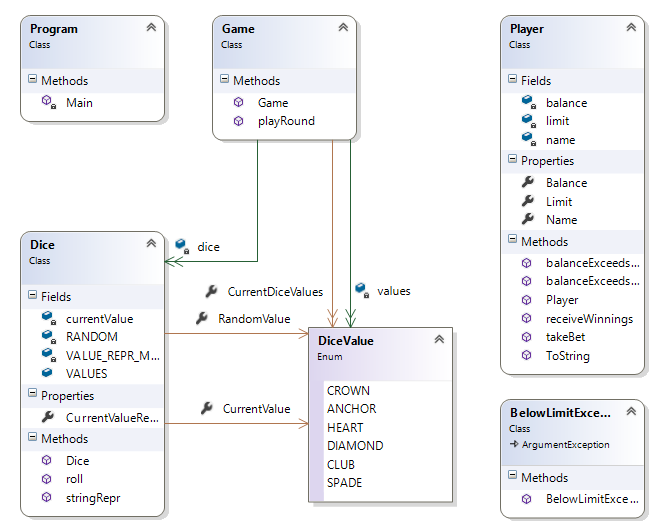
Planned Approach for Assignment 4

# Familiarize myself with the code.

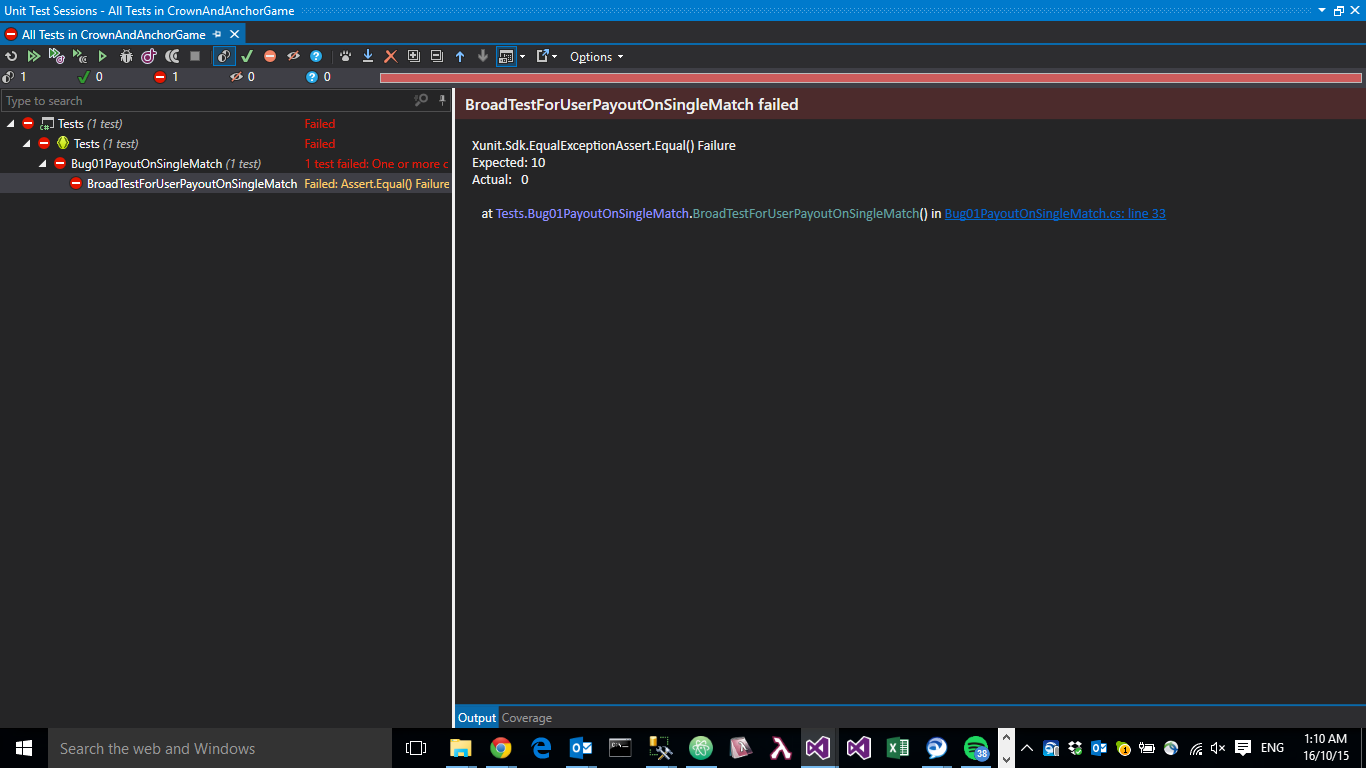
The first step is to do a quick review of the objects, and the structure of the program, see what relates to what and where. I will produce a class diagram so I can visualize what is going on.



# 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);

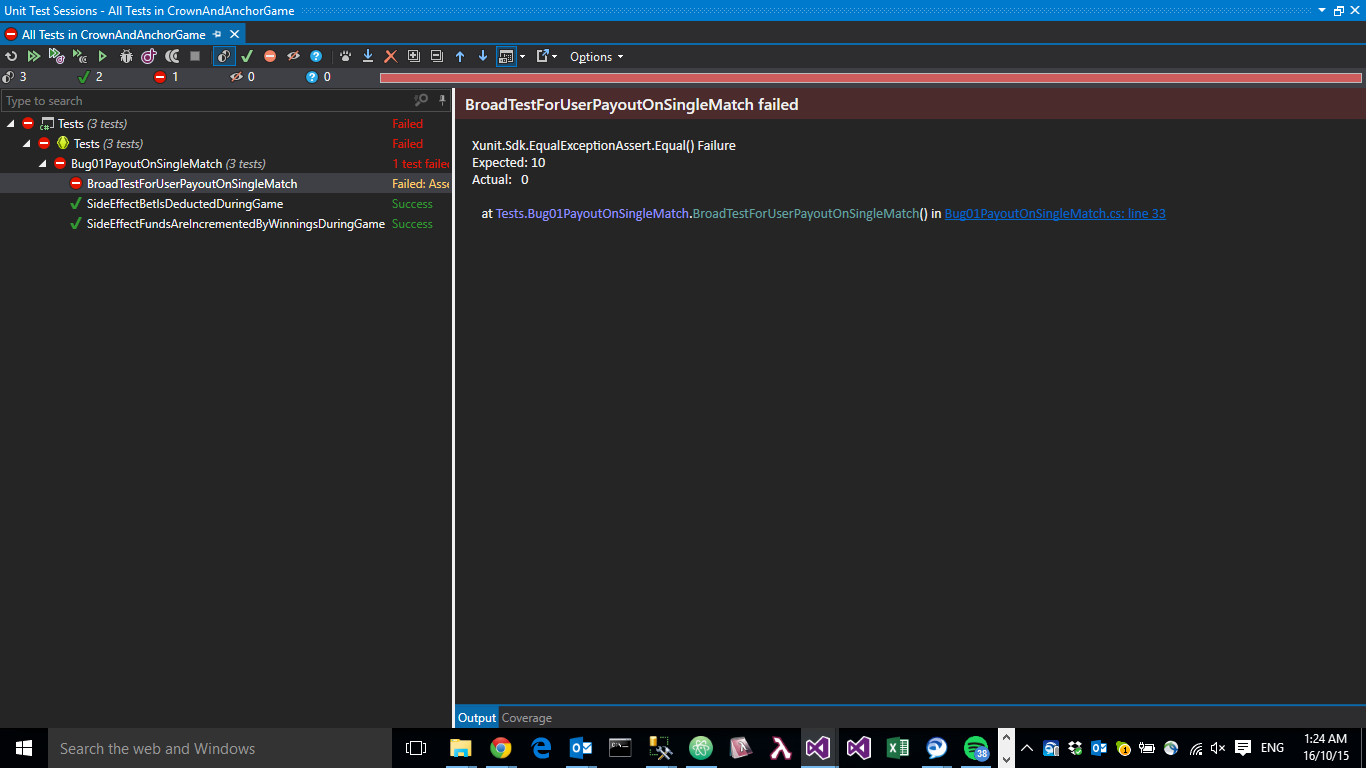
var game = new Game(die1, die2, die3);

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

# Write a Unit Test to fix the bug

# Fix the Bug

# Test for Side-Effects