BUG02 Debug Log

# Bug:

**Bug 2:** Player cannot reach betting limit:  
Limit set to 0, but game ends with player still with 5 (dollars) remaining.

# Assumptions:

* None

# Produce UAT Tests for each Bug

Done

# Changes to Base Code

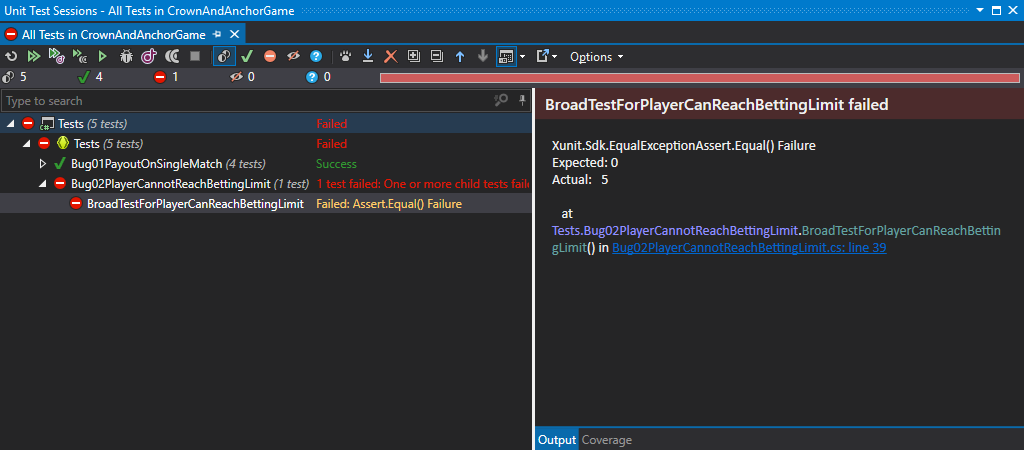
I have created a substantial refactor to Program.cs on commit: https://github.com/andrewtobin/ITC515\_Assignment4/commit/b1a4d6cbe22dcec031d49056664359db0f8b917d

This is to separate the structure of the 100 game play, the single game play, and a single round. This will allow us to tunnel in and test at each level where we need to.

I have also renamed some parameters for the methods, and local variable names for clarity.

I have also made one more change to supply the player from the Play100Games to the PlayGame method, so we inject the player into the game and we can get his final balance for testing.

# Produce a broad Unit Test for each bug



Test has been created and we can see that it’s failing because we expect to end the game with $0 balance, but the balance ends with $5.

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

None are coming to me at present – I may need to add some as I dig into code further.

# Finding the cause

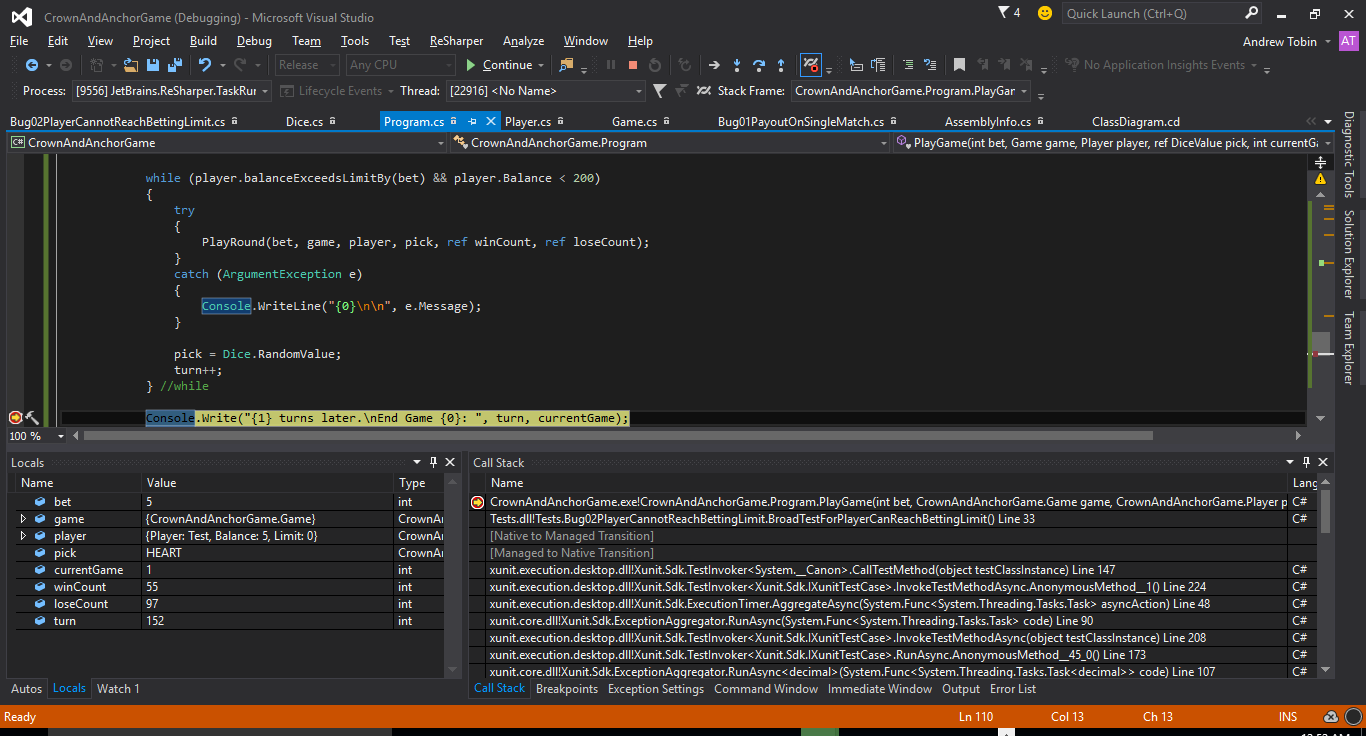
## Run look at where the Unit Tests fail and trace that line of code and check object states at those times

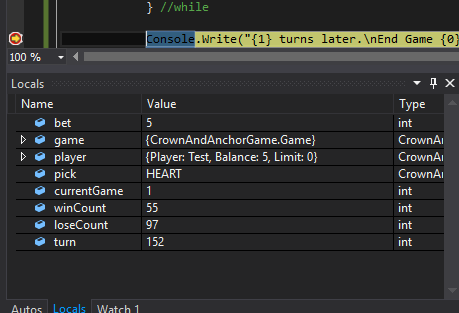
## Manually step through the code, the stacktrace, and the object windows to see what is being set and where the bug is occurring

I am using these two methods to figure it out. As we have refactored the code I know that it will be scoped inside the PlayGame method, so there, inside the objects passed in, or inside the PlayRound or other code this method calls downwards.

The first thing that I know is that it is something that affects or reads from the balance itself – because that is where we’re detecting the bug. We can therefore ignore specifically the dice rolling and win determination code, but we have to pay attention to how it affects the balance on win or loss, and any checks of that figure.

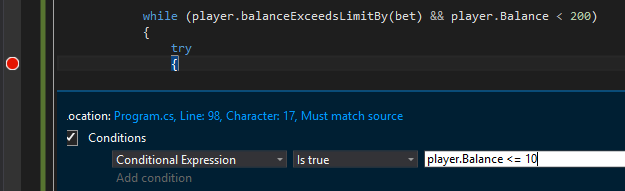
If I look at the Locals values at the end of the PlayGame method, while debugging the test I have created I can see it definitely exits with a balance of 5, beyond the test results confirming.



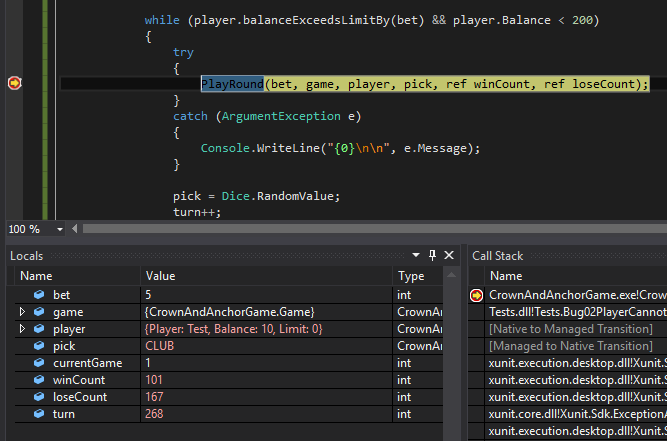


We can assume here that given we’ve proved that it is consistently exiting at 5, and the bug states this as the problem, that we can set a breakpoint at the While code to set and step-through the debugging when the player’s balance is approaching 5.

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

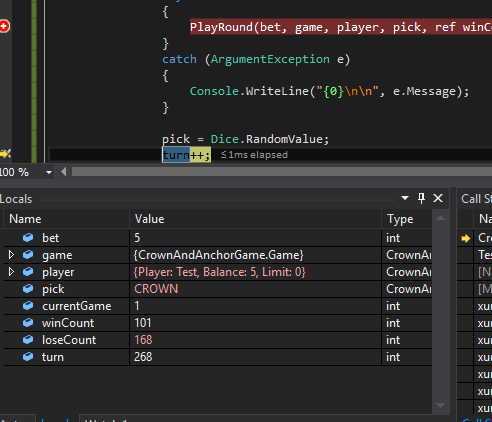


We can see it hits the breakpoint as expected when the player’s balance hits 10:

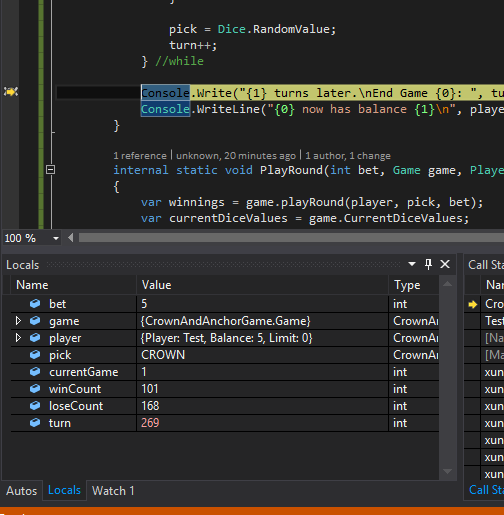


We then step over the PlayRound inside the while, continuing if we win and watching if we lose.

Immediately we lost, so it moved on to increment the turn variable.



And then it has escaped the While loop.



My hypothesis is: since the PlayGame seems to deduct the player’s balance as appropriate, and the effect here seems to be escaping the While loop prematurely, there is no “break” or “continue” statement in the “while”, so therefore it is a problem in the “While” condition.

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

The player’s balance is definitely lower than 200 in this situation, so therefore the problem exists inside the player.balanceExceedsLimitBy(bet) check.

public bool balanceExceedsLimitBy(int amount)

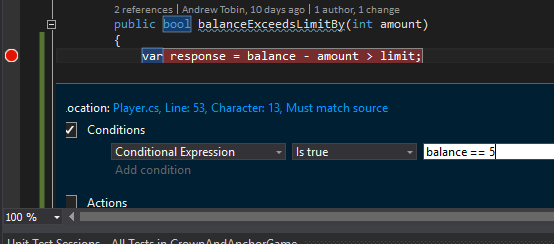
{

return (balance - amount > limit);

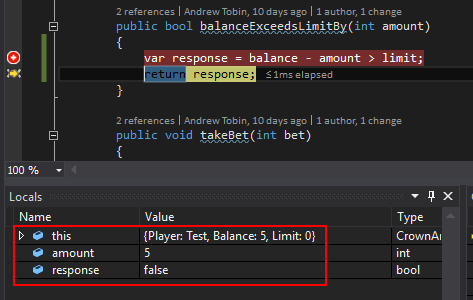
}

We can see that it is checking the Limit needs to be smaller than the player’s balance and the bet amount. In this case we are betting 5, the limit is 0, and the player’s balance is 5. If we apply this method’s logic then does (5 – 5 > 0)? No, that would return false, it would need to be >= to provide a “true” response.

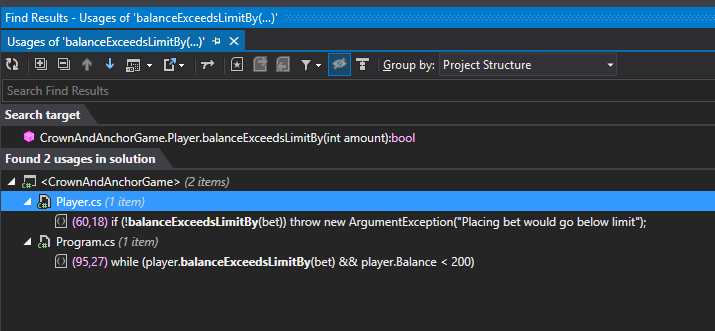
We’ll do a slight refactoring to assure ourselves this is the case and add a conditional breakpoint when the balance is 5.



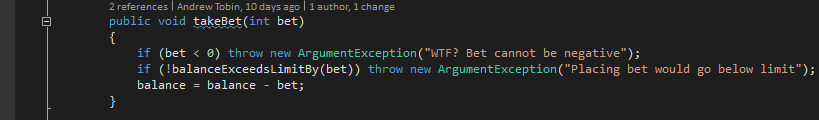
We can see when the balance is 5 it is definitely returning false.



If we look at where this method is called from we find two places, the Program, where we expect and the Player class.



The Player reference checks that there is enough funds in the balance to deduct the bet – we can safely change this code, but we will place some checks for side-effects.



[Fact]

public void TakesBetWhenBalanceExceedsBet()

{

var player = new Player("Test", 10) { Limit = 0 };

player.takeBet(5);

// Check bet was taken and error wasn't thrown.

Assert.Equal(5, player.Balance);

}

[Fact]

public void TakesBetWhenBalanceEqualsBet()

{

var player = new Player("Test", 5) { Limit = 0 };

player.takeBet(5);

// Check bet was taken and error wasn't thrown.

Assert.Equal(0, player.Balance);

}

[Fact]

public void TakeBetThrowsExceptionWhenBetExceedsBalance()

{

var player = new Player("Test", 0) { Limit = 0 };

Assert.Throws<ArgumentException>(() => player.takeBet(5));

}

# Write a Unit Test to fix the bug

I have written a set of tests to check this behavior, wrapping possible returns including the bug effect we are seeing when they’re equal.

[Fact]

public void BalanceExceedsLimitByReturnsFalseWhenBalanceDoesNotExceedBetLimit()

{

var player = new Player("Test", 10) { Limit = 0 };

var response = player.balanceExceedsLimitBy(15);

Assert.False(response);

}

[Fact]

public void BalanceExceedsLimitByReturnsTrueWhenBalanceExceedsBetLimit()

{

var player = new Player("Test", 10) { Limit = 0 };

var response = player.balanceExceedsLimitBy(5);

Assert.True(response);

}

[Fact]

public void BalanceExceedsLimitByReturnsFalseOnEqual()

{

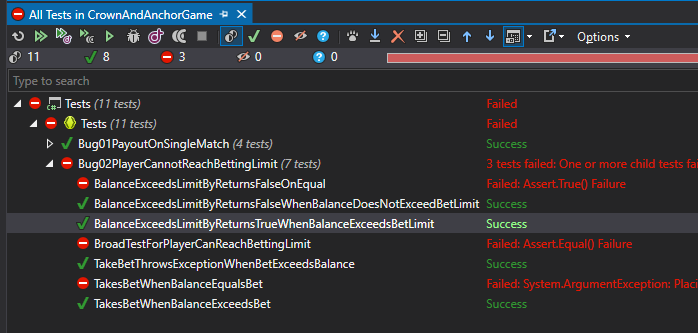
var player = new Player("Test", 5);

var response = player.balanceExceedsLimitBy(5);

Assert.True(response);

}

# Fix the Bug



We can see the tests we wrote for covering the bug and the side-effects are erroring as expected.

We can now fix the code as we’d expect:

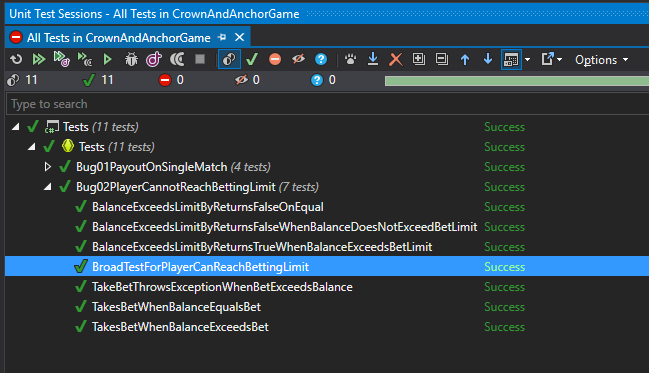
public bool balanceExceedsLimitBy(int amount)

{

return balance - amount >= limit;

}

All tests now pass successfully:



\*Note: The game will still exit prematurely if the balance exceeds $200. I am assuming that $200 is the equivalent of “Breaking the bank” for testing this game and so I am ignoring this condition.

# Test for Side-Effects

Side effects tests are passing as well.

# Additional Discoveries

No additional discoveries, apart from the aforementioned breaking the bank.

# UAT Run