Debugging Log

**Assignment 2**

ITC515 – Professional Programming Practice

Justin McKay - Student ID: 11530610

# Debugging Log

## BUG001 - Game does not pay out at correct level

### Steps taken to trace error

* Check *Player.receiveWinnings* method. Method appears to add winnings to balance correctly.
* Check *Game.playRound* method. Note: playRound method calls the takeBet method prior to testing the rolled dice against the value selected.
* This does not appear to be a bug. There could be confusion with the game only the winnings, so you don’t actually see the bet being taken out, and then the winnings put back onto the balance.
* The following 4 screenshots show the progression from taking the bet (balance before and after) and receiving the winnings (balance before and after)

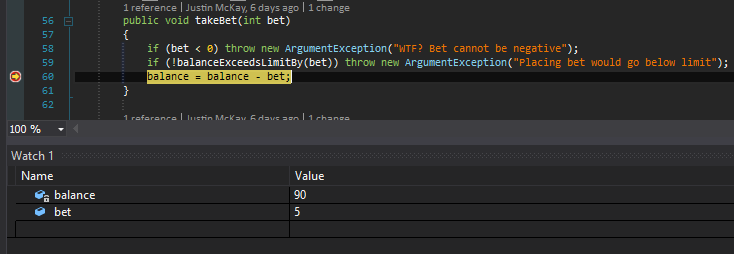


Figure 1 - Showing balance before the bet has been taken

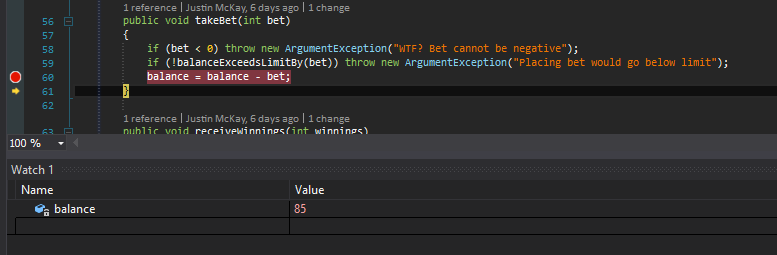


Figure 2 - Showing the balance after the bet has been taken

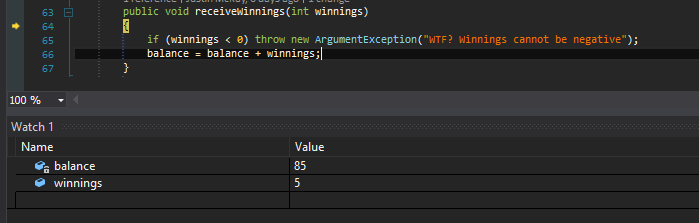


Figure 3 - Showing the balance before the winnings have been received

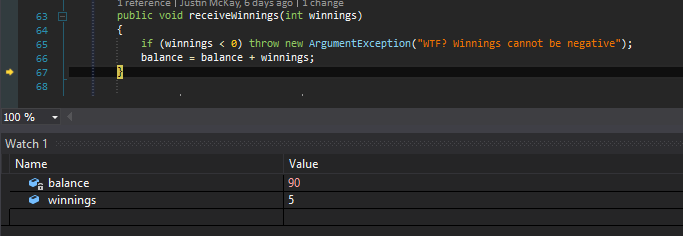


Figure 4 - Showing the balance after the winnings have been added back to the balance

* Will write a test that confirms the bet has been placed (bet deducted from balance) and when a single dice wins, shows the winnings added to the balance.

### Bug elimination steps

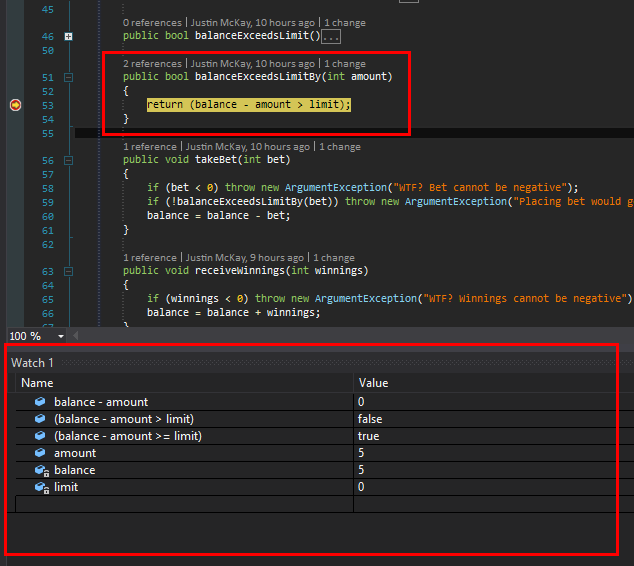
As this is not actually a bug, no steps are required for corrective action.

### Possible Improvements

Change the UI to reflect both the bet being taken off the balance and the winnings being added to the balance. This should correct the issue.

## BUG002 - Player cannot reach betting limit

### Steps taken to trace error

* Check *Player.takeBet* method. Noticed *Player.balanceExceedsLimitBy method* will prevent the bet from being taken if false is returned.
* Step into *Player.balanceExceedsLimitBy* method. Notices that the method will only return true (and allow the player to take a bet) if the balance – bet is greater than 0 (so that you can’t end up with a negative limit). However, as the limit is 0, this method should be checking for balance – bet is greater than *or equal to* the betting limit.
* This behaviour is confirmed with the following debugger screenshot:  
  

### Bug Elimination Steps

To fix the bug, the following code needs to be updated:

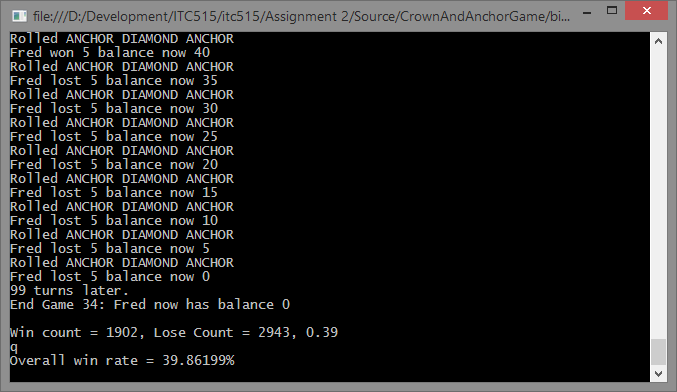
**Player.cs:**Change line 53 to be amount >= limit

### Possible Code Refactoring Improvements

Consider renaming method from *balanceExceedsLimit* to *balanceReachedLimit* so that it’s slightly clearer to the developer that the balance can reach (equal to) or exceed (greater than) the limit.

## BUG003 - Odds in game do not appear to be correct

### Steps taken to trace error

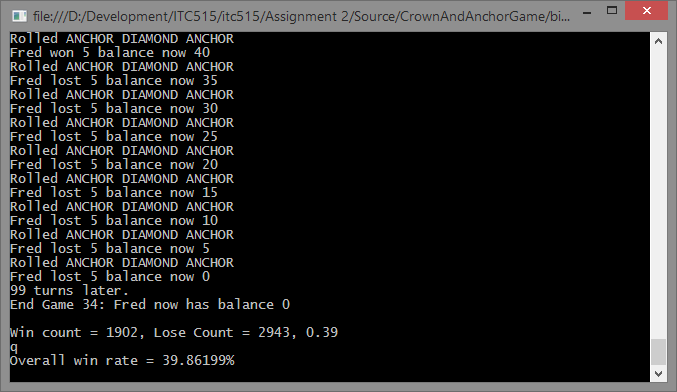
* Initially unable to reproduce this error, as running the program will show a valid score of result of 0.39 (which is much closer to the 0.42) ratio than the reported 0.60 values: See screenshot below:  
  
* Continuing on with the bug report none-the-less, concentrating on the Dice.roll() method as this seems to use a random number to generate the value of the dice rolled.
* Looking at the screenshot above, the last 9 or rolls are the same, so I don’t think the random number stuff is working correctly and incrementing the seed properly (perhaps using milliseconds which is much too slow for these types of iterations).
* This actually makes sense, as the game will roll 3 different dice objects, which will each have its own RANDOM property which is not getting updated, as evidenced by the repeating pattern. I’ll write a unit test for this.
* Created unit test *Dice\_Roll\_Will\_Always\_Return\_A\_High\_Repetition\_Rate* which will detect any repeat rolls where the number of repeats are > 5. This is not the cause the bug. Back to the drawing board.
* New bug found, see BUG004 - Game does not update current dice values rolled.
* Coming back to this bug now. I’m now able to get different values for the ratio. So far I’ve had 0.20, 0.60 & 0.49. Not sure if this is due to resolving the bug of the dice values not being updated (BUG004).
* It does not seem to matter if I close Visual Studio and debug the application or just re-debug the application. Nor can I find a method to reliably get these values to change.

**1/10/2014 5:23pm**

* After running a few iterations of the game from the command line, and outputting the response to a text file, I noticed that the SPADE is never rolled.
* Going to re-visit my dice rolling test and get actual percentages for rolls of dice over 3000 rolls to really check the random number part of the dice rolling. And that each dice value is rolled. Given that a chance of rolling a particular value is 1/6 (1.67), the test will check for a value within the 0.16 and 0.17 range.
* So, I’ve created the ‘Dice\_Roll\_Within\_Acceptable\_Probability\_Ratio’ test, and it appears that the SPADE value is never rolled, which is throwing out all of the ratios.
* This would explain why percentages of the game results are off, as we are no longer 1/6 for 3 dice, we are 1/5.
* Looking at the Dice.RandomValue method, I notice that that the max value is the length – 1, which make sense so as to include all DiceValue indexes without selecting the Length + 1 value, which would return an invalid DiceValue. The problem is that the Random.Next number will never return the maxValue, as this is an *Exclusive* value. To fix this bug, we need to allow the Length + 1 value (so remove the -1).
* Initially, this did not seem to fix the ratios being returned, so I bumped the iteration up to 100,000 and now each 6 dice values a within an acceptable range.

## BUG004: Game does not update current dice values rolled.

### Steps taken to trace error:

* While debugging for BUG03 – Odds in game do not appear to be correct, I noticed that the game is not updating the values of the dice outputted to the console when the game is iterating through the rounds. Screenshot of the bug is below:  
  
* To confirm this, I’ll write a unit test that shows that the each time a round is played, the dice values remain unchanged. 20 rounds without change should be sufficient for this test.
* Unit test confirms the existence of the bug.
* Updated the code to remove the bug by ensuring the values are correctly updated after each roll. I’ll now write a test to confirm that the bug has been resolved.
* Unit test to confirm the bug has been resolved has been written. Bug now confirmed as resolved.