# Project Repository

<https://github.com/agaba01/ITC515Assignment04>

# Bug Report

## Bug 1

Compiled the source code and executed the first test case to reproduce bug 1 which is documented in the UAT document located in the project repository/UAT folder.

### Replication

Output from the test demonstrating the buggy behavior is illustrated in the below figure 1 where we can see that even after consecutive wins of Fred, balance didn’t increase. Figure 1 shows that Fred won in turn 22,23, and 24 but balance didn’t increase.



Figure 1 - Bug-1

### Simplification

Developed a test class called “testGame” and first test method testBug1() to test the bug 1 where game is not paying at correct level.

*# javac -cp junit-4.12.jar;\* testGame.java*

Compiled the testGame class and executed the first test case using Junit runner classes at the command line using below command,

*#java -cp \*;. org.junit.runner.JUnitCore testGame*

*Figure 1 illustrates a simple UNIT TEST CASE to reproduce the bug.*

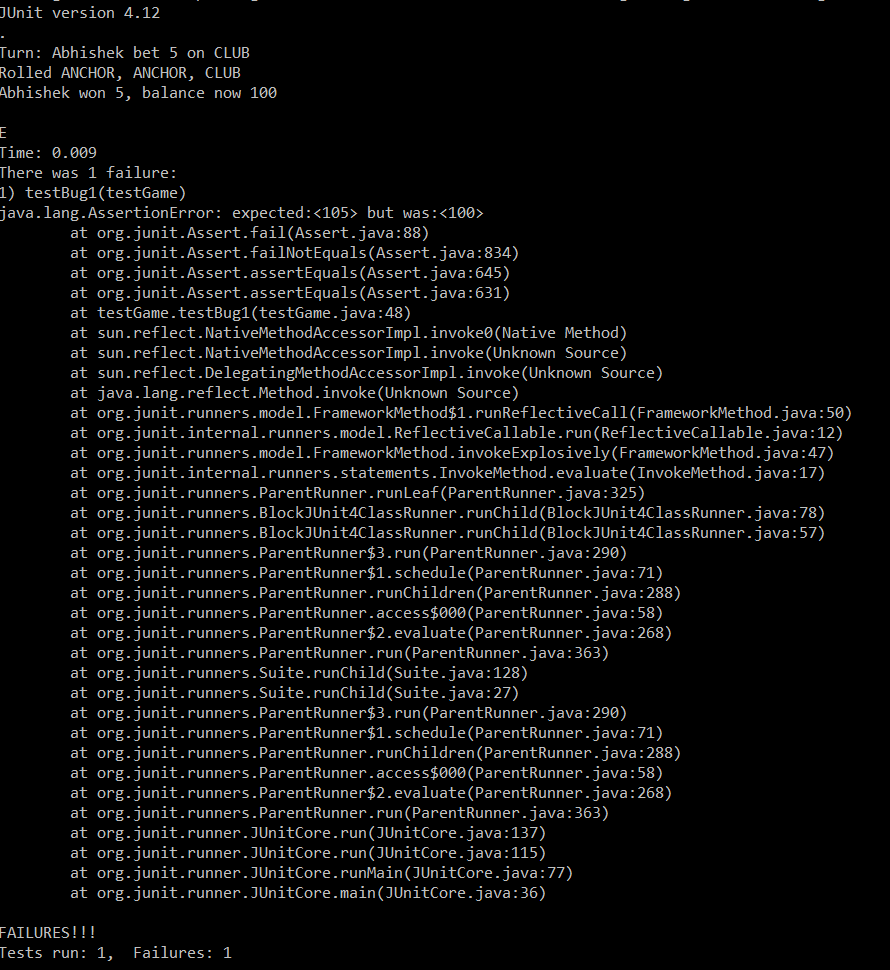


Figure - Bug 1 Unit Test

### Tracing

For bug 1, we will traceback the origin of the problem and for this we have identified that problem is with playRound() function of Player class. It is clear from the code snippet that takeBet() was called in normal flow so balance was being deducted by the bet(5) in all cases.

Figure 3 shows the area where playRound() function code has gone wrong.

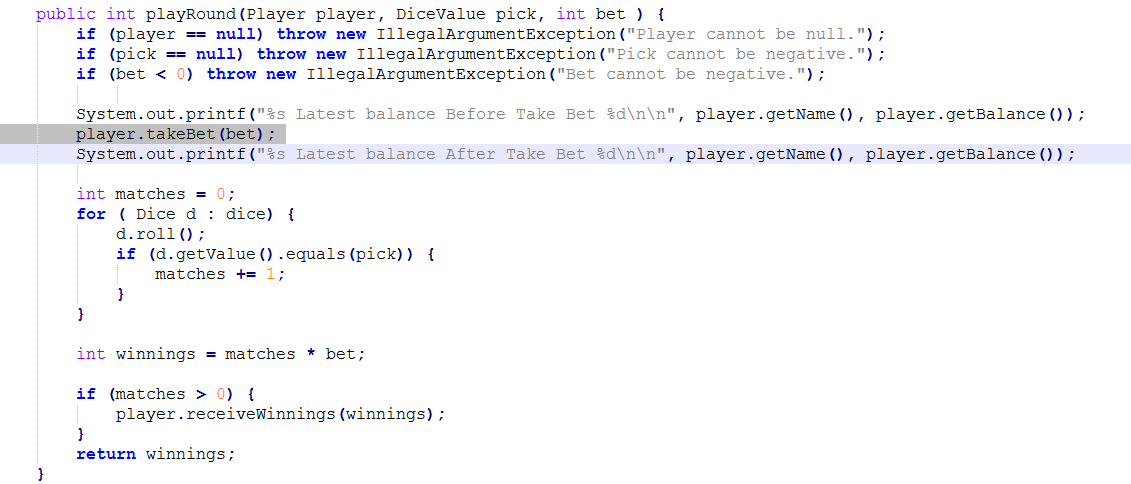


Figure - Error Tracing (a)

Figure 4 before and after screenshots while printing important variable values.

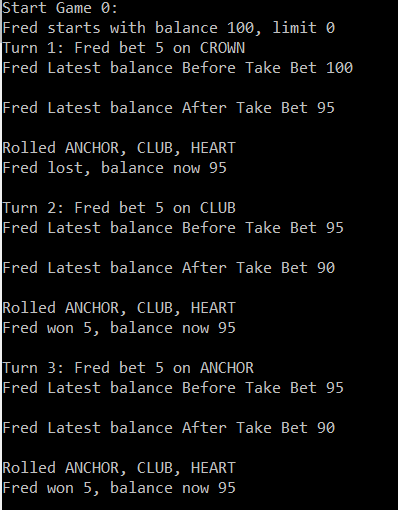


Figure - Error Tracing (b)

We can see in figure 4 that in turn 2 and 3, player won but balance didn’t increase as expected because 5 (bet) is being deduced in all cases.

### Resolution

To fix this bug we just need to put *player.takeBet(bet);* statement to if-else block as shown in figure 5.

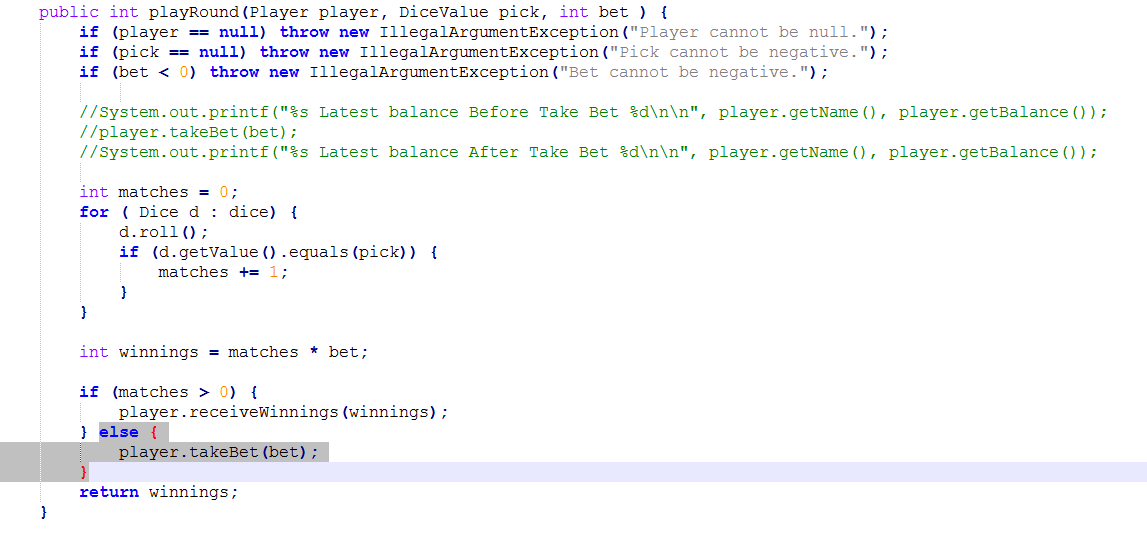


Figure 5 - Resolution (a)

Output of correct operation after resolution is shown in figure 6.

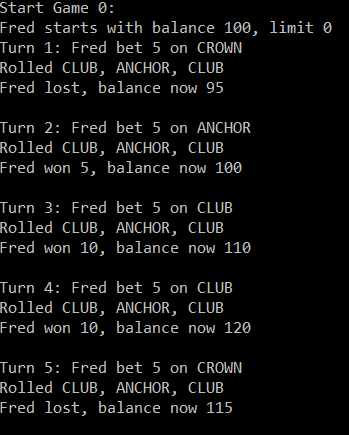


Figure 6 - Resolution (b)

We can see in figure 6 that in turn 2, 3, and 4, player won and his balance increased according to the winnings.

## Bug 2

Compiled the source code and executed the second test case to reproduce bug 2 which is also documented in the UAT document located in the project repository/UAT folder.

### Replication

Output from the test demonstrating the buggy behavior is illustrated in the below figure 2 where we can see that player (Fred) cannot reach the betting limit which is 0 and games ends with $5 remaining.



Figure 7 - Bug2

### Simplification

Added another method testBug2() to the same testGame Class to simplify automated test which reproduces the bug 2 i.e. games ends with $5 balance remaining.

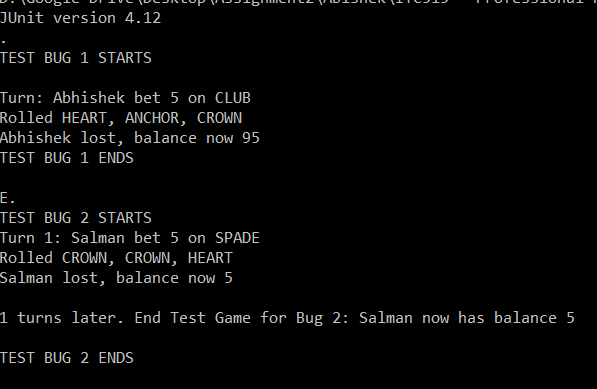


Figure 8 - UNIT TEST BUG 2

Figure 8 shows the output of Unit Test for Bug 2 where player Salman has balance of $5 and game ends.

### Tracing

While tracing back the root cause of Bug 2, we find problem with *player.balanceExceedsLimitBy(bet)* functional call in the While Loop of Main function. If we analyze this function, we see that it has a return statement which says *return (balance - amount > limit);* which means that we can never reach to the limit due the > operator being used.

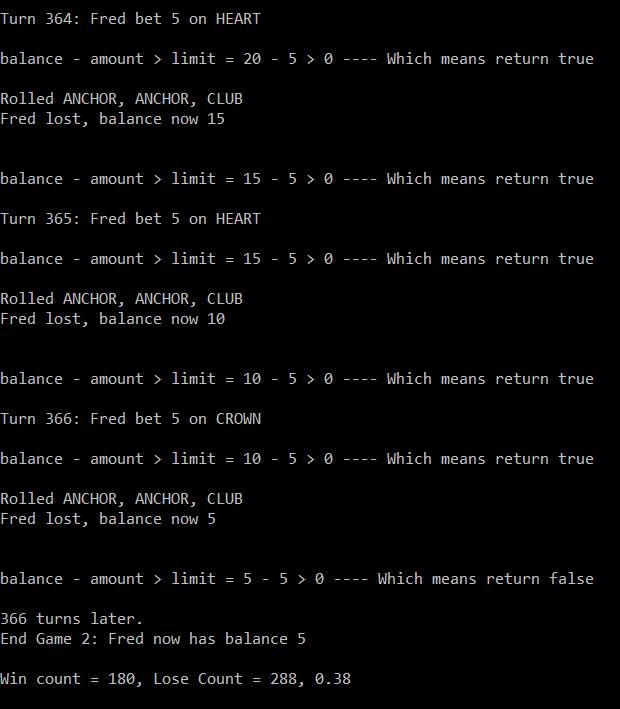


Figure - Tracing Bug 2

Figure 9 shows a dump log right before the limit is checked.

### Resolution

As identified in the previous section, if we simply change the statement from *return (balance - amount > limit)* to *return (balance - amount >= limit)*, it will fix the bug 2. Figure 10 shows the updated code of function *player.balanceExceedsLimitBy(bet).*

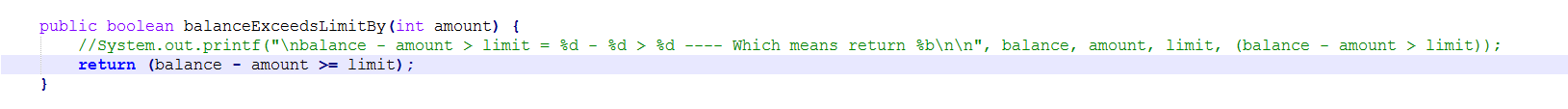


Figure 10 - Resolution of Bug 2 (a)

Figure 11 illustrates the correct operation after resolution of Bug 2.

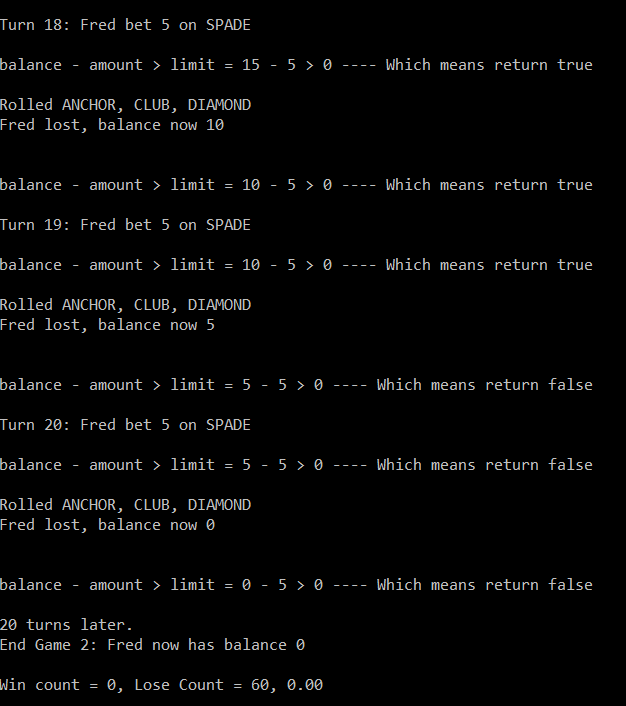


Figure 11 - Resolution of Bug 2 (b)

## Bug 3

Compiled the source code and executed the program to simulate the third test case which is also documented in the UAT documented located in the project repository/UAT folder.

### Replication

Output from the test demonstrating the buggy behavior is illustrated in the below figure 12 where we can see that Win Ratio is approx. 60% (0.60), where as it should be 0.42 based on the 8% bias to the house.

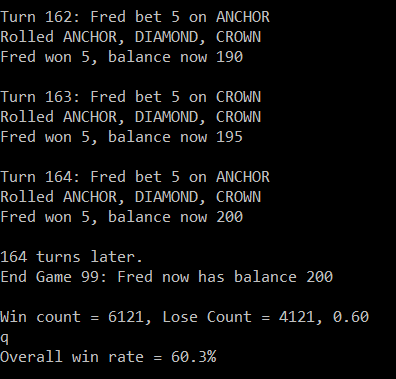


Figure - Bug 3 Replication

### Simplification

Added another test method *testBug3()* to the same class *testGame* to replicate the Bug 3 where win ratio is approx. 60%.

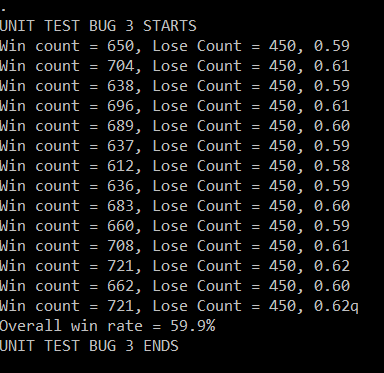


Figure - Replicate Bug 3

### Tracing

Tracing back program dependences show that there is a problem with diceValue.getRandom() function. Added some extra logging to make sure it’s the root of the Bug 3.

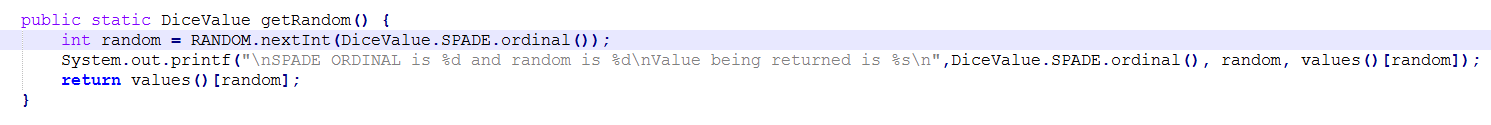


Figure 14 - Bug 3 Tracing (a)

Figure 14 (b) shows the variable values after the dice roll function is called.

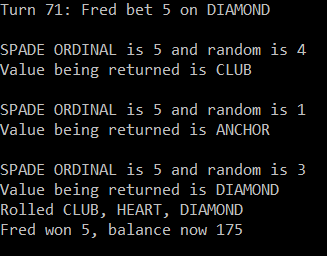


Figure 15 - Bug 3 Tracing (b)

DiceValue.getRandom() function is escaping one value which is SPADE which reduces the chances of losing. If we include SPADE as well in the list of values, chance of winning will decrease.

### Resolution

To fix the issue we simply added 1 to the SPADE ordinal as by definition n in function *nextInt(int n)* is exclusive.

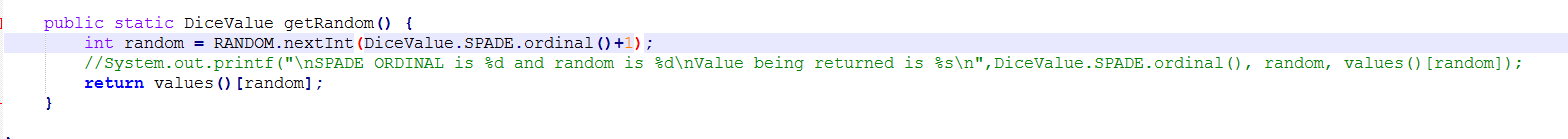


Figure - Fix Bug 3 (a)

Figure 16 (b) and figure 16 (c) shows the output of correct operation after resolution. Figure 16(b) is Unit Test Run and 16(c) is actual production application run.

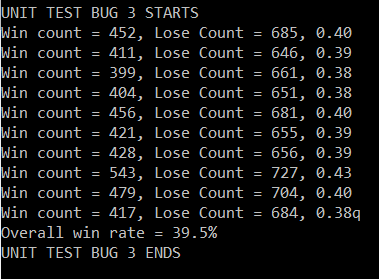


Figure - Fix Bug 3 (b)

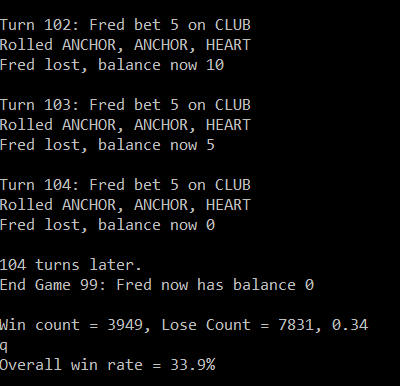


Figure - Fix Bug 3 (c)