BUG01 Debug Log

# Bug:

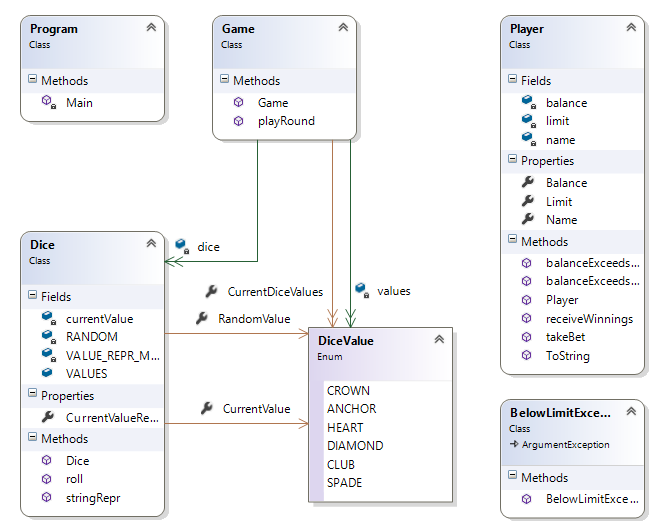
**Bug 1:** Game does not pay out at correct level.  
When player wins on 1 match, balance does not increase.

# Assumptions:

* I am assuming since we are expecting balance to increase that the payout is expected to be 1 to 1, plus the initial stake (bet) and not player receives their bet back.

# Familiarize myself with the code.

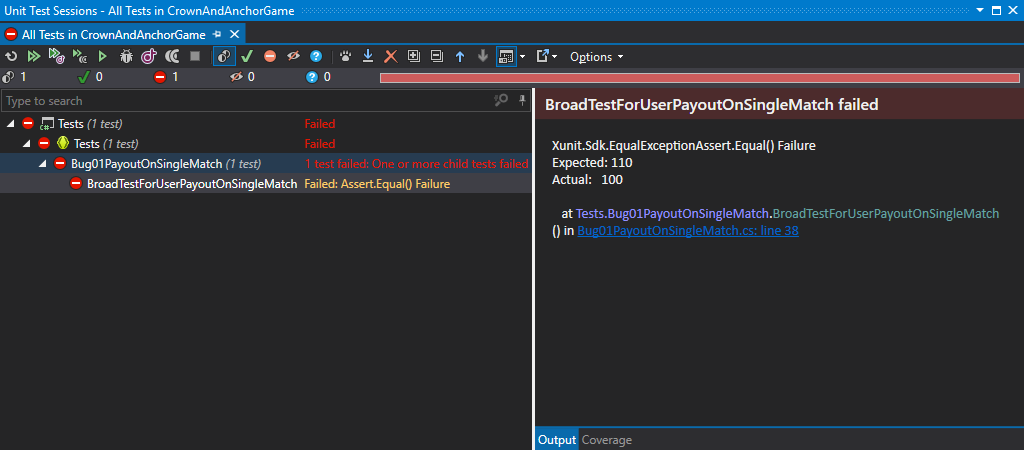
The first step is to do a quick review of the objects, and the structure of the program, see what relates to what and where. I will produce a class diagram so I can visualize what is going on.



# Produce UAT Tests for each Bug

Done

# Produce a broad Unit Test for each bug



Test has been created and is failing because the winning returned is zero and not the bet amount.

Checking in.

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

As well as returning correct winnings, I am checking the gambler’s winnings are incremented correctly:

Assert.Equal(bet, winnings);

Assert.Equal(funds + bet, player.Balance);

I have also added tests to make sure the player methods to deduct the bet and increment the player’s funds by the winnings have been called:

[Fact]

public void SideEffectBetIsDeductedDuringGame()

{

var die1 = Substitute.For<Dice>();

var die2 = Substitute.For<Dice>();

var die3 = Substitute.For<Dice>();

DiceValue pick = DiceValue.ANCHOR;

int bet = 10;

int winnings = 0;

int funds = 100;

var player = Substitute.For<Player>("Test", funds);

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

winnings = game.playRound(player, pick, bet);

// Make sure at some point that the bet is deducted by the funds.

player.Received().takeBet(bet);

}

[Fact]

public void SideEffectFundsAreIncrementedByWinningsDuringGame()

{

var die1 = Substitute.For<Dice>();

var die2 = Substitute.For<Dice>();

var die3 = Substitute.For<Dice>();

DiceValue pick = DiceValue.ANCHOR;

int bet = 10;

int winnings = 0;

int funds = 100;

var player = Substitute.For<Player>("Test", funds);

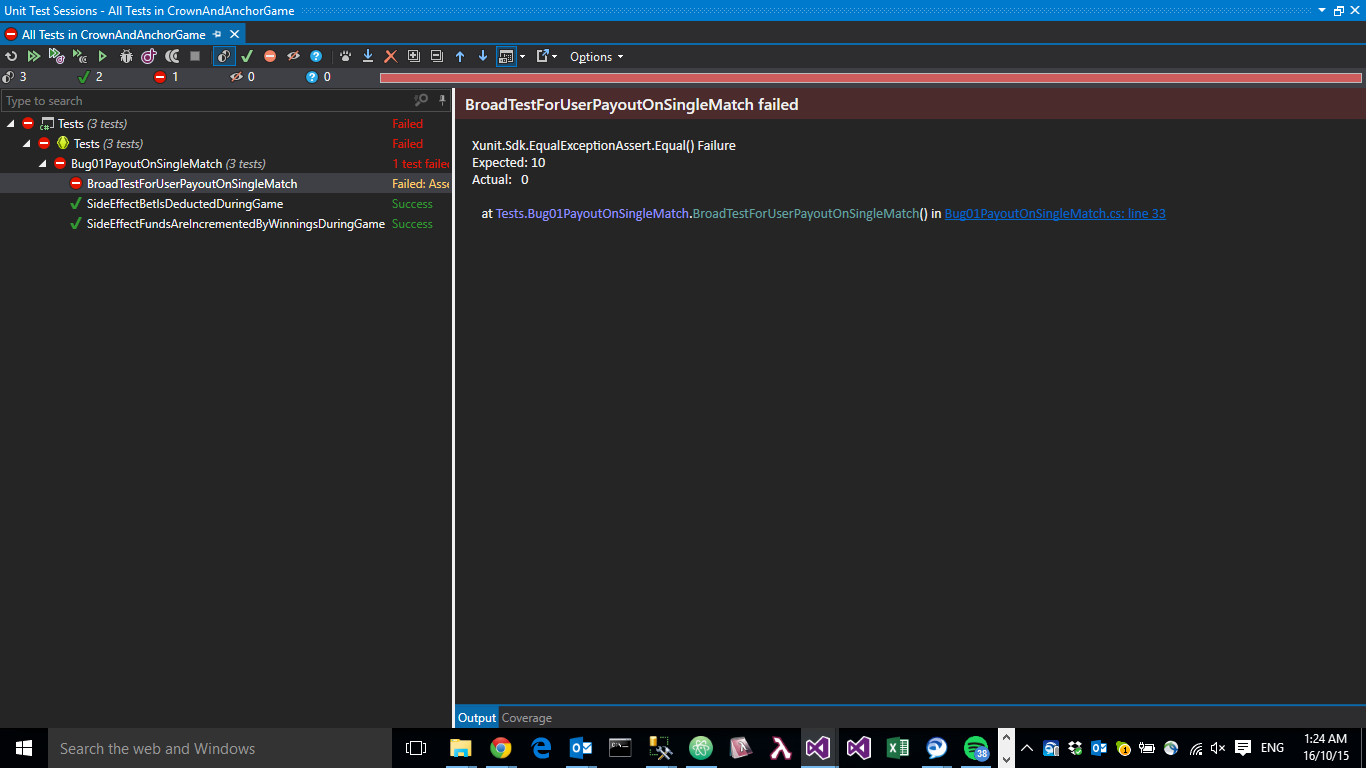
var game = new Game(die1, die2, die3);

winnings = game.playRound(player, pick, bet);

// Make sure at some point that the winnings are added to the funds during the game.

player.Received().receiveWinnings(winnings);

}



# Introduce logging to console and file around the state of object in play

Added a new instance of the logger to the program startup and instantiated it.

// Init a new instance of logger for logging to text file and console.

Log.Logger = new LoggerConfiguration()

.WriteTo.ColoredConsole()

.WriteTo.RollingFile(@"C:\Log-{Date}.txt")

.CreateLogger();

Log.Information($"----------------------------------------------\nStarting new instance at {DateTime.Now.ToString()}\n----------------------------------------------\n\n");

I have added a try{}catch{} around the whole program as well as some basic informational logging.

I have also added specific logging to do with the picks, bets, rolls, and winnings for the issue.

public int playRound(Player player, DiceValue pick, int bet)

{

using (LogContext.PushProperties(new PropertyEnricher("Player", player, true)))

{

Log.Information("Player {Name} has bet {Bet} on {Pick}\tBalance: {Balance}", player.Name, bet, pick, player.Balance);

}

if (player == null) throw new ArgumentException("Player cannot be null");

if (player == null) throw new ArgumentException("Pick cannot be null");

if (bet < 0) throw new ArgumentException("Bet cannot be negative");

Log.Information("Deducting bet");

player.takeBet(bet);

Log.Information("Balance: {Balance}", player.Balance);

int matches = 0;

for (int i = 0; i < dice.Count; i++)

{

dice[i].roll();

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

if (values[i].Equals(pick))

{

matches += 1;

Log.Information("Match!");

}

else

{

Log.Information("Not a Match!");

}

}

int winnings = matches \* bet;

if (matches > 0)

{

player.receiveWinnings(winnings);

}

Log.Information("Winnings are {Winnings}", winnings);

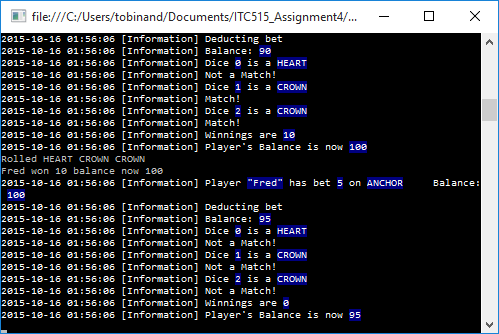
return winnings;

}

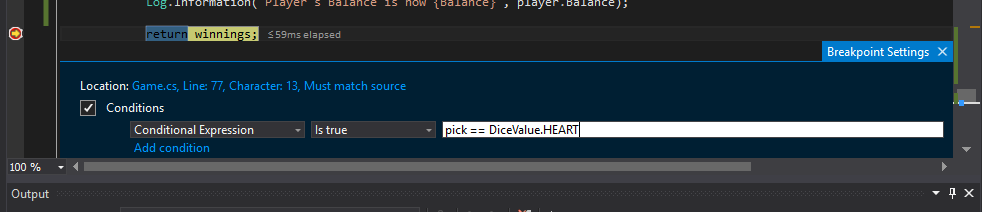
# Run look at where the Unit Tests fail and trace that line of code and check object states at those times

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

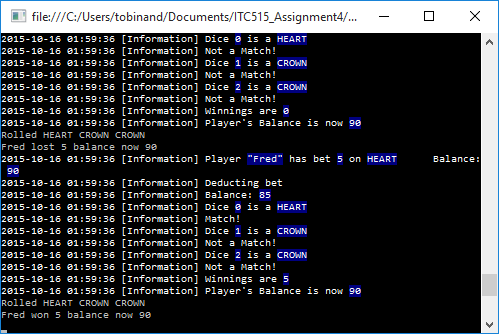
I have run a manual check of the program and discovered the dice are no randomizing each round, so I have added some stacktracing:



On this round it is always rolling a Heart, Crown, Crown, so I am setting a conditional breakpoint on the “return winnings;” line to check if the “pick” was a Heart, so I can see the logging output when the heart has been chosen.



We can see from the output of the logging and gameplay, when the breakpoint runs that it is increasing the player’s balance by the bet amount, but it isn’t also returning the initial stake that the player bet.



Specifically we can see in that last run that as we expected from the conditional breakpoint that “Fred” bet on Heart, and one appeared, and the payout was the same as his bet, and his balance was brought back to what it was before making the bet – and not increased by winnings plus the initial stake.

We can also see logically from stepping through the “playRound” method and the logging, that there isn’t a bug in the code written, but in the logic – that returning the stake is missing.

My Hypothesis, is that if we assume that the bug is correct and the balance should increase, and the stake should be returned – then the program’s bug is that the stake isn’t being returned.

# Write a Unit Test to fix the bug

I have created a unit test to make sure a method is called to return the bet.

[Fact]

public void BugFixForReturningStakeWhenWon()

{

var die1 = Substitute.For<Dice>();

var die2 = Substitute.For<Dice>();

var die3 = Substitute.For<Dice>();

die1.currentValue = DiceValue.ANCHOR;

die2.currentValue = DiceValue.HEART;

die3.currentValue = DiceValue.HEART;

DiceValue pick = DiceValue.ANCHOR;

int bet = 10;

int winnings = 0;

int funds = 100;

var player = Substitute.For<Player>("Test", funds);

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

winnings = game.playRound(player, pick, bet);

player.Received().returnBet(Arg.Is(bet));

Assert.Equal(bet, winnings);

Assert.Equal(funds + bet, player.Balance);

}

# Fix the Bug

In the Game.cs

if (matches > 0)

{

player.receiveWinnings(winnings);

player.returnBet(bet);

}

In the Player.cs

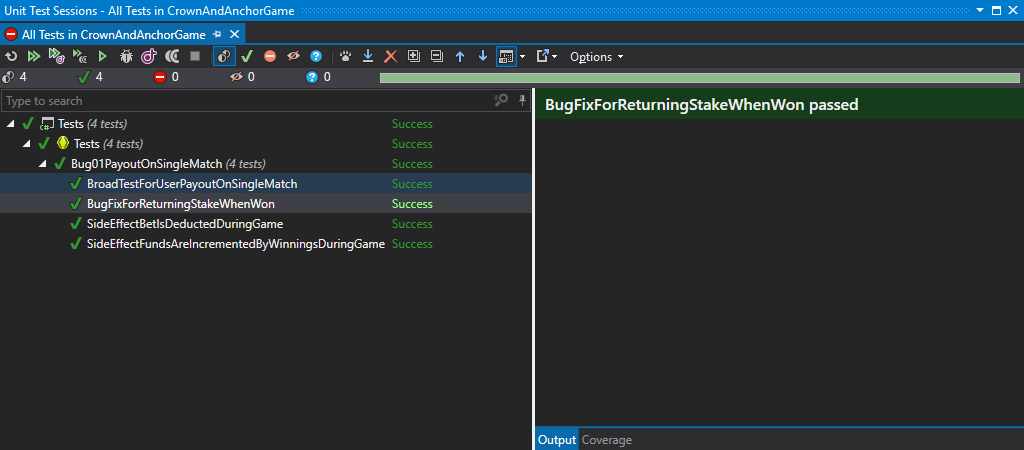
public void returnBet(int bet)

{

balance = balance + bet;

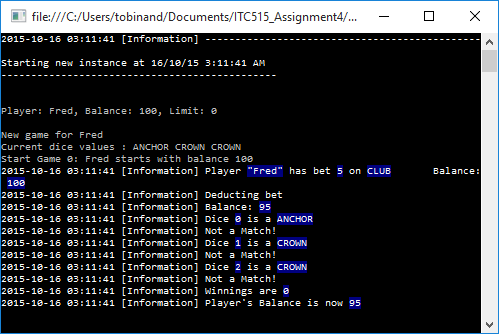
}

# Test for Side-Effects

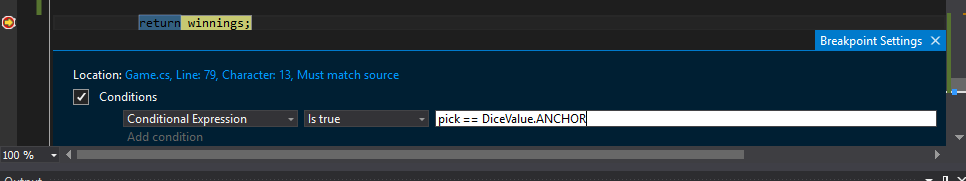


All tests are now passing.

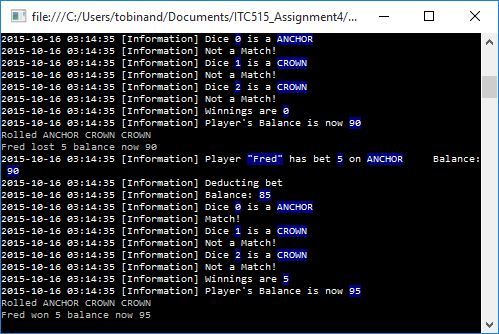
If we now run the program with our logging we find out what the dice are being set to:



As the dice are always returning ANCHOR, CROWN, CROWN now we will add back the conditional breakpoint with it set to watch for an ANCHOR bet.



Once we hit that breakpoint we can see a result:



We see that Fred started the latest round with 90, won on a single match and his balance increased to 95 – which means he received his winnings, plus received his initial stake back.

# Additional Discoveries

We need to log a bug for the fact that the dice never randomize between rounds, so it isn’t working properly. This may be fixed by one of the previously prioritized bugs.

# 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.

I have also recorded to fix the bug in the additional discovery on the UAT.