the ReadLine in the Program.cs so it isn’t inside the Play100 for testing purposes.

# Produce a broad Unit Test for each bug

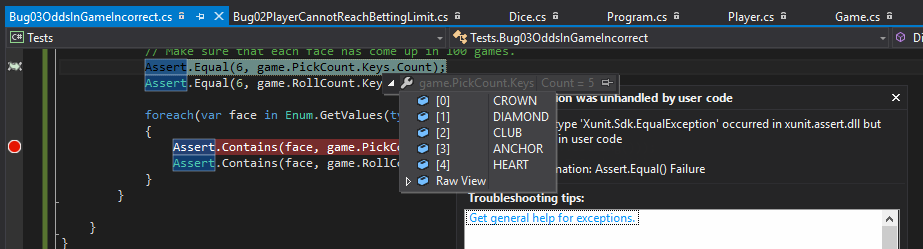


Unit test created – runs through 100 games and keeps track of the ratio, checks if it is within a range.

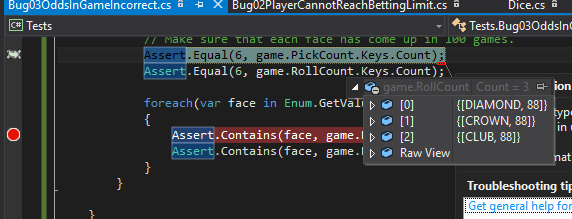
# See if there are any obvious additional Tests that I can introduce to cover side-effects

We already noted earlier in the discoveries of Bug 1 that there was a lack of randomization at play, so we will test that there is a solid randomizing in the game.

Testing results in the following obvious issues on the test created:



Consistently, the SPADE face isn’t showing on the player’s picks.



The RollCount keys are only showing the same three faces rolled every time, and therefore need to be randomized.

# Finding the cause

## Run look at where the Unit Tests fail and trace that line of code and check object states at those times

## Manually step through the code, the stacktrace, and the object windows to see what is being set and where the bug is occurring

First we’ll look at the Pick randomization – we’re missing SPADE consistently when running the test.

We’re using the Play100Games, but it doesn’t actively affect the dice rolling for the player, and instead passes a reference to PlayGame.

while (player.balanceExceedsLimitBy(bet) && player.Balance < 200)

{

try

{

PlayRound(bet, game, player, pick, ref winCount, ref loseCount);

}

catch (ArgumentException e)

{

Console.WriteLine("{0}\n\n", e.Message);

}

pick = Dice.RandomValue;

PlayGame passes the pick down to the PlayRound method, but doesn’t accept anything back, and passes it as a value type and not a reference type, so it can’t be replaced there.

This leaves the one line in PlayGame: pick = Dice.RandomValue;

If we trace into the Dice.RandomValue field we see it is a getter that returns a new random face.

public static DiceValue RandomValue

{

get

{

return (DiceValue)VALUES.GetValue(RANDOM.Next(VALUES.Length-1));

}

}

The issue here is that it returns .Length – 1 which means that it won’t return all 6 values of the array.

I’ve added a test that runs the randomizer 100 times and tests if it returns all six faces:

[Fact]

public void DiceReturnsAllSixFaces()

{

var faces = new HashSet<DiceValue>();

for (int i = 0; i < 100; i++)

{

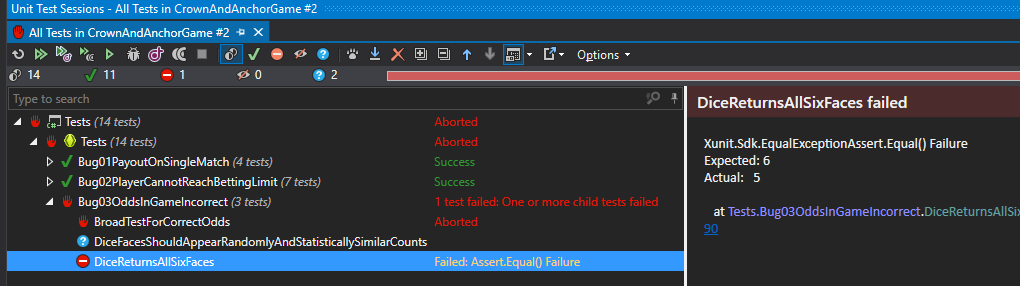
var face = Dice.RandomValue;

if (!faces.Contains(face)) faces.Add(face);

}

Assert.Equal(6, faces.Count);

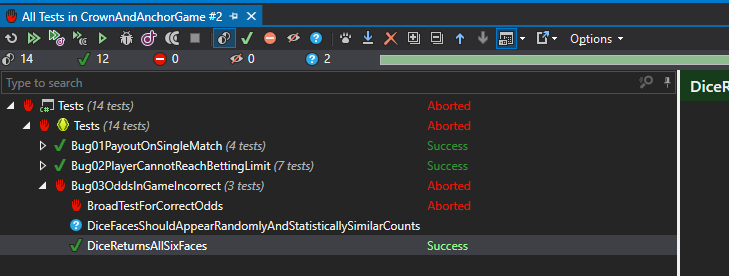
}



# Fix the Bug

Attacking first the fact the randomizer only picks between 5 faces, I am fixing that bug initially.

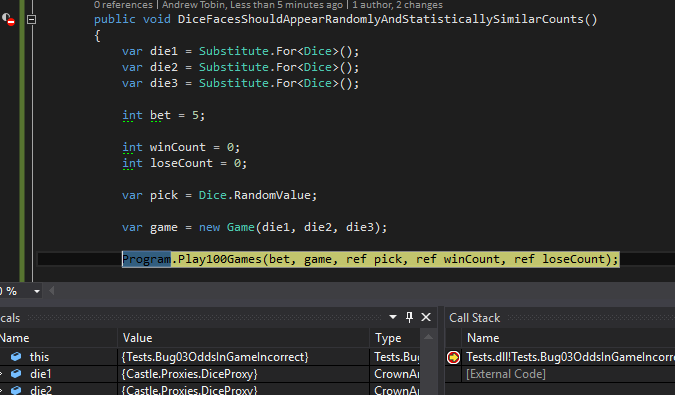
return (DiceValue)VALUES.GetValue(RANDOM.Next(VALUES.Length));



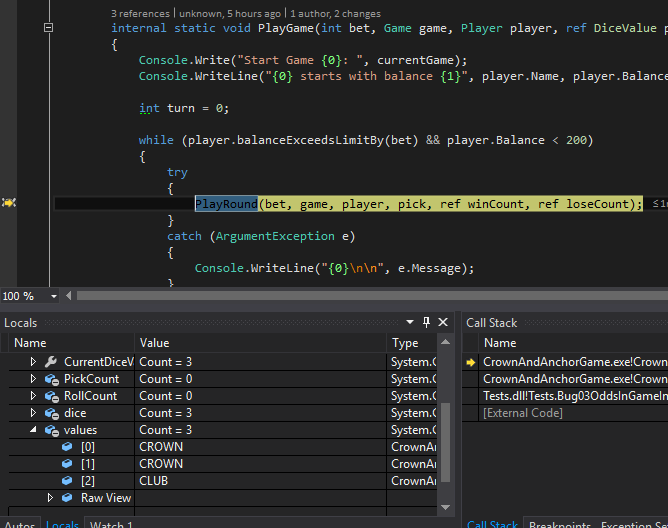
# Cause

Now if we look at why the dice for the game don’t randomize between each round:

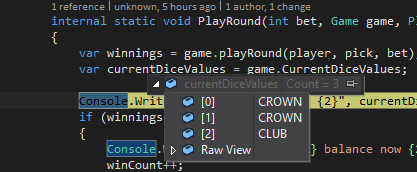
We’re going to debug into a breakpoint for the 100 games in our statistics unit test:



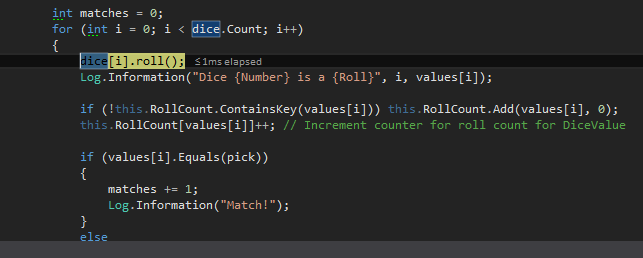
I’ve traced into the PlayGame method in the Program.cs and we are entering the round with the dice showing “CROWN”, “CROWN”, “CLUB”.



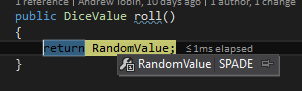
I can see inside PlayRound after the game has been played, and the dice should have been re-rolled, we exit the round with the dice still showing “CROWN”, “CROWN”, “CLUB”.



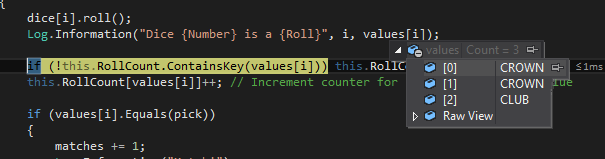
We can see in the code that for each die, it is rolled and then the value is retrieved from values[i]. We must therefore assume that dice[i].roll() should affect a change on that corresponding value;



We can see the dice[i].roll() method returns a new random die roll, but doesn’t set any values itself.



The dice[i].roll() method just returned a SPADE, but we can see after that roll that none of the “values” are set to a SPADE:



Therefore, we must set the associated value for the dice to what the new roll returned.

[Fact]

public void WhenDiceRolledValueShouldReflectNewRoll()

{

var die1 = new Dice();

var die2 = new Dice();

var die3 = new Dice();

int bet = 5;

int winCount = 0;

int loseCount = 0;

var pick = Dice.RandomValue;

var player = new Player("Test", 100);

var game = new Game(die1, die2, die3);

game.playRound(player, pick, bet);

var newDie1Value = die1.roll();

var newDie2Value = die2.roll();

var newDie3Value = die3.roll();

Assert.Equal(die1.CurrentValue, game.values[0]);

Assert.Equal(die2.CurrentValue, game.values[1]);

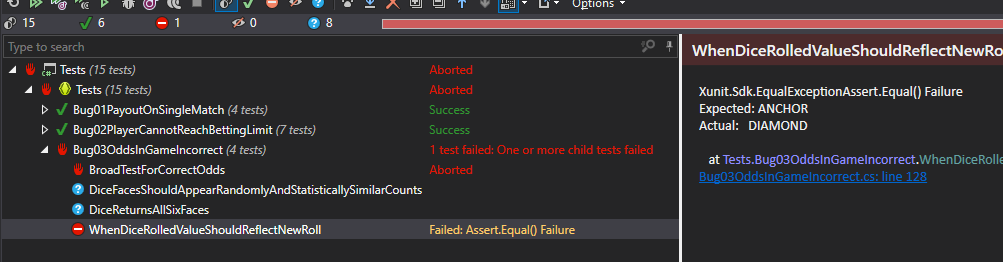
Assert.Equal(die3.CurrentValue, game.values[2]);

Assert.Equal(newDie1Value, die1.CurrentValue);

Assert.Equal(newDie2Value, die2.CurrentValue);

Assert.Equal(newDie3Value, die3.CurrentValue);

}



# Fix the Bug

We make sure the dice’s value is reset with the roll:

public DiceValue roll()

{

this.currentValue = RandomValue;

return CurrentValue;

}

And we make sure that the values for the rolls are retrieved from the dice, and not another source.

The current values list in Game.cs becomes:

public IList<DiceValue> CurrentDiceValues

{

get { return dice.Select(d => d.CurrentValue).ToList().AsReadOnly(); }

}

And instead of looking up in a list of values when the die is rolled, we take the return value and compare against that now:

var value = dice[i].roll();

Log.Information("Dice {Number} is a {Roll}", i, value);

if (!this.RollCount.ContainsKey(value)) this.RollCount.Add(value, 0);

this.RollCount[value]++; // Increment counter for roll count for DiceValue

if (value.Equals(pick))

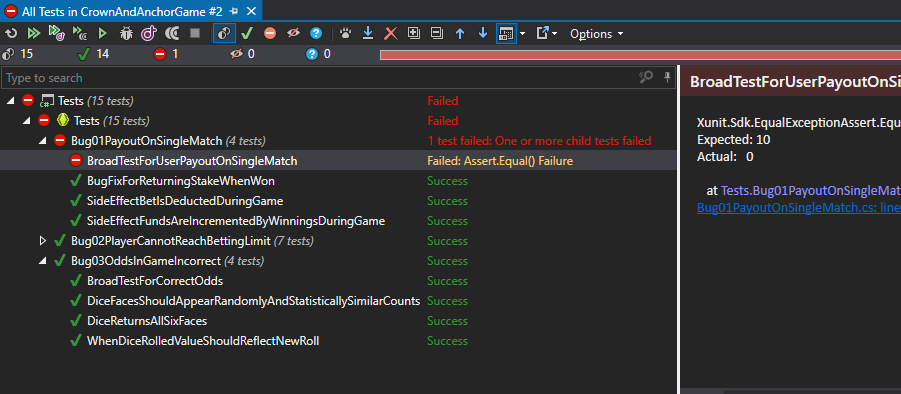
{

matches += 1;

Log.Information("Match!");

}

The tests for this section now pass, but I have a failing test from a previous bug:



# Test for Side-Effects

It has broken my test for BroadTestForUserPayoutOnSingleMatch() as that relied on being able to set the dice faces, which are now being randomly determined as they should be.

What I have done is instead created an IDice interface, so I can mock the return members.

public interface IDice

{

DiceValue CurrentValue { get; }

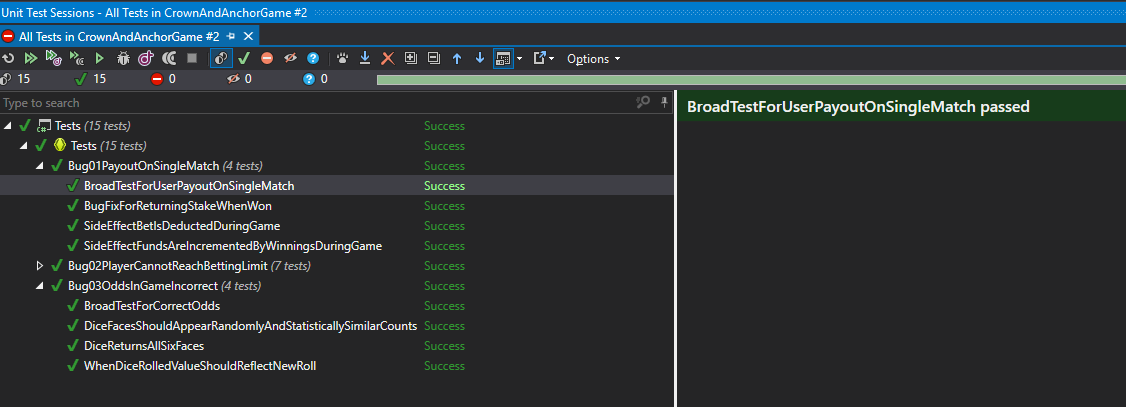
string CurrentValueRepr { get; }

DiceValue roll();

}

public Game(IDice die1, IDice die2, IDice die3)

All tests are now passing:



# Additional Discoveries

None

# UAT Run

The run has passed, but I have noted that the UI at present doesn’t allow for a single run, and even if it did, it would be difficult to test a random game of chance in a single run-through of the UAT as recorded.

I have chosen an appropriate part of the log to highlight the successful resolution of the bug.