# **Noman Ahmed Khan**

# **11567969**

# **ITC205**

# **Professional Programming Practice**

# **Debugging Logbook**

Skip Page!

## **Replication:**

Under this process, I will be creating User Acceptance Test Cases to report the bugs that have been found in the application. The output will also be posted which shows the buggy behaviour of the application.

### **Known Bugs:**

### Bug # 1:

**Bug Name:** Game not paying out at correct level.

**Bug Description:** Whenever a player wins on a match, the winnings amount in not added in the correct manner. The player only receives but does not get back the amount that they originally betted.

**Test Case 1:**

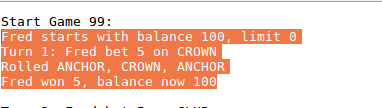
**Overview:**

Under this test I will carry out the testing to check whether the amount the player receives after winning a match is correct or not.

**Test:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 1a | Play the game, when the player wins one match, check whether their balance is increased by the amount of money they were supposed to receive on winning or not. | Player "Fred" starts the game with balance 100 and bets $5 on Crown. The player wins the turn as he rolls one Crown. | Since the bug has already been reported our expected output will have error. The result will be that instead of balance becoming $105 after winning, the balance will stay $100. The player only gets back the winning but does not get back the original betting amount. | Shown in the screenshot below. |

**Screenshot test 1a:**



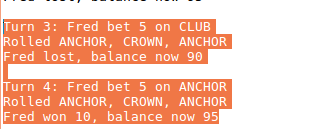
**Explanation:**

As it can be seen in the screenshot above, the player bet $5 and won $5 has he rolled only one matching die. Instead of the balance becoming $105 it remains $100.

**Test:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 1b | Play the game, when the player wins one match, check whether their balance is increased by the amount of money they were supposed to receive on winning or not. | Player "Fred" starts the turn with balance 90 and bets $5 on Anchor. The player wins the turn as he rolls two Anchors. | Since the bug has already been reported our expected output will have error. The result will be that instead of balance becoming $100 after winning, the balance will become $95. The player only gets back the winning but does not get back the original betting amount. | Shown in the screenshot below. |

**Screenshot test 1b:**



**Explanation:**

As you can see in the screenshot above, Fred starts the 4th turn with balance $90 and bets $5 on Anchor. He rolls two Anchors according to which his winnings are $10. His balance should now become $100 as he should get back the original amount plus, the winnings. However, the balance is now only $95 and not $100.

Bug # 2:

**Bug Name:**  Player unable to reach the betting limit.

**Bug Description:** As the betting limit is set to 0, the game should stop only when the balance of player has gone down to $0. However, the game ends with the player still having a balance of $5 remaining.

**Test Case 2:**

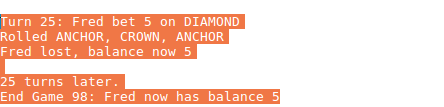
**Overview:**

Under this test I will carry out the testing to check whether the game ends before the players balance has gone down to $0.

**Test:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 2a | Play the game, check whether the game ends before the limit reaches $0 or not. | Keep playing the game until the game ends. Check the value of the remaining balance at the time the game ends. | As this is a known bug, we expect the buggy behaviour in the output. The game will end with the player still having a balance of $5 remaining. | Shown below. |

**Screenshot test 2a:**



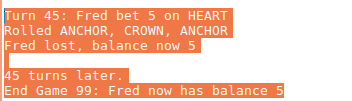
**Explanation:**

As you can see in the screenshot above, the game ends after the 25th turn while the balance of Fred is still $5 and not 0.

**Test:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 2b | Play the game, check whether the game ends before the limit reaches $0 or not. | Keep playing the game until the game ends. Check the value of the remaining balance at the time the game ends. | As this is a known bug, we expect the buggy behaviour in the output. The game will end with the player still having a balance of $5 remaining. | Shown below. |

**Screenshot test 2b:**



Explanation:

It can be seen in the above screenshot, that even for the 99th game, the game ends after 45 turns but the remaining balance that Fred has is still $5 and he has not yet reached the limit of $0.

### Bug # 3:

**Bug Name:** Odds of winning the game are not correct.

**Bug Description:** The win/loss ratio should be around 0.42 as the game is 8% bias in the favour of the house. However, this does not appear to be the case. The ratio is at times around 0.42 but also, many times it goes around 0.60 and stays there.

**Test Case 3:**

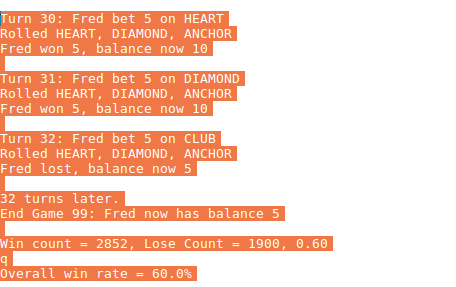
**Overview:**

Under this test I will carry out the testing to check whether the win/loss ratio for the game is around the amount it should be at which is 0.42.

**Test:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 3a | Play the game until the end, once the game finishes check whether the win/loss ratio is around the amount is should be or not. | Play game until reaching the end. | Win percentage should not be around 42% as this is known and reported bug so we should expect the faulty behaviour. | Shown below. |

**Screenshot test 3a:**



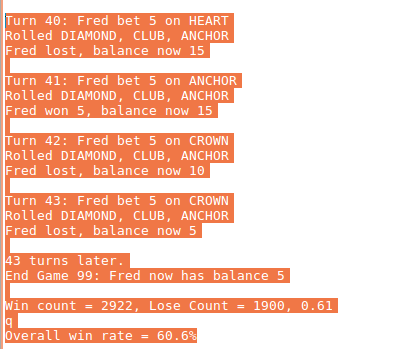
**Explanation:**

As seen in the screenshot above, when the game ends, the win/loss ratio is 0.60. This means that the player had an overall win rate of 60 % which a lot higher than it should be (42%). It clearly shows the bug in the application.

**Test:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 3b | Play the game until the end, once the game finishes check whether the win/loss ratio is around the amount is should be or not. | Play game until reaching the end. | Win percentage should not be around 42% as this is known and reported bug so we should expect the faulty behaviour. | Shown below. |

**Screenshot test 3b:**



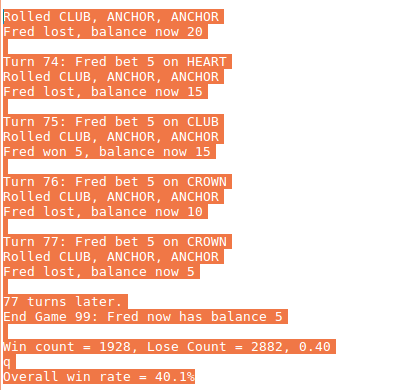
**Explanation:**

As seen in the screenshot above, the overall win rate is again around 60.6% with ratio 0.61. This is not what the game is programmed to be. According to the rules it should have an 8% bias in the favour of house with winning percentage to be around 42. The bug is clearly present.

**Test:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 3c | Play the game until the end, once the game finishes check whether the win/loss ratio is around the amount is should be or not. | Play game until reaching the end. | Win percentage should not be around 42% as this is known and reported bug so we should expect the faulty behaviour. | Shown below. |

**Screenshot test 3c:**



**Explanation:**

As seen in the screenshot above, the winning rate is 40.1% with the ratio around 0.40. This is good news that the bug was not present this time around however, it still does not mean that the bug will not come back in the later games.

## **Unknown Bugs:**

The following bugs were not reported, and have been found while trying to resolve other reported or known bugs.

### Bug # 4:

**Bug Name:** The dice rolled always have the same value.

**Bug Description:** Whenever the dice are rolled, they get a fixed value for one set of 100 games. This means, that each time the values are rolled. If you run the application again, different values get fixed however they stay the same for that set of 100 games.

**Test Case 4:**

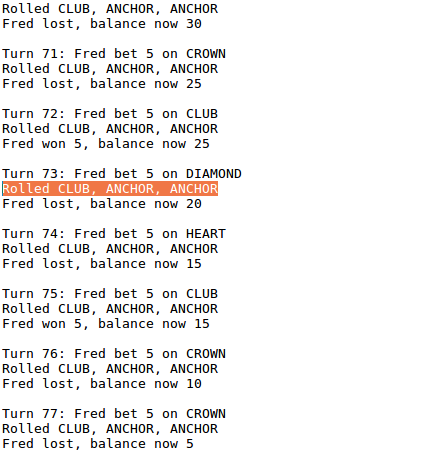
**Overview:**

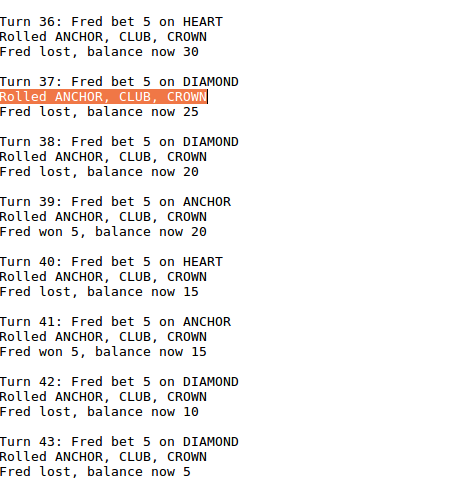
Under this test I will carry out the testing to check whether the value of rolled dice stays the same for each turn or if it changes and turns out to be random.

**Test:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 4 | Play the game and check whether the rolled values of the dice are random each time or remain the same. | Play game until reaching the end. | As this is a bug, the expected result is that the dice rolled values remain the same for every turn. | Shown below. |

**Screenshot test 4:**





**Explanation:**

As seen in the above screenshots, the rolled values of the dice remain fixed. In the first screenshot it is a repetition of CLUB, ANCHOR, ANCHOR and in the second screenshot it is a repetition of ANCHOR, CLUB, CROWN in each turn. The bug is clearly visible.

## Bug # 5:

**Bug Name:** Spade is never rolled.

**Bug Description:**  No matter how many games you play, each time you will notice that spade never gets rolled on any one of the die's.

**Test Case 4:**

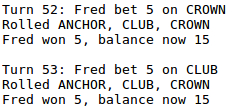
**Overview:**

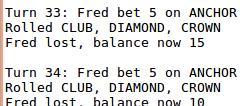
Under this test I will carry out the testing to check whether the Spade is rolled or not on the dice for any of the turns on any of the games.

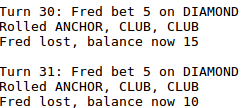
**Test:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # 5 | Description | Test Values | Expected Result | Actual Result |
| 5a,b,c | Run the game as usual and check whether the dice values are rolled to Spade anytime. | Play the game and check the values of the dice rolled. | As this is a found bug, the result should show the buggy behaviour which means that Spade will never get rolled. | Shown below. |

**Screenshots 5a, 5b, 5c:**







**Explanation:**

As seen above in the screenshots of test 5a, 5b and 5c the dice has never once rolled a Spade. These tests were carried out numerous times however only 3 results have been shown to reduce the length of the document.

## **Simplification:**

Under this process I will create simplified tests that will reproduce the buggy behaviour.

**Bug # 1: Game not Paying out at Correct Level**

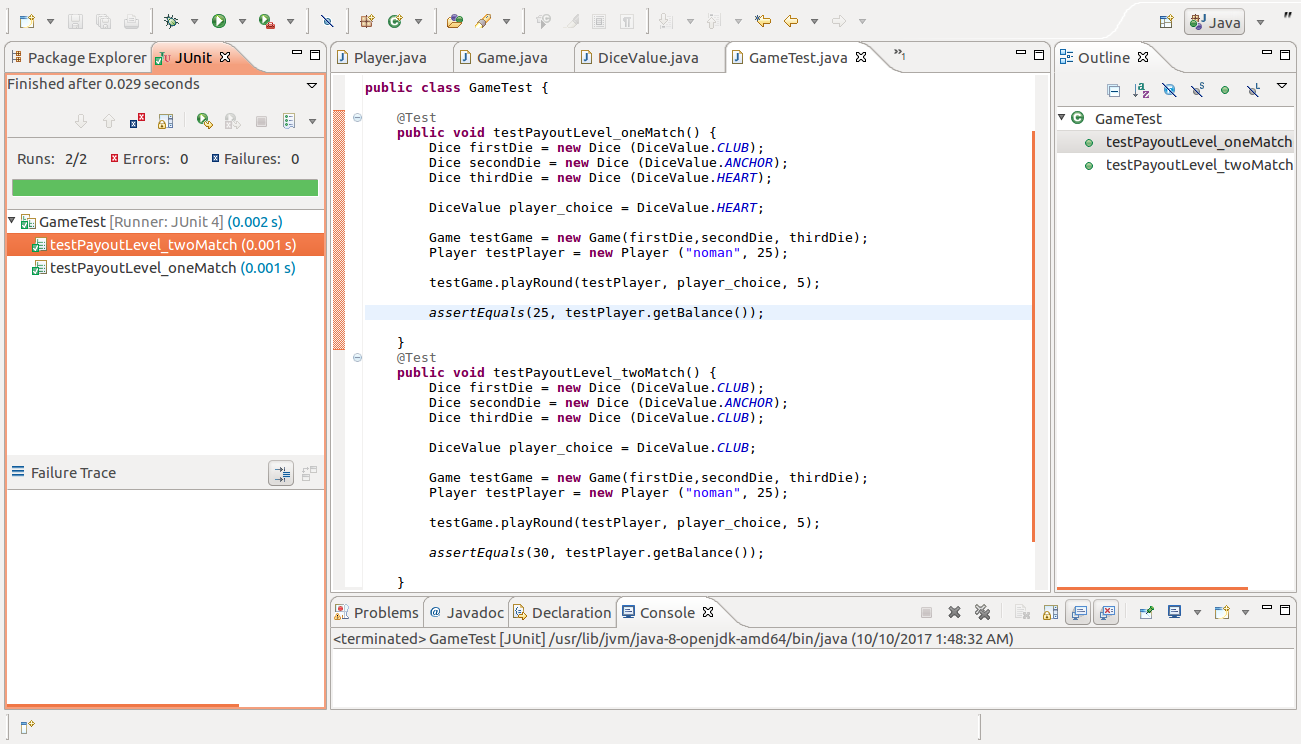
In order to test Bug 1, I created two simple JUnit tests that test the following:

Test1: This test's if the output shows buggy behaviour by not adding the correct amount to the balance when a player wins a match with one die matching.

Test2: This test's if the output shows buggy behaviour by not adding the correct amount to the balance when a player wins a match with two dice matching.

The code for this test was added to GitHub under the filename GameTest.java

Result is shown below:



**Explanation:**

As seen in the above screenshot, I have carried out two tests with JUnit framework. In the first test I made the player bet $5 on the Heart and made the outcome of rolled dice to have just one Heart present. According to this the player should have had the final balance of 30 after winning however since there is a bug, the player only has a final balance of 25 which is true for our AssertEquals test.

In the second test I made the player bet $5 on Club and made the outcome of the rolled dice to have two Clubs. According to this the player should have had the final balance of 35 after winning however, due to the bug present they will have final balance of 30 which is true for our AssertEquals test.

The above simplified tests have successfully reproduced the buggy behaviour.

**Bug # 2: Player Unable to Reach Betting Limit**

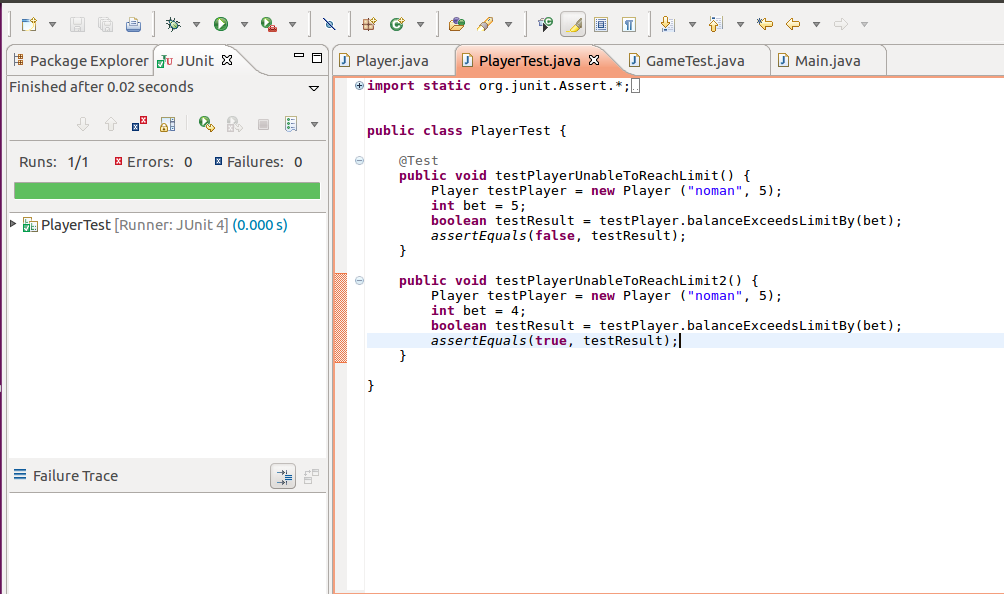
For simplification of bug 2 I created junit tests to replicate the buggy behaviour. The following tests were created:

Test1: set player balance to $5 and bet $5. This will return false with balanceExceedsLimitBy(bet) and therefore the player will not be able to place the bet.

Test2: set player balance to $5 however bet $4. This will allow player to place the bet as the limit will not have been exceeded or reached.

The test code has been committed to the GitHub repository.

The results are shown below:



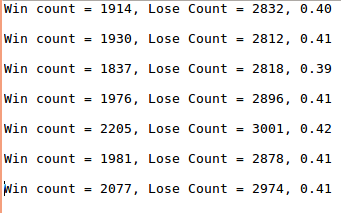
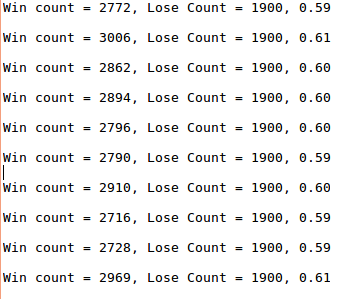
**Explanation:**

As seen in the above screenshot, the balanceExceedsLimitBy method returns false when the current balance and betting amount is same. This is the cause of the buggy behaviour in our program as the players are programmed to always bet the same amount in every turn and when they reach the last $5, the method returns false which leads the loop within the program to end, ending the game.

**Bug # 3: Odds of winning the game are not correct**

For the simplification of bug 3 I modified the code that it would run automated games and just show the ratio output for each game. The results would then show id the buggy behaviour is present the code or not.

The results for the automated tests shown below:



**Explanation:**

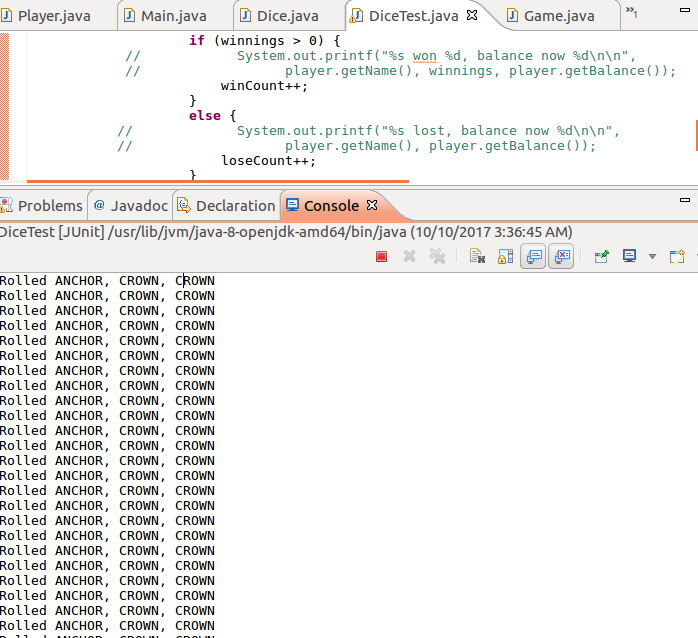
As seen in the above screenshots, I have replicated the buggy behaviour with the test code that I wrote and committed to GitHub. It can be seen that the ratio is fluctuating a lot but It is never really around 0.42 which is requirement for the game.

**Bug # 4: The diced rolled always have the same value**

For simplification of bug 4 I created a class DiceTest and replicated the running of application by modifying some of the code so that it only shows the result of the dice rolled. The result for the DiceTest class showed the buggy behaviour present in the code.

The code was committed to GitHub repository.

The result is shown below:



**Explanation:**

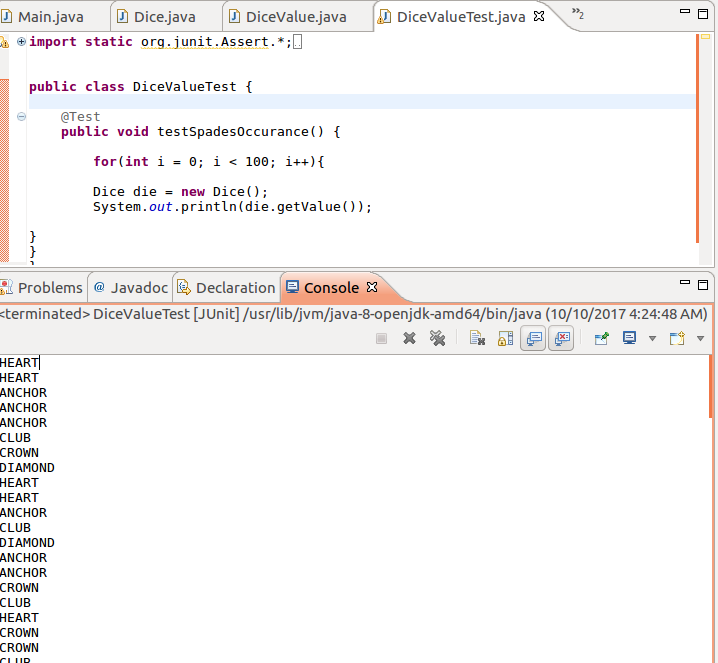
As seen in the screenshot above, the automated test shows the buggy behaviour of the application. Each time the dice were rolled, the result was the same as Anchor Crown and Crown were rolled.

**Bug # 5: Spade is never rolled**

For the simplification of bug 5 I created a Junit class DiceValueTest and carried out an automated test by printing out the value of Die using the getRandom method of DiceValue for 100 different values. I noticed the result that it never showed the value of spade once.

The test code was committed to GitHub.

The result is shown below:



**Explanation:**

As you can see in the screenshot above, spade is never once added to the value die using the getRandom method of DiceValue enum class. This same method is used in the application and therefore spade is never rolled and hence the bug is there.

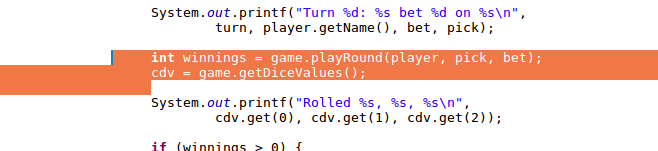
## **Tracing:**

Under this process I will be tracing the bug back to where it is actually located within the source code. For tracing the bug, I used certain hypothesis and with the help of simplification process done earlier, I was able to locate the origin of the bugs.

**Bug # 1 Game not paying out at correct level**

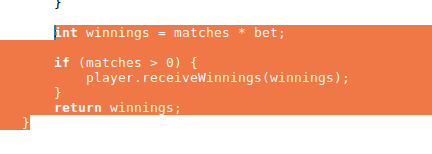
For tracing bug 1, I looked into the results of the simplification process and realized that the problem lies somewhere in balance or winnings incrementation at the end of each turn. I came up with the hypothesis that the bet is not being added into the balance even though the winnings are. I looked back at the code wherever the balance is being incremented or the winnings are being added to the balance and finally came to the origin of the code.

The following screenshots show the highlighted area where from where I traced back to the bug.



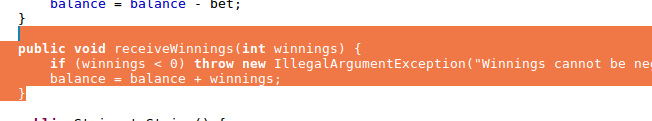
Here in the Main class the game is played with the help of playRound method of Game class. The winnings are returned back from this method.

I then had a look at playRound method and highlighted the following code:



I noticed in playRound that the receiveWinnings method takes the winnings as an input so then I had a look at Player class for the working of this method.

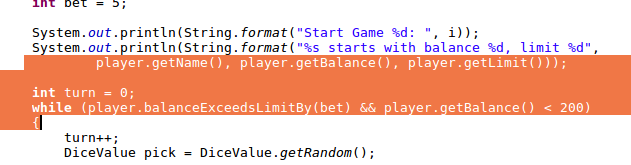
The following is highlighted code in Player class's receive Winnings method:



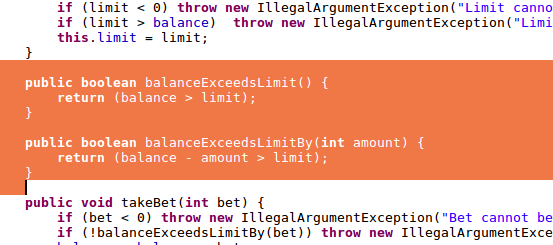
After having a look at this part of the code, I figured out that this is where the bug is located. I noticed how the winnings are added into the balance, however the bet that was deducted from balance was not given back to the player. This is the part that needs to be fixed which I will do in resolution.

**Bug # 2: Player unable to reach betting limit**

For the tracing of bug 2, I had a look at the simplification process results and figured out that the bug lied somewhere in the bettingExceedsLimit methods of the Player class. The following codes highlight the origin of the bug:



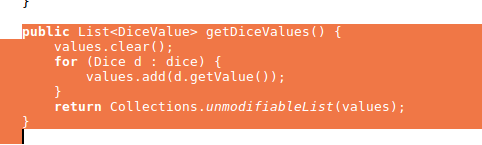
As you can see the screenshot above, the player only plays the round until when the return of balanceExceedsLimitBy method returns. I then had the look at the following code to check what the problem may be in balanceExceedsLimitBy method:

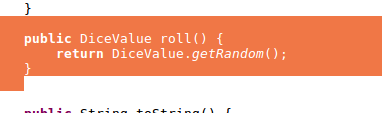


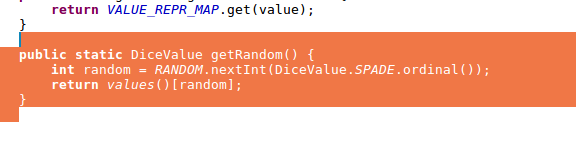
In the above screenshot is the highlighted part where the bug lies. It can be seen that the equation used in the method will always return false whenever the betting amount is the same as the balance as the operator used is ">" instead of ">=". This means that if the betting amount is same as balance it will become 0 and make the condition false, hence the betting limit not being able to reach 0.

**Bug # 3: Odds of winning the game are not correct**

For the tracing of bug 3 i had a look at the results of the simplification process of this bug and realized that the bug lied somewhere in the logic of the game and not calculations of win/loss ratio itself as they were being done correctly. I noticed that the game was rolling the same values of the dice each time and this was having a great effect on the winning percentage. I therefore had a look into the code of getting the random values from dice. I also noticed how spade was never being rolled and had a look at the code for that too. I highlighted the code that I though was buggy and it shown below:



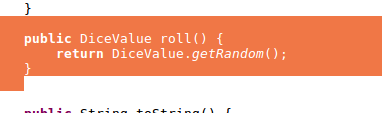




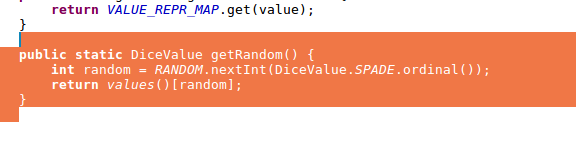
I had a look at the above code in Game, Dice and DiceValue classes that seemed to be causing the problem in getting the dice to roll the same values each time and hence causing the win ratio to increase quite a bit. This is surely where the bug has to be and it will be resolved in the next process.

**Bug # 4: The dice rolled always have the same value**

For the tracing of bug 4 I had a look at the results of the simplification processes. I noticed how there was a problem in either the roll method within Dice class or the getRandom method of the DiceValue class. I investigated the code for these two methods in their classes and highlighted where the buggy code may have been located. The following screenshots show the highlighted code:



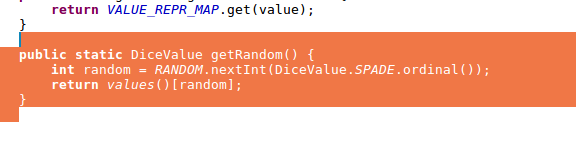
As you can see that the roll method is getting the values back from DiceValue's getRandom method so then I had a look at the following code for further investigation:



From having a look at the above code, I kind of figured out where the problem was in the code. I was not hundred percent sure but I knew that these were the only possible places I could fix the bug in. The origin was not quite found but I was ready to try out different solutions when working on the resolution process.

**Bug # 5: Spade never gets rolled**

For the tracing of bug 5 I had a look at the results of the simplification processes. Also, while tracing the previous two bugs, bug 3 – odds not being correct and bug 4 – rolling same dice value, I noticed the origin of the fifth bug. The following code has been highlighted where the buggy behaviour may be present:



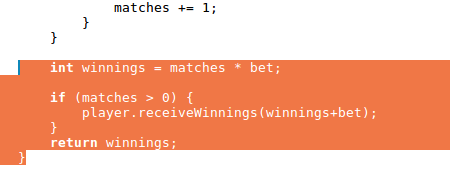
As seen in the screenshot above, the nextInt method is used to get the random values from the HashMap. The value of DiceValue.SPADE.ordinal will be 5 and this value will always be skipped by the nextInt method, therefor SPADE will never be included in the random value selection hence the bug is created.

## **Resolution:**

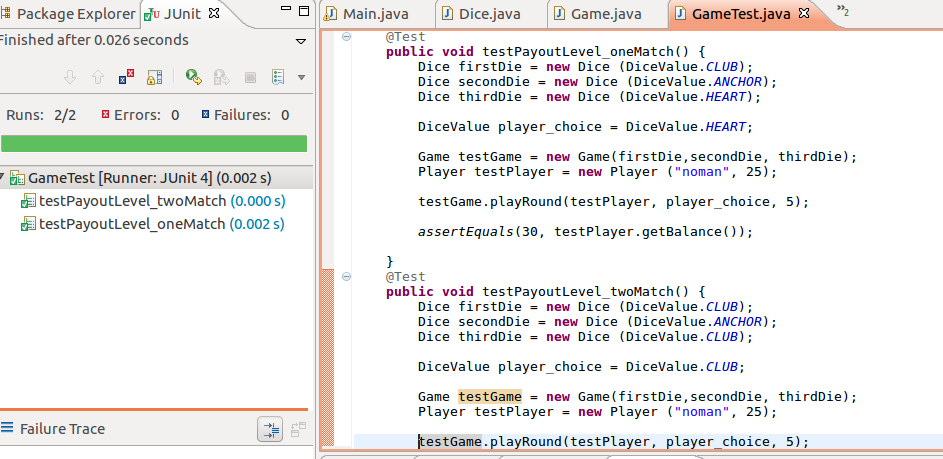
Under the resolution process, I fixed the bugs that were reported and discovered during the first three stages. After fixing the bugs I carried out some more tests to make sure the program was no longer having the buggy behaviour.

**Resolution Bug # 1: Not paying out at correct level**

The following changes were made to the code in order to fix the bug:



The automated tests now returned the correct results as shown in the image below:



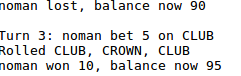
As seen in the image above, the game is now paying out at correct level for the automated tests. It is correct for one match as well as two matches. This is further confirmation that the bug seems to be fixed.

The following User Acceptance Tests also confirms the resolution of bug 1.

**Before Resolution**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 1a | Check whether the winnings are added correctly to the balance. | Player noman had balance 90 and bet $5 on Club. He rolled two Clubs with dice. | His balance should now be 100 after winning on the turn however his balance is only 95. | Shown below. |

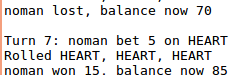
**Result:**



**After Resolution**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 1b | Check whether the winnings are added correctly to the balance. | Player noman had balance 70 and bet $5 on Heart. He rolled three Hearts with dice. | His balance should now be 85 after winning on the turn. | Shown below. |

**Result:**

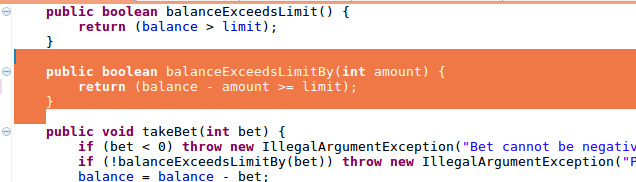


**Explanation:**

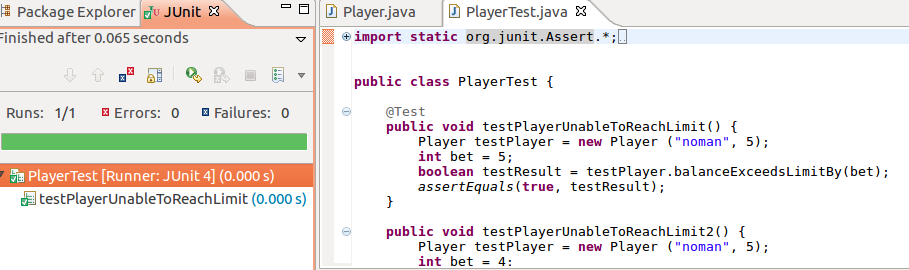
As seen from the UAT tests above, it is clear that the buggy behaviour has been removed and the game is now paying out at correct levels.

**Resolution Bug # 2: Player unable to reach betting limit**

The following changes were made to the code in order to fix the bug.



The automated tests returned the correct results as shown in the image below:



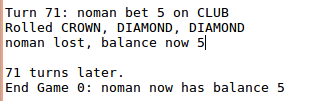
As you can see in the image above, the assertEquals now returns true even when the betting amount is same as the balance, which means it will now go down to 0.

The following UAT tests also confirm the resolution of bug 2:

**Before Resolution:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 2a | Check whether the limit is reaching 0 when the game ends. | Play game until reaching 0 balance. | At the end of the game, the balance will still be greater than 0 showing the buggy behaviour. | Shown below. |

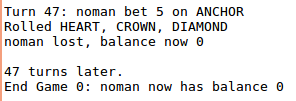
**Result:**



**After Resolution**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 2b | Check whether the limit is being reached or not at the end of the game. | Play the game until the end and check the remaining balance. | His balance should now be 0 at the end of the game meaning that the betting limit has been reached | Shown below. |

**Result:**

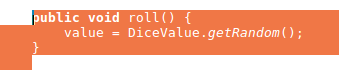


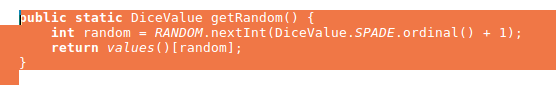
**Explanation:**

As you can see from the results of the UAT tests above, the betting limit was not being reached before (test 2a) however after the bug has been fixed, the betting limit is now being reached to 0 (test 2b).

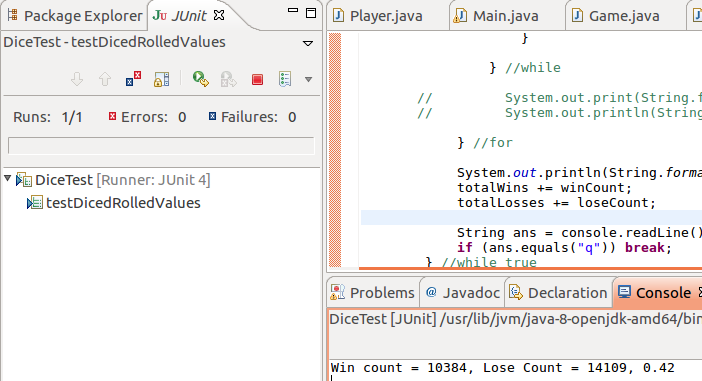
**Resolution Bug # 3: Odds not being correct.**

The following changes were made to the code in order to fix the third bug:





The automated test returned the correct results as shown in the images below:







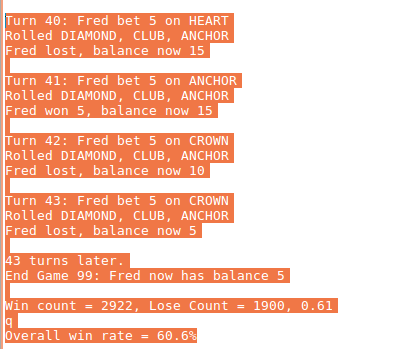
As you can see in the image above the winning ratio is now consistently 0.42 as it should be after the fixing of the bugs.

The following UAT tests further clarify the resolution of the bug:

**Before Resolution:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 3a | Check whether the winning percentage is around 0.42 or not at the end of the game. | Play game until end and check the winning ratio. | At the end of the game, the ratio will still be greater than 0.42 showing the buggy behaviour. | Shown below. |

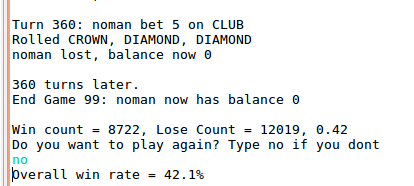
**Result:**



**After Resolution:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 3b | Check whether the winning percentage is around 0.42 or not at the end of the game. | Play game until end and check the winning ratio. | At the end of the game, the ratio will be around 0.42 which shows that the buggy behaviour has been eliminated. | Shown below. |

**Result:**

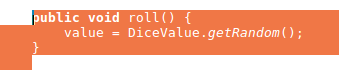


**Explanation:**

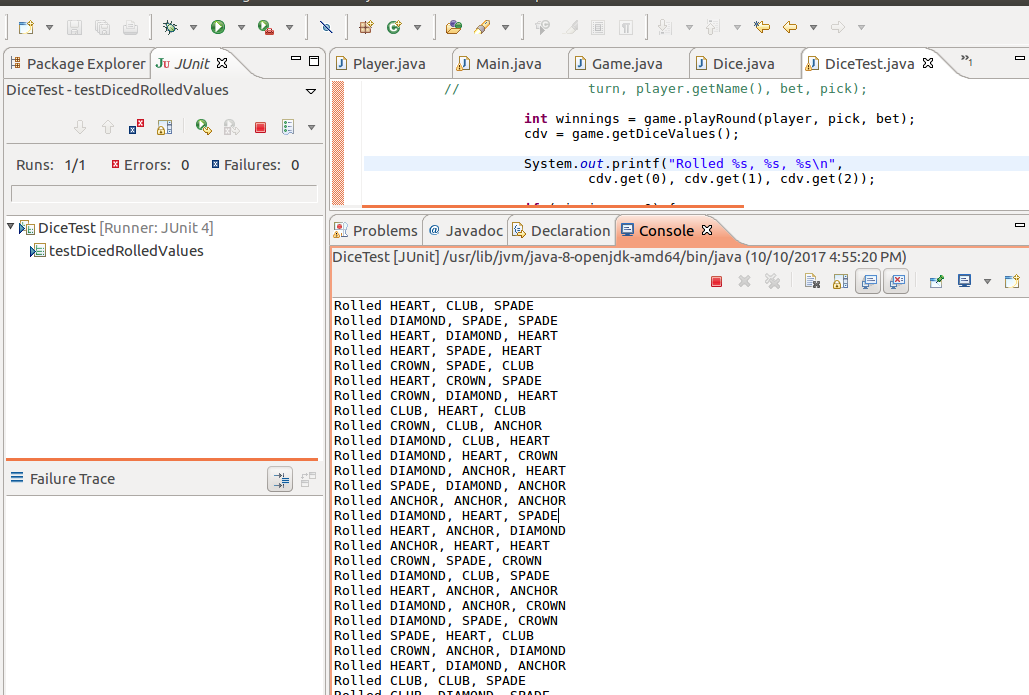
As you can see from the results of the before and after resolution UAT tests, the bug has been fixed as the win/loss ratio is now 0.42 after 100 games.

**g # 4: Dice rolled same values every time**

The following changes were made in order to fix the bug:



The automated tests returned the correct results as shown in the image below:



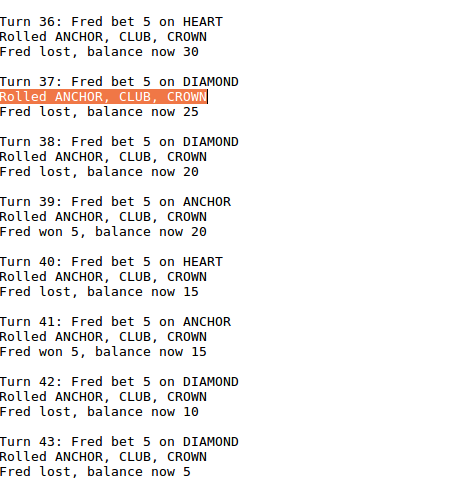
As you can see in the image above, the automated test shows that the rolled values were now changed and random every time and the buggy behaviour was no longer present.

The following UAT tests also confirm the resolution of the Bug.

**Before Resolution:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 4a | Check whether the Dice rolled the same values or random values each time the game was played. | Play game. | The values were same each time as the bug was present. | Shown below. |

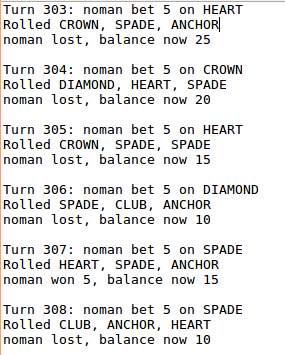
**Result:**



**After Resolution:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Values | Expected Result | Actual Result |
| 4b | Check whether the Dice rolled the same values or random values every time the game was played. | Play game. | The values are now randomly rolled showing that the bug is no longer present. | Shown below. |

**Result:**

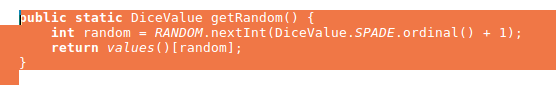


**Explanation:**

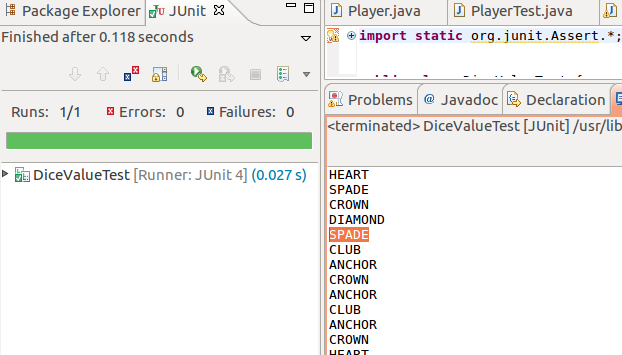
As you can see from the test results above, the bug has been fixed and now the rolled dice values are random each time.

**Resolution Bug #5: Spade never gets rolled**

The following changes were made to the code in order to fix the bug.



The automated tests returned correct results which can be seen in the image below:



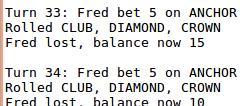
As seen in the image above, the automated tests returned correct values this time. Spade is now visible which means it is now being rolled by die and that the bug is no longer preset.

The UAT tests below also confirm the resolution of the bug:

**Before Resolution:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # 5 | Description | Test Values | Expected Result | Actual Result |
| 5a | Run the game as usual and check whether the dice values are rolled to Spade anytime. | Play the game and check the values of the dice rolled. | As this is a found bug, the result should show the buggy behaviour which means that Spade will never get rolled. | Shown below. |

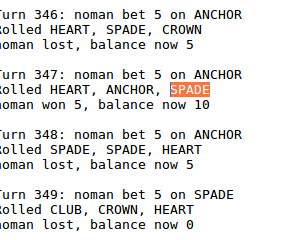
**Result:**



**After Resolution:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # 5 | Description | Test Values | Expected Result | Actual Result |
| 5b | Run the game as usual and check whether the dice values are rolled to Spade anytime. | Play the game and check the values of the dice rolled. | As the bug is no longer present, the results will show that spade is now being rolled. | Shown below. |

**Result:**



**Explanation:**

As seen in the above screenshots, SPADE is now being rolled by the dice, which was not the case before resolution. This means that the bug has been fixed.

## **Additional Bugs (Features) Fixed:**

Apart from the Known and Unknown bugs that have been fixed above, there were some rules of the game that were not being applied. Further changes have been made to the code in order to allow following features to be used:

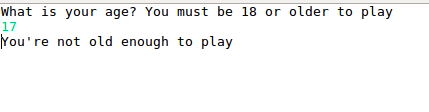
1. Not allow users under 18 to play the game.
2. Ask for user's name when they play the game.
3. Ask for how much balance the user wants to start with.
4. Ask user how much money they want to bet. Bet 0 to end round.
5. Ask users if they want to play again.
6. Ask new users their age and name.

The following screenshots show the above features and rules being applied:

**1: Do not allow users user 18 to play the game:**

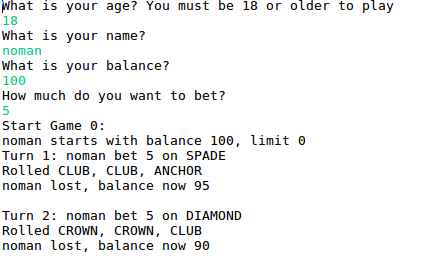
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Value | Expected Result | Actual Result |
| 1 | Check Whether the user is under 18 or not. | Enter user age as 17. | The game should output some message stating that user is not allowed to play the game. | **Shown below.** |

**Result:**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test # | Description | Test Value | Expected Result | Actual Result |
| 1.1 | Check Whether the user is under 18 or not. | Enter user age as 18. | The game should continue as normal and allow user to play. | **Shown below.** |

**Result:**



**2: Ask user to enter their name at the start of the game:**

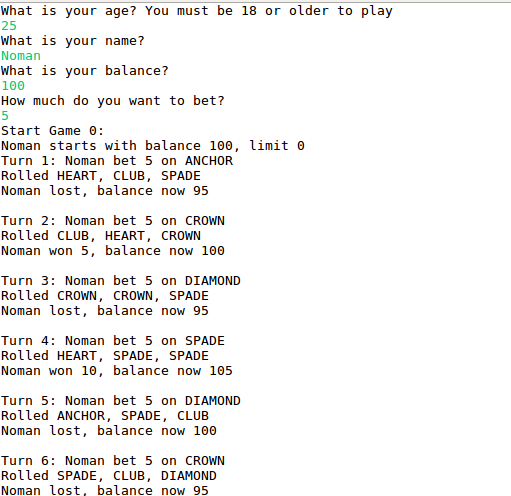
**3: Ask user how much balance they want to start with:**

**4: Ask user how much money they want to bet:**

The game is no longer automated. This means that every time it is run, the game prompts user to enter their age, after verifying the age, the game asks user to enter their name, how much balance they want to play with and the amount of money they want to bet each time.

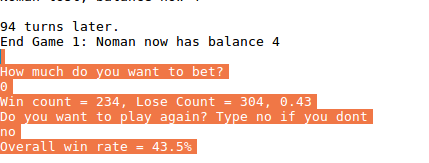
Before the game was automated and only Fred was playing the game with prefixed balance and betting amount. Now the game has become more interactive allowing more players to play.

The screenshots below show the result of the fix's made for 2, 3, and 4:



**Explanation:**

As seen in the image above, the user is now promted to enter their name, balance and betting amount. The game is then played for that user. After each game ends, the game asks user of they want to bet another amount.



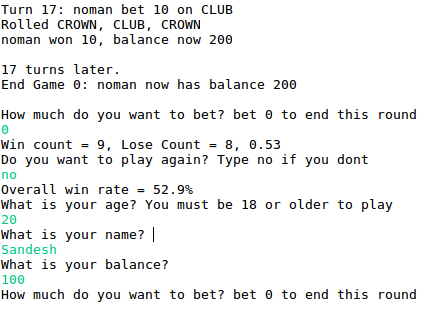
**Explanation:**

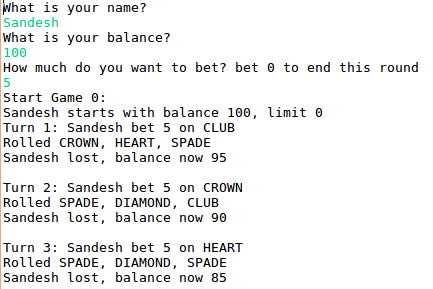
The game asks user if they want to play the game again, if they do, it starts again with asking how much money they want to bet, otherwise it shows them their winning percentage.

**7: Ask users if they want to play again.**

**8: Ask new users their age and name.**

The game asks users if they want to play the game again, if they say yes then the game asks for new balance amount and how much the user wants to bet. If they say no, it goes to age question and prompts new users to enter their age and name. The following screenshots show the above features in the game.





**Explanation:**

As you can see in the above screenshots, user noman ended the game and the game then went straight to asking new users their age and name and starting new game again. New user Sandesh started the game and as they were older than 18 they could play the game.