STUDENT NAME: HAI HA DO

Student number: 11635010

ITC515 - Assignment item 4

DEBUGGING

Contents

[1 Repository Information 3](#_Toc495914586)

[1.1 Location of Files in Repository 3](#_Toc495914587)

[2 Bug 1 - The player is not paid out correctly 5](#_Toc495914588)

[2.1 Replication 5](#_Toc495914589)

[2.2 Simplification 7](#_Toc495914590)

[2.3 Tracing 10](#_Toc495914591)

[2.4 Hypothesis 12](#_Toc495914592)

[2.5 Resolution 13](#_Toc495914593)

[2.6 Result 15](#_Toc495914594)

[2.6.1 Bug1Test.java 15](#_Toc495914595)

[2.6.2 Console from Main 16](#_Toc495914596)

[3 Bug 2 - Player cannot reach betting limit 17](#_Toc495914597)

[3.1 Replication 17](#_Toc495914598)

[3.2 Simplification 18](#_Toc495914599)

[3.3 Tracing 20](#_Toc495914600)

[3.4 Hypothesis 21](#_Toc495914601)

[3.5 Resolution 22](#_Toc495914602)

[3.6 Result 22](#_Toc495914603)

[3.6.1 Bug2Test.java 22](#_Toc495914604)

[3.6.2 Console from Main 23](#_Toc495914605)

[4 Bug 3 - The DiceValues are the same for each game 24](#_Toc495914606)

[4.1 Replication 24](#_Toc495914607)

[4.2 Simplification 25](#_Toc495914608)

[4.3 Tracing 26](#_Toc495914609)

[4.4 Hypotheses 28](#_Toc495914610)

[4.5 Resolution 29](#_Toc495914611)

[4.6 Result 30](#_Toc495914612)

[4.6.1 Bug3Test 30](#_Toc495914613)

[4.6.2 Console from Main 31](#_Toc495914614)

[5 Bug 4 - SPADE is ever rolled or guessed 32](#_Toc495914615)

[5.1 Replication 32](#_Toc495914616)

[5.2 Simplification 33](#_Toc495914617)

[5.3 Tracing 35](#_Toc495914618)

[5.4 Hypotheses 36](#_Toc495914619)

[5.5 Resolution 38](#_Toc495914620)

[5.6 Result 39](#_Toc495914621)

[5.6.1 Bug4Test 39](#_Toc495914622)

[5.6.2 Console from Main 40](#_Toc495914623)

[6 Bug 5 - Odds of game are incorrect 41](#_Toc495914624)

[6.1 Replication 41](#_Toc495914625)

[6.2 Simplification 42](#_Toc495914626)

[6.3 Tracing 45](#_Toc495914627)

[6.4 Hypothesis 46](#_Toc495914628)

[6.5 Resolution 46](#_Toc495914629)

[6.6 Result 46](#_Toc495914630)

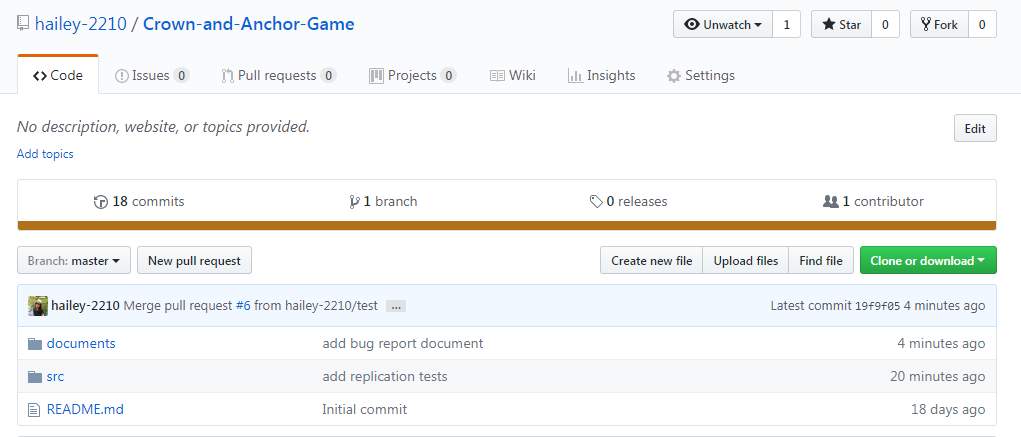
[6.6.1 Bug5Test 46](#_Toc495914631)

[6.6.2 Main Console 47](#_Toc495914632)

# Repository Information

The URL for the assignment 4 repository is:

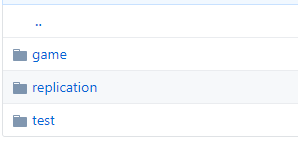
[https://github.com/hailey-2210/Crown-and-Anchor-Game.git](https://github.com/hailey-2210/Crown-and-Anchor-Game.git%20)

****

## Location of Files in Repository

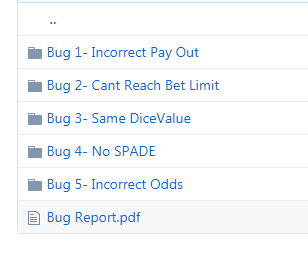
[**Crown-and-Anchor-Game**](https://github.com/hailey-2210/Crown-and-Anchor-Game)**/src/**

This consists of the source code for the game, unit test of individual files and replication test cases



[**Crown-and-Anchor-Game**](https://github.com/hailey-2210/Crown-and-Anchor-Game)**/documents/**

This consists of all relevant documents, including bug report. For each bugs, the files are located within the bug's folder.



# Bug 1 - The player is not paid out correctly

## Replication

| **Test Name** | Test balance increase correctly after winning |
| --- | --- |
| **Use Case Tested:** | Crown and Anchor Game |
| **Test Description:** | Test whether the player is paid the correct amount  In particular:   * If he makes one match, he should get the initial balance and the bet. * If he makes two matches, he should get initial balance plus two times the bet. * If he makes three matches, he should get the initial balance plus three times the bet. |
| **Pre-conditions** | Run the program to simulate the game. |
| **Post-conditions** | Player’s balance has increased by his bet. |

|  | **TEST STEP** | **EXPECTED TEST RESULTS** | **RESULT** |
| --- | --- | --- | --- |
|  | Run Main.java with player details:  Player name = “Fred”  Balance = 100  Limit = 0 | Console opens and results for 100 games are displayed in it. | Pass |
|  | Look at each turn and identify one where the player has made matches. | A turn should be presented | Pass |
|  | Look at the previous balance from the end of the previous turn, the bet amount, and the balance at the end of the identified turn. | The balance for the end of the identified turn should be equal to the previous balance plus the bet amount. | Fail |
|  | Repeat steps 2-3 two more times to identify and examine different turns. | Same as steps 2-3. | Fail |

Examples of bugs

|  |  |
| --- | --- |
| **EXAMPLES OF BUGS** | **RESULT** |
|  | Initial balance: 70  Balance after turn: 70  Expected: 75  Result: FAIL |
|  | Initial balance: 95  Balance after turn: 95  Expected: 100  Result: FAIL |
|  | Initial balance: 100  Balance after turn: 100  Expected: 105  Result: FAIL |
|  | Initial balance: 90  Balance after turn: 95  Expected: 100  Result: FAIL |
|  | Initial balance: 85  Balance after turn: 90  Expected: 95  Result: FAIL |
|  | Initial balance: 95  Balance after turn: 100  Expected: 105  Result: FAIL |

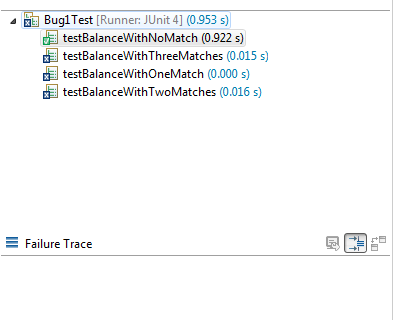
## Simplification

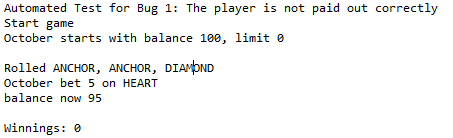
| **Test Name** | Test balance increase correctly after winning |
| --- | --- |
| **Use Case Tested:** | Automate the testing of errors in UAT Test 1 |
| **Test Description:** | Test whether the player is paid the correct amount  In particular:   * If he makes one match, he should get the initial balance and the bet. * If he makes two matches, he should get initial balance plus two times the bet. * If he makes three matches, he should get the initial balance plus three times the bet. |
| **Pre-conditions** | Single player "October" created, balance = 100, bet = 5  Each run to use a single value "HEART" as the player's pick.  Run game for 4 given combination of dice values. |
| **Post-conditions** | n/a |

|  | **TEST STEP** | **EXPECTED TEST RESULTS** | **RESULT** |
| --- | --- | --- | --- |
|  | Run Bug1Test.java | Junit test and Console are opened. | Pass |
|  | Check result of testBalanceWithNoMatch in Failure Trace | JUnit test should be no error and no failure | Pass |
|  | Check result of testBalanceWithNoMatch in Console | Balance = 95  Winnings = 0 | Pass |
|  | Check result of testBalanceWithOneMatch in Failure Trace | JUnit test should be no error and no failure | Fail |
|  | Check result of testBalanceWithOneMatch in Console | Balance = 105  Winnings = 5 | Fail |
|  | Check result of testBalanceWithTwoMatches in Failure Trace | JUnit test should be no error and no failure | Fail |
|  | Check result of testBalanceWithTwoMatches in Console | Balance = 110  Winnings = 10 | Fail |
|  | Check result of testBalanceWithThreeMatches in Failure Trace | JUnit test should be no error and no failure | Fail |
|  | Check result of testBalanceWithThreeMatches in Console | Balance = 115  Winnings = 15 | Fail |

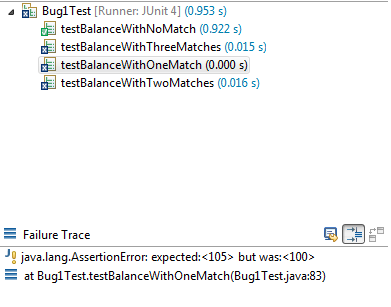
**Result**

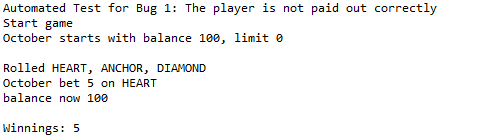
*1. No match: PASS*



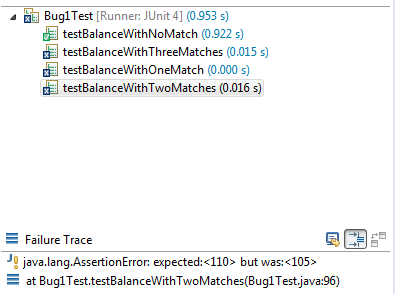


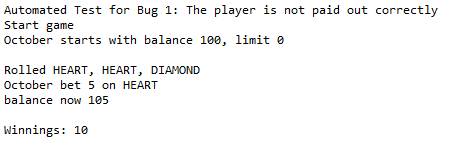
*2. One match: FAIL - Winning is correct but balance is wrong*

**

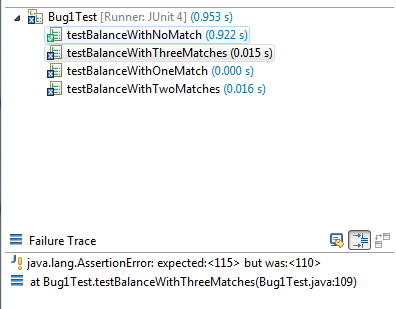
**

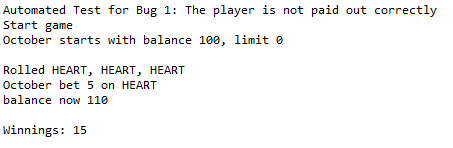
*3. Two matches: FAIL - Winning is correct but balance is wrong*

**

**

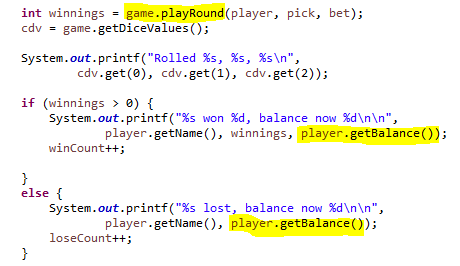
*4. Three matches: FAIL - Winning is correct but balance is wrong*

**

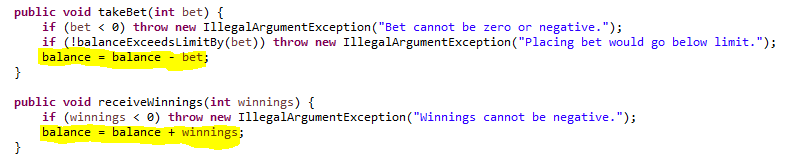


## Tracing

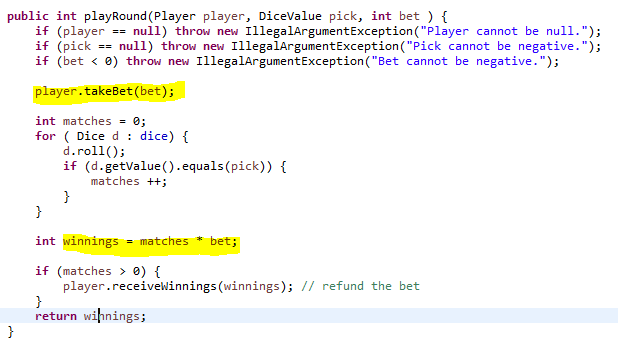
From the display of winning in **Main.java**, it can be seen that the winning is calculated in the method *playRound* in **Game.java.** The winning is displayed through the method *getBalance* in **Player.java**. There are probably two possibilities in this case, either problems with the winnings (**Player.java**) or the game (**Game.java**).



Tracing from the **Player.java**, it is showed that the balance before the turn and winning after turn are calculated as:

****

Tracing from the **Game.java**, the *playRound* method is calculated as:



So the bug here can be first explored that when the player plays round:

* Before the dice rolling*,* the player has to take the bet*:*
* If one match, instead of
* If two matches, instead of
* If three matches, instead of

## Hypothesis

There are three hypotheses in this case that needs to be verified:

* **Hypothesis 1:** The number of matches is calculated correctly.
* **Hypothesis 2:** The winnings are added correctly
* **Hypothesis 3:** The balance becomes incorrect after taking the bet

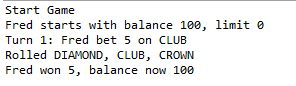
The testing of the hypotheses is conducted by putting breakpoints at

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

(2) player.receiveWinnings(winnings);

The debugging shows the results for the hypotheses as below:

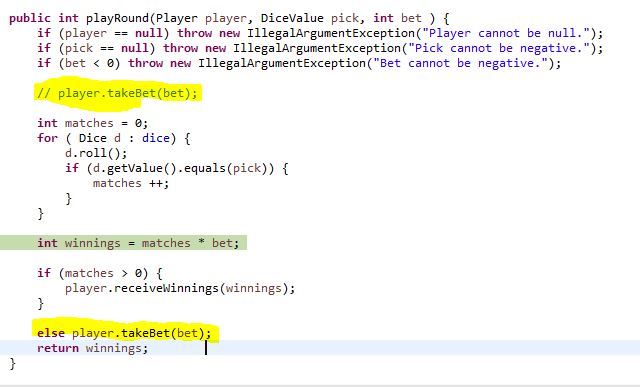
* Result from the console:



|  |  |
| --- | --- |
| **HYPOTHESIS** | **RESULT** |
| **Hypothesis 1:** The number of matches is calculated correctly. | At the stage of calculating match: the number of match is right (1 match). So (1) is true. |
| **Hypothesis 2:** The winnings are added correctly | At the stage of calculating winnings. So (2) is true. |
| **Hypothesis 3:** The balance becomes incorrect after taking the bet | Moving to the stage of take bet: the balance of player is wrong. It should be 100. So (3) is true. |

## Resolution

One possible solution is to take the bet after the game finishes. This means that if the player wins, the winnings will be added and no bet is taken from the balance. Otherwise, if the player makes no match, the bet will be taken from the balance.

This could be done by moving the takeBet method as below: 

There is no risk if fixing according to this, as other variables are not dependent on the balance at this stage. Also, if the matches is zero, the winnings will be calculated as zero.

## Result

### Bug1Test.java

| **BEFORE** | **AFTER** |
| --- | --- |
|  |  |
| **One match** | **One match** |
| **Two match** | **Two match** |
| **Three match** | **Three match** |

### Console from Main

Result from the console shows no error as well:

| **BEFORE** | **AFTER** |
| --- | --- |
|  |  |

# Bug 2 - Player cannot reach betting limit

## Replication

| **Test Name** | Test whether player can reach zero balance |
| --- | --- |
| **Use Case Tested:** | Crown and Anchor Game |
| **Test Description:** | Test whether player can play when he has $5 balance with the bet of $5 |
| **Pre-conditions** | Bug 1 has been fixed  Run the program to simulate the game. |
| **Post-conditions** | The player was able to reach zero balance. |

|  | **TEST STEP** | **EXPECTED TEST RESULTS** | **RESULT** |
| --- | --- | --- | --- |
|  | Run Main.java with player details:  Player name = “Fred”  Balance = 100  Limit = 0 | Console opens and the results of games are displayed. | Pass |
|  | Look for the line  “End Game: X Fred now has balance 0” | The line should exist. | Fail  “End Game: X Fred now has balance 5” |
|  | Repeat steps 1-2 two more times to identify and examine different turns. | Same as step 1-2 | Fail |

Examples of bugs

|  |  |
| --- | --- |
| **EXAMPLES OF BUGS** | **RESULT** |
|  | Result: FAIL |
|  | Result: FAIL |
|  | Result: FAIL |

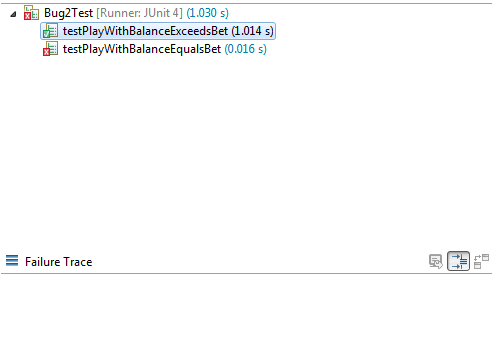
## Simplification

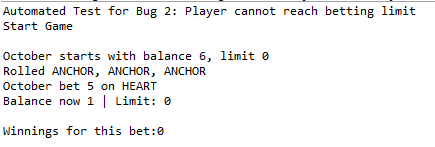
| **Test Name** | Test whether player can reach zero balance |
| --- | --- |
| **Use Case Tested:** | Automate the testing of errors in UAT Test 2 |
| **Test Description:** | Test whether player can play when he has $5 balance with the bet of $5 |
| **Pre-conditions** | Single player "October" created, bet = 5, limit = 0  Each run to use a single value "HEART" as the player's pick.  Run game for 2 different balances (one exceeds bet , one equal bet) |
| **Post-conditions** | The player was able to reach zero balance. |

|  | **TEST STEP** | **EXPECTED TEST RESULTS** | **RESULT** |
| --- | --- | --- | --- |
|  | Run Bug1Test.java | Junit test and Console are opened. | Pass |
|  | Check result of testPlayWithBalanceExceedsBet in Failure Trace | JUnit test should be no error and no failure | Pass |
|  | Check result of testPlayWithBalanceExceedsBet in Console | Balance = 5  Limit = 0 | Pass |
|  | Check result of testPlayWithBalanceEqualsBet in Failure Trace | JUnit test should be no error and no failure | Fail |
|  | Check result of testPlayWithBalanceEqualsBet in Console | Balance = 0  Limit = 0 | Fail |

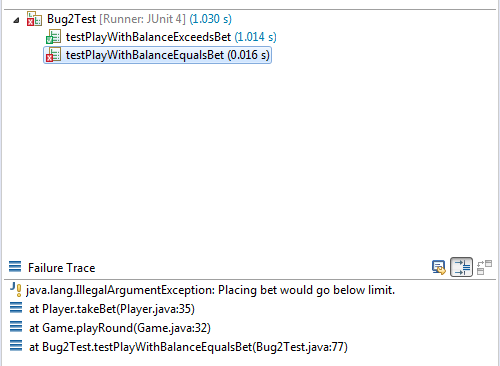
**Result**

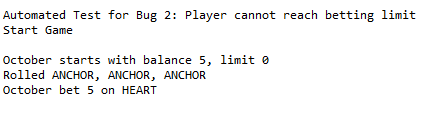
*1. When balance exceeds bet: PASS*

**

**

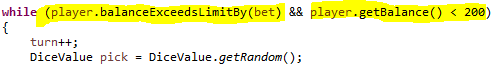
*2. When balance equals bet: FAIL*



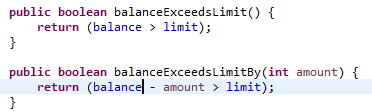


## Tracing

It could first be identified that the first trace is from the **Main.java** where the game is decided to continue or stop.



It is then identified that the bug problem is probably with the first condition, where the game must stop while the player exceeds the balance by bet amount. This method is included in the **Player.java** as:



So in this case, if the limit is 0, the game would continue as long as the balance - bet > 0. So in the case that the bet is 5 as in main file, when the balance is 5, the game would stop as "5 - 5 > 0" is false.

## Hypothesis

From the above guess, three hypotheses are proposed:

* **Hypothesis 1:** *setLimit()* method correctly assigns limit as zero.
* **Hypothesis 2*:*** *balanceExceedsLimitBy* does not include the minimum limit.
* **Hypothesis 3:** The game ends when the limit is reached.

The testing of the hypotheses is conducted by putting breakpoints at

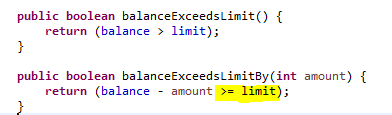
**return** (balance - amount > limit); (in Player.java)

The debugging shows the results for the hypotheses as below:

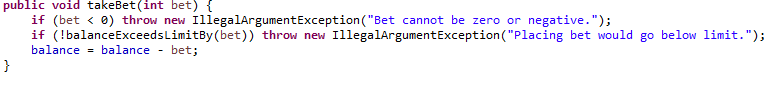
|  |  |
| --- | --- |
| **HYPOTHESIS** | **RESULT** |
| **Hypothesis 1:** *setLimit()* method correctly assigns limit as zero. | (1) is correct |
| **Hypothesis 2:** The game ends (negatively) when the limit is reached. | After run, the game as at balance = 5, limit = 0 so (2) is verified. |
| **Hypothesis 3*:*** *balanceExceedsLimitBy* does not include the minimum limit. | This is apparently true |

## Resolution

The bug can simply be fixed if putting an equal sign into the method as:



There is no risk in changing the method. As the method appears in **Main.java** and **Player.java,** changing may fix both logical errors and errors within the loop. However, as the method also appears in *takeBet* method, it needs to be considered. From the first version, if the balance is equal to bet (balanceExcessLimitBy(bet) = false), the exception would be called, and it is not correct. So if changing the code, this may fix potential problem.



## Result

### Bug2Test.java

| **BEFORE** | **AFTER** |
| --- | --- |
|  |  |
|  |  |

### Console from Main

| **BEFORE** | **AFTER** |
| --- | --- |
|  |  |

# Bug 3 - The DiceValues are the same for each game

## Replication

| **Test Name** | Test whether the dices are different in each game |
| --- | --- |
| **Use Case Tested:** | Crown and Anchor Game |
| **Test Description:** | Test whether the dices are different in each game |
| **Pre-conditions** | Bug 1 and 2 are fixed.  Run the program to simulate the game. |
| **Post-conditions** | The dice values should be different for different run |

|  | **TEST STEP** | **EXPECTED TEST RESULTS** | **RESULT** |
| --- | --- | --- | --- |
|  | Run Main.java with player details:  Player name = “Fred”  Balance = 100  Limit = 0 | Console opens and results for games are displayed in it. | Pass |
|  | Look at each individual line of rolls | There are at minimum two different rolls. | Fail |
|  | Repeat Steps 1-2. | Same as Steps 1-2. | Fail |

**Examples of bugs**

|  |  |
| --- | --- |
| Run 1 | Run 2 |

## Simplification

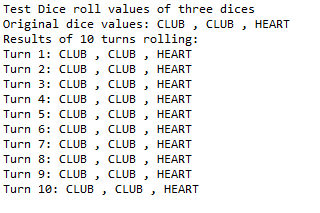
| **Test Name** | Test whether the dices are different in each game |
| --- | --- |
| **Use Case Tested:** | Automate the testing of errors in UAT Test 3 |
| **Test Description:** | Test whether the dices are different in each game |
| **Pre-conditions** | Run 3 dices only for 10 times. |
| **Post-conditions** | The dice values should be different for different run |

|  | **TEST STEP** | **EXPECTED TEST RESULTS** | **RESULT** |
| --- | --- | --- | --- |
|  | Run Bug3Test.java | Console opens and results for games are displayed in it. | Pass |
|  | Look at the result of 10 turns rolling | There are at minimum two different rolls. | Fail |
|  | Repeat the test 2 times | There are at minimum two different rolls. | Fail |

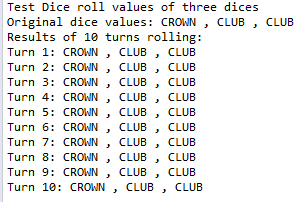
**Result**

As there is no assertion methods, all the tests have no failure trace.

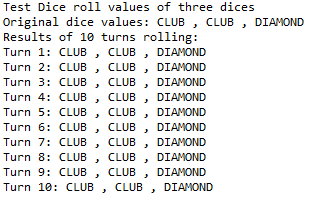
***Run 1: FAIL***

****

***Run 2: Fail***

**

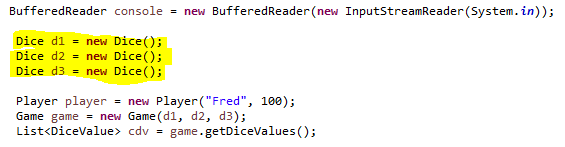
***Run 3: Fail***

**

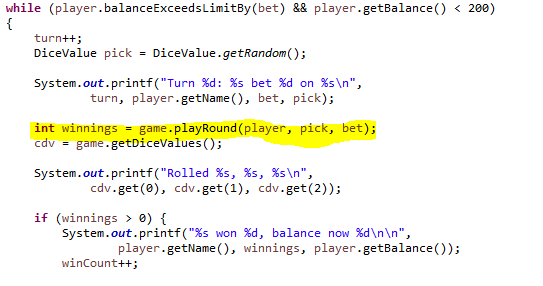
## Tracing

The methods that generate a new DiceValue for each dice occur in the **Main.java**, **Game.java** and **Dice.java**.

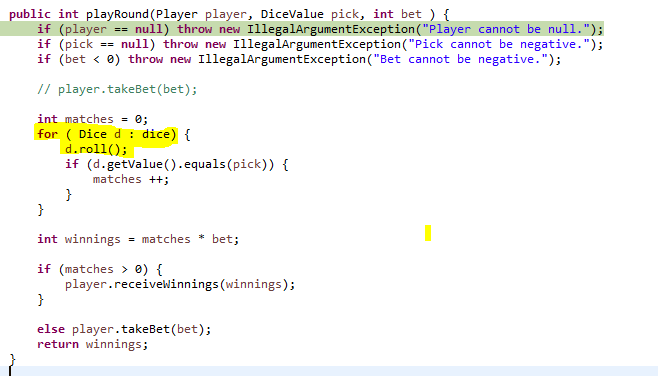
Starting from the **Main.java**, three dices of the game are called:



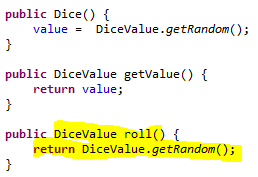
The **Main.java** also implies that the assigning of dice occurs in the *playRound* method of Game.java.



The rolling of dice occurs in *playRound* method as:



The rolling in **Dice.java** occurs as:



So from this, it could be identified that:

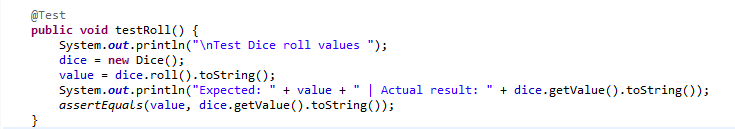
* The *value* variable in **Dice.java** is invariant throughout the game as it is just assigned in the constructor of Dice, but no changing occurs further in the file.
* The compare of the pick is with the *d.getValue()*, which is the value of the Dice originally, rather than the value of the dice after rolling.
* The *roll()* method, although assigns a new DiceValue to the dice, the comparison of DiceValue is with the *getValue()* method, which returns the invariant variable *value.*
* The three dices are set at the beginning of the game in **Main.java**, with the given *value* at the setting of constructor. If the *value* variable in Dice(), the dices may be reused again throughout the game.

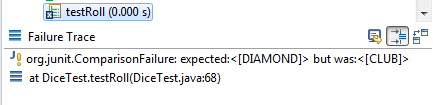
## Hypotheses

Given the matches are correct (as demonstrated in Bug 1), the hypotheses are:

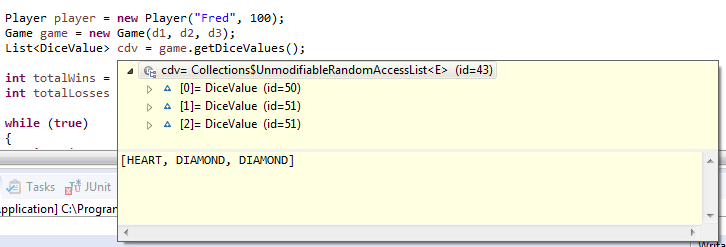
* **Hypothesis 1**: the *roll()* method does not change the value of DiceValue (incorrect).
* **Hypothesis 2:** the result of each turn is decided by comparing the pick with the *value* of the dice (correct).
* **Hypothesis 3:** three dices are assigned with each value and reused throughout the game (incorrect).

To test the first hypothesis, a unit test (**DiceTest.java**) is set up to examine the method of getRandom method. Result of the test shows that the *roll()* method is incorrect.

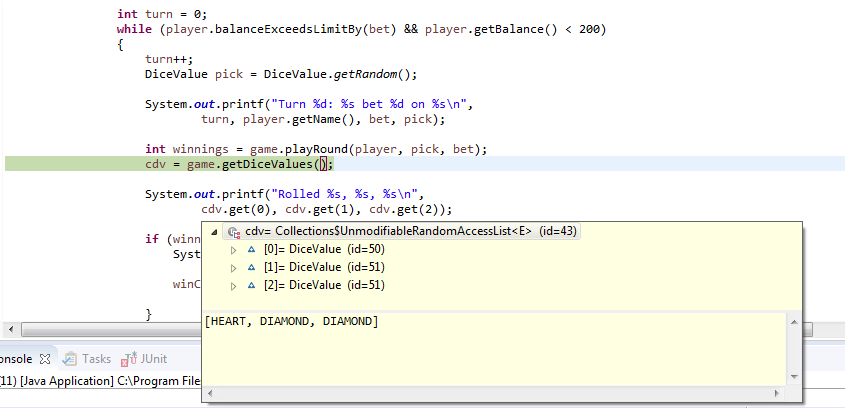




To test the second and third hypotheses, one breakpoint is put within the loop to monitor the changing state of DiceValue. It is verified as at the start of the game, three dice values in *cdv* are:

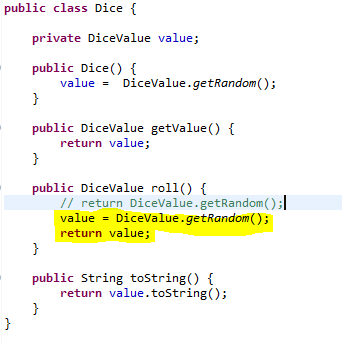


but after rolling, three dices value are still the same:



## Resolution

As stated in Tracing document, the issue is while *roll()* method returns new value, the game just compare and display the pick value with the first *value* of the dice from the constructor. Therefore, the solution is to assign new value for variable *value* after the roll.



There is also no risk for the resolution, as the *roll()* method is just called in **Dice.java** and **Game.java**. The game is still decided with the comparison of *pick* with *d.getValue()*, which returns value.

## Result

### Bug3Test

Different results are showed

| **BEFORE** | **AFTER** |
| --- | --- |
|  |  |

### Console from Main

After the change, there is no error as showed in the console.

| **BEFORE** | **AFTER** |
| --- | --- |
|  |  |

# Bug 4 - SPADE is ever rolled or guessed

## Replication

| **Test Name** | Test whether the SPADE is guessed or rolled in each game |
| --- | --- |
| **Use Case Tested:** | Crown and Anchor Game |
| **Test Description:** | Test whether SPADE is guessed by player or never appears in the game. |
| **Pre-conditions** | Bug 1, 2, 3 are fixed  Run the program to simulate the game. |
| **Post-conditions** | The results should have some SPADE. |

|  | **TEST STEP** | **EXPECTED TEST RESULTS** | **RESULT** |
| --- | --- | --- | --- |
|  | Run Main.java with player details:  Player name = “Fred”  Balance = 100  Limit = 0 | Console opens and results for games are displayed in it. | Pass |
|  | Look at each individual line of rolls | There are at minimum two SPADE. | Fail |
|  | Repeat Steps 1-2. | Same as Steps 1-2. | Fail |

**Examples of bugs**

|  |  |
| --- | --- |
| **Run 1** | **Run 2** |

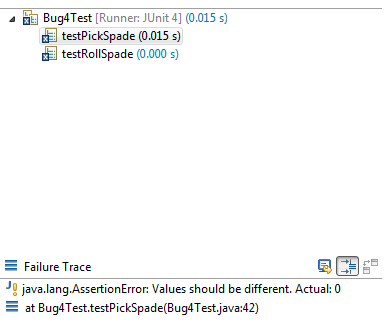
## Simplification

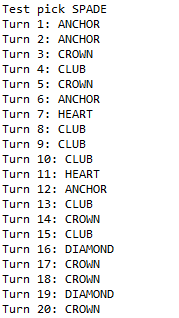
| **Test Name** | Test whether the SPADE is guessed or rolled in each game |
| --- | --- |
| **Use Case Tested:** | Automate the testing of errors in UAT Test 4 |
| **Test Description:** | Test whether SPADE is guessed by player or never appears in the game. |
| **Pre-conditions** | Bug 1, 2, 3 are fixed.  Run 20 turns of random pick and rolls. |
| **Post-conditions** | The results should have some SPADEs. |

|  | **TEST STEP** | **EXPECTED TEST RESULTS** | **RESULT** |
| --- | --- | --- | --- |
|  | Run Bug4Test.java | Console opens and results for games are displayed in it. | Pass |
|  | Check result of testPickSpade in Failure Trace | JUnit test should be no error and no failure | Fail |
|  | Check result of testPickSpade in Console | Some Spades | Fail |
|  | Check result of testRollSpade in Failure Trace | JUnit test should be no error and no failure | Fail |
|  | Check result of testRollSpadeh in Console | Some Spades | Fail |

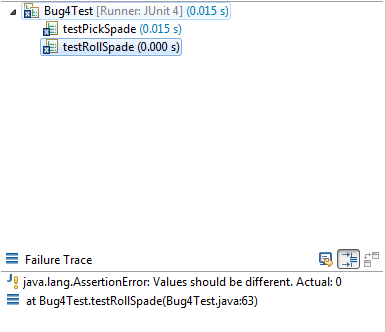
Results

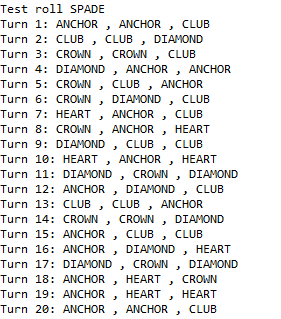
*1. Pick Space: FAIL*

**

**

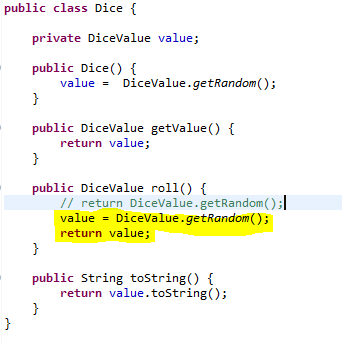
*2. Roll Space: FAIL*

**

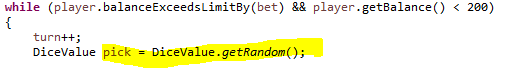
**

## Tracing

From Section 3 above, it is noticed that the DiceValue is determined by the *roll()* method:



It is also noticed that the pick value is also from *getRandom()* method:



As the method call the *getRandom()* method from **DiceValue.java**, the problem is probably from there:



It is then noticed that the value of RANDOM is from the Math class Random, but the *nextInt()* calls the value of enum with the ordinal value as SPADE. As from the enum declaration, the *nextInt()* method will call randomly a number from ordinal value 0 to ordinal value of SPADE as 5. So SPADE never appears in any Random method.



## Hypotheses

The hypotheses are:

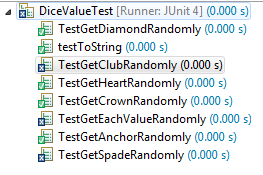
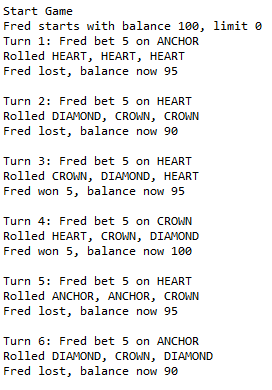
* (1): As both pick and roll are decided randomly, this cause the issue that SPADE never appears in pick and roll.
* (2): SPADE is never called due to the failure of logic in *nextIn()* of *getRandom()* method.

The testing of the hypothesis can also be done by replicating the bugs if changing the codes:

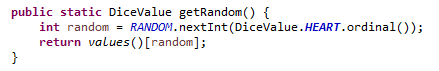
The first is changing the position of CLUB and SPACE in the enum declaration, so the ordinal value of SPACE will be 4. If the hypotheses are correct, CLUB and SPACE will not appear.



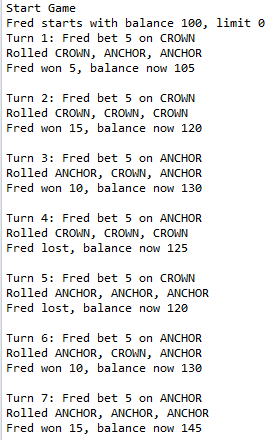
The console shows that no SPADE or CLUB. The unit test also confirms that there are errors.



The second is to change the SPADE in the *nextInt* into HEART (ordinal value = 2). If the hypotheses are correct, only CROWN or ANCHOR will appear.

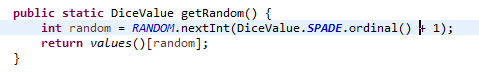


The console confirms the error:



## Resolution

The resolution here is to increase the value in *nextInt* up to 1 ordinal value, so that SPADE will be covered in the range:



There is also no risk at changing according to this. The change will improve the game logic. However, the inclusion of 1 DiceValue may change the win rate of the game (see Bug 5).

## Result

### Bug4Test

| **BEFORE** | **AFTER** |
| --- | --- |
|  |  |
| **Roll SPADE** | **Roll SPADE** |
| **Pick SPADE** | **Pick SPADE** |

### Console from Main

As the result, there are some SPADEs in guessing and in dices.

| **BEFORE** | **AFTER** |
| --- | --- |
|  |  |

# Bug 5 - Odds of game are incorrect

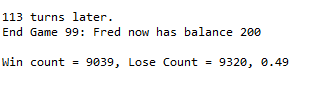
## Replication

| **Test Name** | Test whether the winning odds is around 42%. |
| --- | --- |
| **Use Case Tested:** | Crown and Anchor Game |
| **Test Description:** | Test that the winning ratio is correct at around 42% |
| **Pre-conditions** | Bug 1, 2, 3 have been fixed  Run the program to simulate the game. |
| **Post-conditions** | The winning ratio is around 42%. |

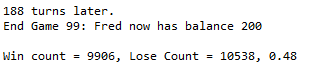
|  | **TEST STEP** | **EXPECTED TEST RESULTS** | **RESULT** |
| --- | --- | --- | --- |
|  | Undo the changing of SPADE as in Bug 4 | Only bugs 1, 2, 3 are fixed. | Pass |
|  | Run Main.java | Console opens and results for games are displayed in it. | Pass |
|  | Look at the win count line | There should be ratio of 0.42 | Fail (ratio =0.48) |
|  | Repeat Steps 2-3. | Same as Steps 2-3. | Fail |

**Examples of bugs**

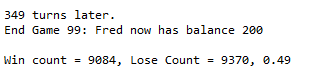
Run 1: FAIL



Run 2: FAIL



Run 3: FAIL



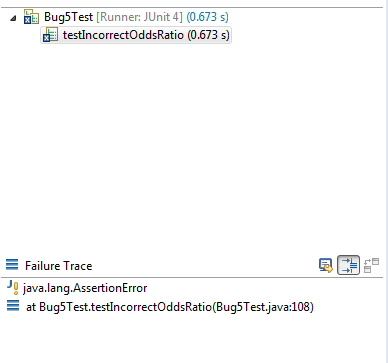
## Simplification

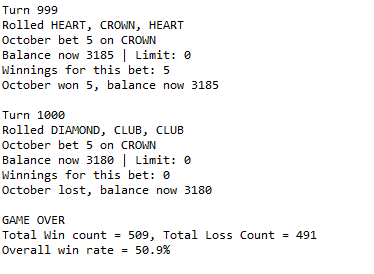
| **Test Name** | Test whether the winning odds is around 42%. |
| --- | --- |
| **Use Case Tested:** | Automate the testing of errors in UAT Test 5 |
| **Test Description:** | Test that the winning ratio is correct at around 42% |
| **Pre-conditions** | Bug 1, 2, 3 have been fixed.  New player "October" with initial balance of $5000, and number of turns 1000 |
| **Post-conditions** | The winning ratio is around 42%. |

|  | **TEST STEP** | **EXPECTED TEST RESULTS** | **RESULT** |
| --- | --- | --- | --- |
|  | Run Bug4Test.java | Console opens and results for games are displayed in it. | Pass |
|  | Check win ratio in Console | Should be between 41 - 43% | Fail |
|  | Check Failure Trace | JUnit should show no error or failure | Fail |
|  | Rerun the test 2 times | Same like step 2-3 | Fail |

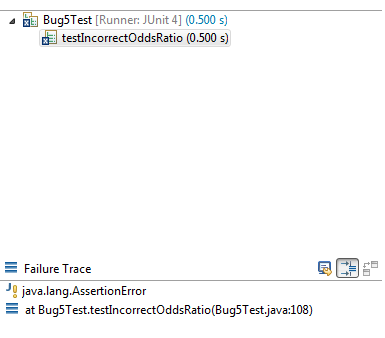
***Result***

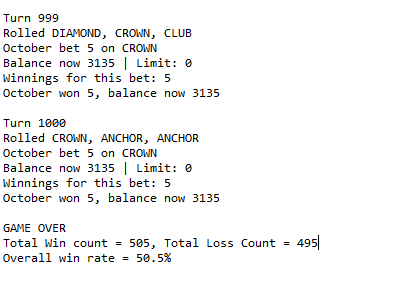
**Run 1: FAIL**



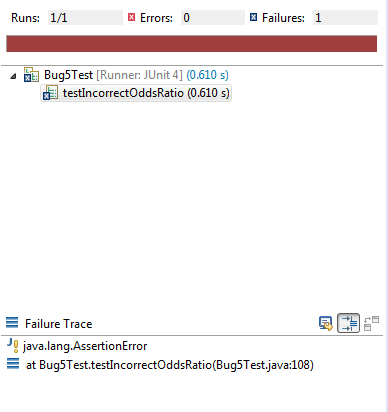


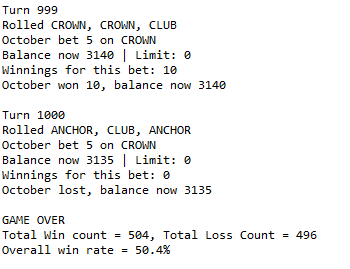
**Run 2: FAIL**





**Run 3: FAIL**





## Tracing

The tracing of this issue should first start at considering the winning ratio of *n* values in the game. There are possible combinations of 3 dices. The player will lose if the pick is not the same as any of the dices, which means the losing pick lays within the domain of . The winning ratio is calculated as:

So if there are 6 DiceValues, the winning ratio will be approximately 0.42:

If there are 5 DiceValues (excluding SPADE), the winning ratio will raise up to 0.48 or 0.49 as in the bug example below.

## Hypothesis

The root of this bug is the same as bug 4 above - no SPADE in the game. So by adding the SPADE, the winning will be resolved.

## Resolution

The root of this bug is the same as bug 4 above - no SPADE in the game. So by adding the SPADE, the winning will be resolved.

## Result

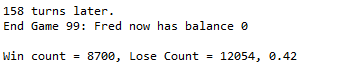
### Bug5Test

| **BEFORE** | **AFTER** |
| --- | --- |
|  |  |
|  |  |

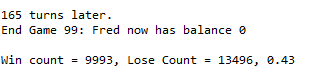
### Main Console

After changing the line in *getRandom()* method, the winning ratio is correct.

Run 1:



Run 2



Run 3

