# Bug 3 – Debugging Log

The bug is a macro bug that affects the probability of a winning game occurring. If you looked at a single round of a single game, nothing seems to be amiss. As such, for the automated test, I used a very large sample size so that the probability should even out.

While fixing other bugs, one possible cause for this bug became very obvious. The rolls done by the three game dice were identical every round. Depending on how many duplicates the three dice rolled, the percentage of winning games would change. Rolling three different values gave the player a high chance of winning a round, while rolling 3 identical values greatly tipped the balance in the houses favour. I have labelled this problem “Bug 3a”.

This was not the whole problem, however. If the three dice rolled three different possibilities, then logically, any roll should have a 50% chance of winning. However, this was not what was observed. There was, on average, 60% wins over losses in that situation. This behaviour indicates another bug, which I have labelled “Bug 3b”. But first I shall fix Bug 3A.

Bug 3a boils down to the program not assigning new DiceValues to the Dice each round. Therefore, we can shift focus from Main to Game.playRound().

In Game.playRound(), the program calls the roll() and getValue() methods on each of the dice. We shall narrow our focus to these methods as they are most likely to contain the bug.

In the Dice class, Netbeans makes a suggestion: The Value field can be made final. This means that the IDE cannot find an instance where that value is altered. This is not intended behaviour, and it clearly indicates the nature of the bug – the dice values are never changed after they are initially set.

The bug is in the roll() method. Instead of setting the value to a new, random value, it simply returns that value. The playRound() method doesn’t assign that value to any variable or use it at all. To demonstrate this, I added code to the roll() function to intercept and output the results of the roll to the console. Here is some of the output:

Start Game 0:

Fred starts with balance 100, limit 0

Turn 1: Fred bet 5 on ANCHOR

CrownAnchorHeartRolled CROWN, HEART, DIAMOND

Fred lost, balance now 95

Turn 2: Fred bet 5 on CROWN

ClubAnchorAnchorRolled CROWN, HEART, DIAMOND

Fred won 5, balance now 100

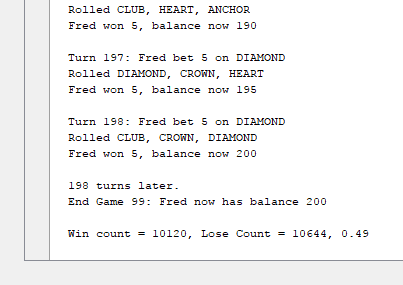
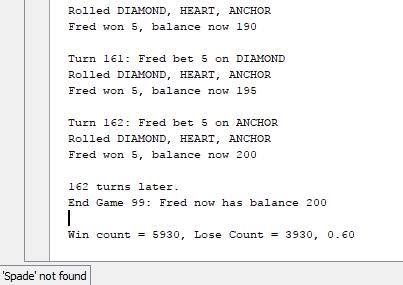
Turn 3: Fred bet 5 on ANCHOR

HeartHeartDiamondRolled CROWN, HEART, DIAMOND

Fred lost, balance now 95

As you can see, the game rolls new values correctly, but still uses the same values initially generated in the Dice() constructor.

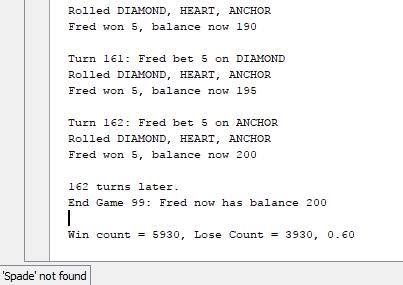
The fix made was to make the roll() method to set the new random value as the new value stored in the dice object, instead of just passing those values to nothing, we fix bug 3a. I also changed roll() to return void, as it had already functionally been doing. Before and after screenshots are below.

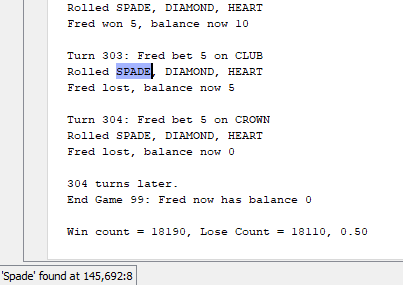


I decided to wait to actually make the fix for bug 3a, because the properties of the bug would help to demonstrate bug 3b. Because it messed with the overall results in the way it did, I suspected it had something to do with the random number generation itself. The DiceValue class uses java.util.Random to generate random values, by calling the nextInt(int n) method. This method, according to the Javadoc, “Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive)”.

The specified value is the ordinal of the final value in the enum – which is 5, since the ordinal begins at 0. This would mean that the final value in the enum would never be selected. To prove this, I searched the Output for “SPADE” and “Spade”, and got no results.

The fix made was simply to add 1 to the value passed to the Random.NextInt() method. Before and after screenshots are below. As you can see, the average win ratio is now the expected 0.50 for the earlier case of the results always returning the same 3 unique results.





Combining both fixes together, we can see that bug 3 is well and truly solved. I tightened the variance on the automated test to make it three times stricter, and it still passes the test 10 times with flying colours.

