| **Test Name** | | | Game does not pay out at the correct level | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Use Case Tested:** | | | Crown and Anchor Game | | | |
| **Test Description:** | | | If the player wins on 1 match, the players balance will not increase | | | |
| **Pre-conditions** | | | Default game settings | | | |
| **Post-conditions** | | | After each match the player is able to check the output to determine if the balance is correct | | | |
| **Notes:** | | **At present a pass indicates that the bug does exist within the game. This is the expected behavior in order to confirm the bug is really a bug** | | | | |
| **Result (Pass/Fail/Warning/Incomplete)** | | **Fail – Indicates the bug does exist** | | | | |
|  | **TEST STEP** | | | **EXPECTED TEST RESULTS** | P | F |
|  | Run program | | |  | x |  |
|  | Search through the output from the game and locate the first instance of the player winning a match | | | Confirm the balance that the player had before the match started does not increase | x |  |
|  |  | | |  |  |  |

# Test Result

Output from game

Rolled CLUB, ANCHOR, DIAMOND

Fred lost, balance now 95

Turn 2: Fred bet 5 on CROWN

Rolled CLUB, ANCHOR, DIAMOND

Fred lost, balance now 90

Turn 3: Fred bet 5 on CLUB

Rolled CLUB, ANCHOR, DIAMOND

Fred won 5, balance now 90

Diagnose

From the above output from the game we can see that after three match’s the player had a win. Once the player won a match we can see that his balance did not increase as expected. The output indicates that the player won 5, which should have given his balance a total of 95; he had 90 due to the first to lose of the game. This test confirms that the bug does actually exist within the game.

From looking through the output of the game the balance calculation seems to be the issue, the game is subtracting correctly when the player loses but is failing to add to the balance on a win. Instead it is just keeping the balance as it was

The output below confirms my diagnoses as the player won a few games in a row and never increased his balance.

Extra output to confirm diagnoses

Turn 4: Fred bet 5 on HEART

Rolled CLUB, ANCHOR, DIAMOND

Fred lost, balance now 85

Turn 5: Fred bet 5 on ANCHOR

Rolled CLUB, ANCHOR, DIAMOND

Fred won 5, balance now 85

Turn 6: Fred bet 5 on CLUB

Rolled CLUB, ANCHOR, DIAMOND

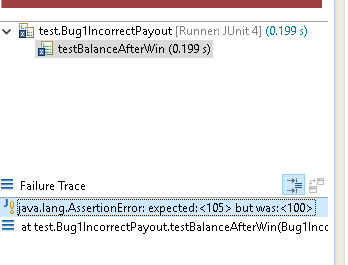
Fred won 5, balance now 85

Turn 7: Fred bet 5 on CLUB

Rolled CLUB, ANCHOR, DIAMOND

Fred won 5, balance now 85

# Automated Test with Junit to show the bug



The screenshot above shows the results of a Junit test that automates the bug, the fail of the Junit test indicates that the bug does exist and the program is not increasing the players balance correctly on a win. In order to indicate the bug within an automated process the Junit test fails on checking the players balance. Even though the fail occurs on the check balance method this does not indicate where the actual bug is. As the bug would be happening more likely to happen when the program calculates the value for a win/lose

# Tracing the code back to find the bug

The bug is happening due to the fact that the game takes the bet away from the player before the game starts. When the player wins a match this value is never taken into account which results in the player gaining back the initial minus from there balance without the win being taken into account. We can confirm this by tracing the code back to where all the calculations take place. In order to do this we need to first start with the file Main.java

The first line that I looked at was line 56 within Main.java. The section of code that I was looking for was something that would indicate the system adding to the balance, or removing from the balance. I noticed that on line 56 the system called the function player.getBalance(). This was a dead end as all this function is doing is returning the current balance for the player. This is also confirmed on line 61 when the player loses. This file does not actually do the calculations for a win/lose. So we need to head back a little and check what is actually going on.

We know the game loop starts on line 21, and we also know that the output for a win/lose starts on line 54, so the calculations has to been done somewhere in between these lines. On line 48 we can see the statement

**int** winnings = game.playRound(player, pick, bet);

This line seems like it might be a good spot to start looking for the calculations. So with that said lets open up the Game.java file and find this function. We can find this function on line 27 of Game.java if we move down a few lines within this function to line 32 we can see a method call that indicates the bet is taken away from the player before the game has even started

player.takeBet(bet);

So at this point within the game we know the player has lost the bet amount before anything within the game happens. We need to keep this in mind as we move down this function.

Now that we know that the player loses their bet amount before the game even starts we need to find the section that calculates there winning if the player does in fact win the round. This can be seen on line 42

**int** winnings = matches \* bet;

This one line here actually shows us the bug, as I will now explain. The winnings is calculated by the number of match’s times the bet amount. So if the player has one match and bets five, we actually end up with the following equation

**int** winnings = 1 \* 5;

This equals

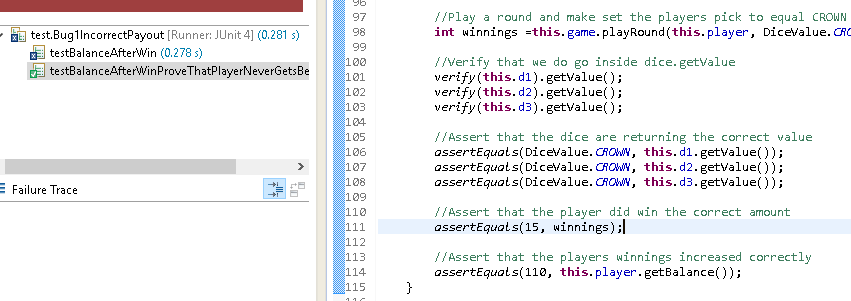
**int** winnings =5;

Now that is correct in its self, but if you remember beforehand that the game has already taken away the bet amount from the balance, so what this means is, if we give the player a balance of 100 and take the bet away, so say 5, we end up with a balance of 95, now when the player wins the win amount is only calculated without taking into account what the player has already betted. So in this case we end up adding 5 back onto which gives the player a balance of 100 again, they never actually receive the correct winning amount just what they won minus the bet amount.

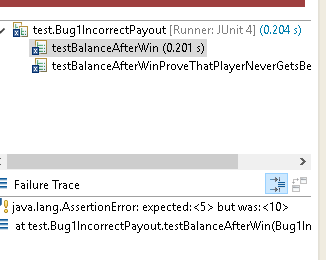
There are two solutions that come to mind in order to fix this bug, the first solution would be to not remove the bet from the player until it is determined if the win or lose, the second solution would be to add the players bet back onto the winning amount. To me the second solution would be the better solution to the issue as only removing the bet after a loss could introduce another bug into the game, so with that said the winning calculation on line 42 needs to be changed to

**int** winnings = (matches \* bet) +bet;

Before I change the code to confirm my solution will fix the bug I want to first run a Junit test that does in fact confirm the players balance will always be the winnings amount –the bet, we can do this by forcing the player to win with three matches, which means they should receive back 10, and not 15 (remember we lose the bet amount regardless), the expected result for the player balance in the Junit test is 110 for the player balance, which if happens does confirm we found the bug. The below screen shot indicates we have in fact found the bug



After correcting the code with solution two, I found out that even though solution two will fix the bug, it actually introduces anther bug, this is confirmed from the first Junit test where I check the winnings amount. Because this solution changes the reported winnings I am going to say that this solution is not correct. The next solution that I will do will be to move the plus of the bet into line 46; this will allow the program to report back the correct winnings will also taking into account the bet amount. The below image shows how solution two failed.



After running a Junit test after implementing solution three. I can confirm that I have fixed the bug any the program is now functioning as it should. The below screen shot indicates that the bug has been fixed, The failed Junit is expected as it was testing the bug to confirm that I was looking at the correct place.

