Table of Contents

Introduction:

This document defines the bug present in the Crown and Anchor game program, its identification, its hypothesis of presence in the program and its resolution. This document describes the various user acceptance test and junit test to figure out the bug and tests its resolution after debugging process. This document looks into the bugs that are described in bug report below.

Bug Report:

Bug 1: Incorrect pay-out on win

Game doesn't seem to pay the player with the correct amount after win. The balance doesn't seem to increase when the user wins but only decreases when the user loses.

Bug 2: Player cannot reach the betting limit

Game doesn't let the user reach betting limit which is 0. The game ends abruptly even when the user has the balance 5.

Bug 3: Game overall odds

Odds in the game do not appear to be correct. The game should be 8% bias to the house which means win/ (win+loose) ratio should be approx. 0.42 which isn't the case in this scenario.

Bug 4: Dice seem to roll the same after first roll

Dice seem to always roll the same after first roll. The dice values are always same for each round of a game.

Bug 5: "Spade" doesn't appear in roll.

Dice roll seems not to show SPADE. It means that symbol doesn't appear on the dice roll.

User tests (in UAT format):

UAT1: Bug 1: Incorrect pay-out on win

Scenario –Reports on Bug

Scenario Description

This test is designed to figure out the bugs from the given "Crown & Anchor" game.
Each test will represent the single bug found in the program.

Version Control

Version #	Date	Author	Description
0.1	07/10/2017	Bijaya Raj Basnet	Initial Draft

Test Scripts

The following scripts will cover this scenario:

Bug – Payment is incorrect on winnings.

Test Components/Requirements

User plays the Crown and Anchor game

Script: Bug – Payment is incorrect on winnings

Script Description

This script looks into the bug in which when the player wins the game, his/her balance doesn't increase.

Testing Requirements

- This test follows the following rules of the game:
- A player bets on a particular symbol and wins if one or more symbol appears on the three dice that is rolled and if none appears the player loses the game.
- The win amount of the player will be bet amount times the number of symbol matches to that the user has placed the bet on.

Pre-conditions

- The game should be running.
- The user balance should be greater than minimum bet.
- The user should place a bet on one of the symbol.

Required Data

- A Valid user "Fred"
- Starting Balance "100"
- Three dices "d1,d2,d3"
- A bet "Crown"

Post-conditions

- Dices returned determine the post conditions. Following are some of the cases for post conditions.

Cases

- No matches. The balance should decrease by -5
- 1 match, balance should increase by 5, balance should be previous balance +5
- 2 matches, balance should increase by 10, balance should be previous balance +10
- 3 matches, balance should increase by 15, balance should be previous balance +15

Script Steps

Step #	Test Action	Expected Results	Pass/ Fail
1	Start new game	A new valid game exists	P
2	Pick Crown, Bet 5	3 values of dice and a result	P
3	Result check	Confirm winnings amount is correct only on winnings. 0 crowns – Winnings = -5 1 crown – Winnings = 5 2 crowns – Winnings = 10 3 crowns – Winnings = 15	P
4	Check player balance	Balance is adjusted as per win	P

Test Execution

Date/Time	Tester	Test ID	Test Phase	Status
07/10/2017 10pm	Bijaya	11636540	UAT1-Test1	Fail
14/10/2017 9:10 pm	Bijaya	11636540	UAT2-Test2	Pass

UAT2: Bug 2: Player cannot reach the betting limit

Scenario – Reports on Bug

Scenario Description

This test is designed to figure out the bugs from the given "Crown & Anchor" game.
Each test will represent the single bug found in the program.

Version Control

Version #	Date	Author	Description
0.1	07/10/2017	Bijaya Raj Basnet	Initial Draft

Test Scripts

The following scripts will cover this scenario:
Bug – Player Balance isn't 0 at the end of game

Test Components/Requirements

User plays the Crown and Anchor game

Script: Bug – Player Balance isn't 0 at the end of game Script

Description

This test script looks into the fact that the game ends before player balance is 0.

Testing Requirements

- A player bets on a particular symbol and wins if one or more symbol appears on the three dice that is rolled and if none appears the player loses the game.
- The win amount of the player will be bet amount times the number of symbol matches to that the user has placed the bet on.
- The player keeps on betting until his balance is less than the betting limit.

Pre-conditions

- The game should be running.
- The user balance should be greater than minimum bet.
- The user should place a bet on one of the symbol.

Required Data

- A Valid user "Fred"
- Starting Balance "100"
- Three dices "d1,d2,d3"
- A bet "Crown"

Post-conditions

- The player balance should be equal to 0.

Script Steps

Step #	Test Action	Expected Results	Pass/ Fail
1	Start new game	The game executes	P
2	Pick Crown, Bet 5	3 values of dice and a result	P
3	Check result	Confirm winnings amount is correct based on dice values as follows: 0 crowns – Winnings = -5 1 crown – Winnings = 5 2 crowns – Winnings = 10 3 crowns – Winnings = 15	P
4	Check player balance	Balance is adjusted as per win	P
5	Repeat steps 4 & 5 until game play is ended	Confirm player balance is zero	P

Test Execution

Date/Time	Tester	Test ID	Test Phase	Status
07/10/2017 10:30pm	Bijaya	11636540	UAT2-Test1	Fail
14/10/2017 10:50pm	Bijaya	11636540	UAT2-Test2	Pass

UAT3: Bug 3: Incorrect game overall odds

Scenario –Reports on Bug

Scenario Description

This test is designed to figure out the bugs from the given "Crown & Anchor" game. Each test will represent the single bug found in the program.

Version Control

Version #	Date	Author	Description
0.1	07/10/2017	Bijaya Raj Basnet	Initial Draft

Test Scripts

The following scripts will cover this scenario:

- Bug - Odds in the game do not appear to be correct.

Test Components/Requirements

- User plays the Crown and Anchor game

Script: Bug – Win ratio do not appear to be correct.

Script Description

This script tests the win and loss ratio. The game states that it should be 8% bias to the house. This means that the win ratio should be 0.42% which is not the case in this game.

Testing Requirements

- A player bets on a particular symbol and wins if one or more symbol appears on the three dice that is rolled and if none appears the player loses the game.
- The win amount of the player will be bet amount times the number of symbol matches to that the user has placed the bet on.
- The overall result should be that the game should be 8% favor of the house. The player win ratio should be 42% (+-3%)that means the player should loose 58 times out of 100.

Pre-conditions

- A user must have registered for gameplay and have a positive balance greater than the minimum bet.
- A game must be initialized with 3 dice.

Required Data

- A Valid user "Fred"
- Starting Balance "100"
- Three dices "d1,d2,d3"
- A bet "Crown"
- No of plays=100

Post-conditions

- The win ratio of the player should be 42%, that means the player should lose 58 times out of 100.

Script Steps

Step #	Test Action	Expected Results	Pass/ Fail
1	Start game	Game starts	P
2	Player bets on $5 on crown	3 values of dice and a result	P
3	Check result	Confirm winnings amount is correct based on dice values as follows: 0 crowns – Winnings = -5 1 crown – Winnings = 5 2 crowns – Winnings = 10 3 crowns – Winnings = 15	P
4	Play until game ends	Games ends	P
6	Check win rate	Win rate of 42% +/- 3%	P

Test Execution

Date/Time	Tester	Test ID	Test Phase	Status
07/10/2017 10:45pm	Bijaya	11636540	UAT3-Test1	Fail
14/10/2017 11:10	Bijaya	11636540	UAT3-Test2	Pass

UAT4: Bug 4: Dice seem to roll the same after first roll

Scenario –Reports on Bug

Scenario Description
This test is designed to figure out the bugs from the given "Crown & Anchor" game.
Each test will represent the single bug found in the program.

Version Control

Version #	Date	Author	Description
0.1	07/10/2017	Bijaya Raj Basnet	Initial Draft

Test Scripts
The following scripts will cover this scenario:

- Bug – Dice rolls same for each round

Test Components/Requirements

- User plays the Crown and Anchor game

Script: Bug – Dice rolls same for each round

Script Description
This script looks into the bug that the same symbols are rolled after the first round until the end of game.

Testing Requirements
Each turn of play should give different symbols of the three dices that are rolled.

Pre-conditions

- The game should be running.
- The user balance should be greater than minimum bet.
- The user should place a bet on one of the symbol.

Required Data

- A Valid user "Fred"
- Starting Balance "100"
- Three dices "d1,d2,d3"
- A bet "Crown"

Post-conditions
Each round should provide different symbols on the roll of the dices.

Script Steps

Step #	Test Action	Expected Results	Pass/ Fail
2	Start new game	New game starts	P
3	Pick "Crown" bet 5	3 values of dice and a result	P
4	Check the symbols on the roll of dice	Symbols should be random on each round	P

Test Execution

Date/Time	Tester	Test ID	Test Phase	Status

07/10/2017 11:15pm	Bijaya	11636540	UAT4-Test1	Fail
14/10/2017 11:20pm	Bijaya	11636540	UAT4-Test2	Pass

UAT5: Bug 5: "Spade" doesn't appear in roll

Scenario –Reports on Bug

Scenario Description

This test is designed to figure out the bugs from the given "Crown & Anchor" game. Each test will represent the single bug found in the program.

Version Control

Version #	Date	Author	Description
0.1	07/10/2017	Bijaya Raj Basnet	Initial Draft

Test Scripts

The following scripts will cover this scenario:

- Bug – Dice rolls same for each round

Test Components/Requirements

- User plays the Crown and Anchor game

Script: Bug – Spade doesn't appear on Dice roll

Script Description

This script looks into the bug that the "Spade" doesn't appear on dice roll.

Testing Requirements

- Dice roll should display "Spade" at least once in any round of play.

Pre-conditions

- The game should be running.
- The user balance should be greater than minimum bet.
- The user should place a bet on one of the symbol.

Required Data

- A Valid user "Fred"
- Starting Balance "100"
- Three dices "d1,d2,d3"
- A bet "Spade"

Post-conditions

- Each round should provide different symbols on the roll of the dices with anyone of the round containing spade.

Script Steps

Step #	Test Action	Expected Results	Pass/ Fail
2	Start new game	New game starts	P
3	Pick “Spade” bet 5	3 values of dice and a result	P
4	Check the symbols on the roll of dice	Spade should appear at least once	P

Test Execution

Date/Time	Tester	Test ID	Test Phase	Status
07/10/2017 11:30pm	Bijaya	11636540	UAT5-Test1	Fail
14/10/2017 11:30pm	Bijaya	11636540	UAT5-Test2	Pass

Output from your automated test demonstrating the buggy behaviour:

Bug 1: Incorrect Payout on win

Following is the screenshot of the output of the junit test for the payout test. So to carry out his test, I created a player “Fred” with starting balance 100 and bet 5 on “CROWN”.

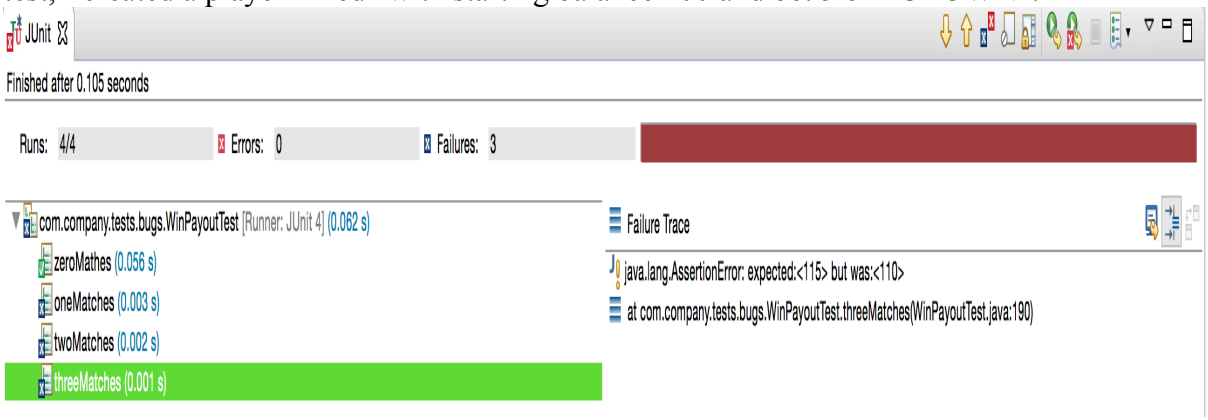


Fig: screenshot showing the failure on one match

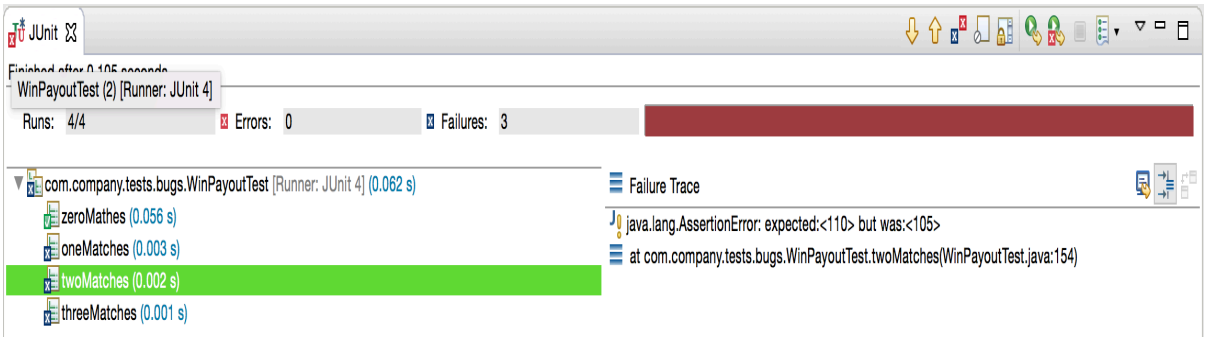


Fig : screenshot showing the failure on two matches

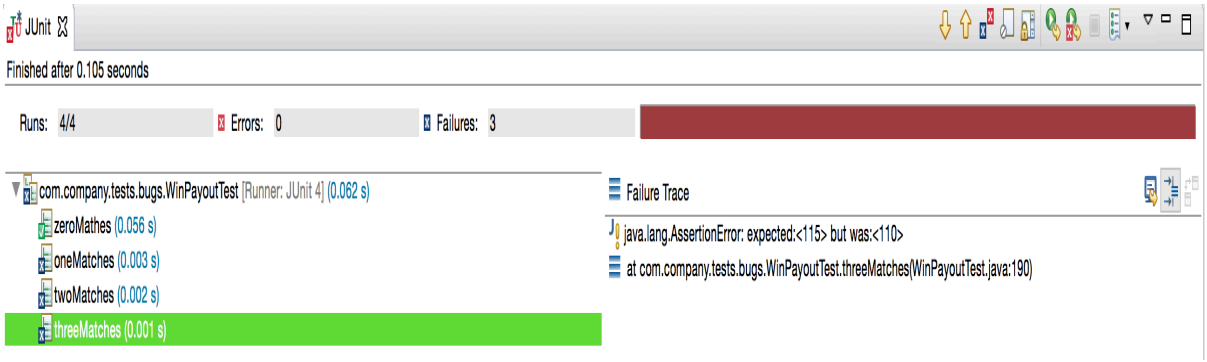


Fig : screenshot showing the failure on two matches

The above screenshot shows there is a failure on one match, two match and three match while one match passes because we have only bug on returning the bet which is only done whilst wining.

Bug 2: Player cannot reach betting limit

This junit test tests the ending balance of the player. As per the rule of the game the game ends only when the balance of the player is less than the bet balance which is 5. Even if the balance of the player is equivalent to the bet balance then game shouldn't end.

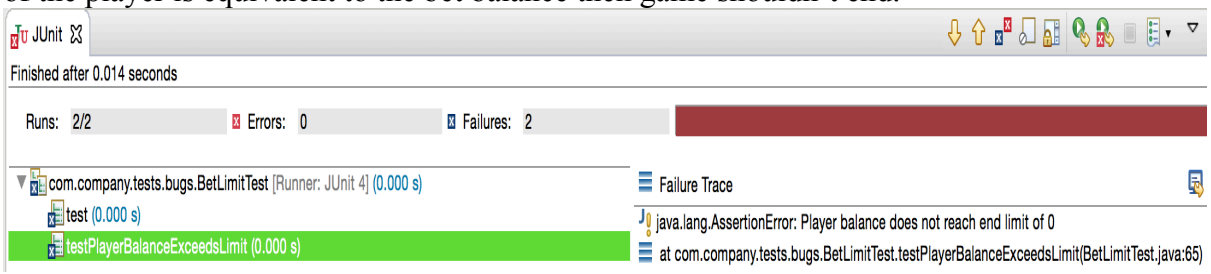


Fig: screenshot confirming the balance isn't 0

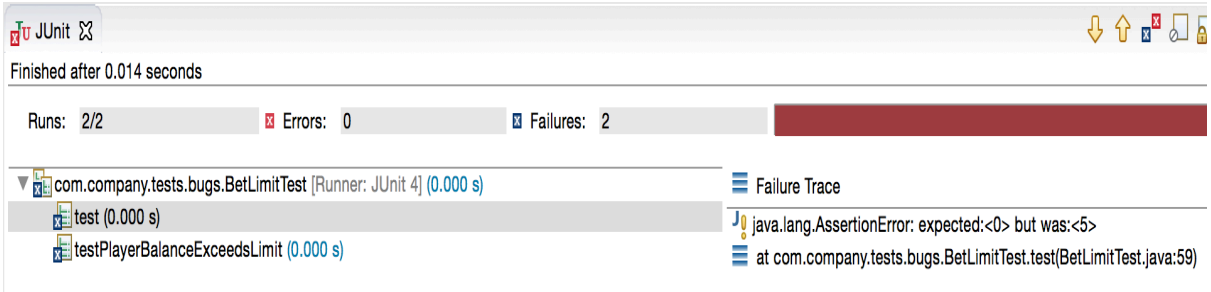


Fig: screenshot showing the failure of the test.

In the above screenshot we can see that the expected result was 0 but the actual result was 5. This is why the unit test has failed.

Bug 3: Incorrect game overall odds

This test tests the fact if the win ratio is approx. 0.42. As per the rule of the game the win ratio should only be in between 0.42 to 0.45. Following is the result of the test.

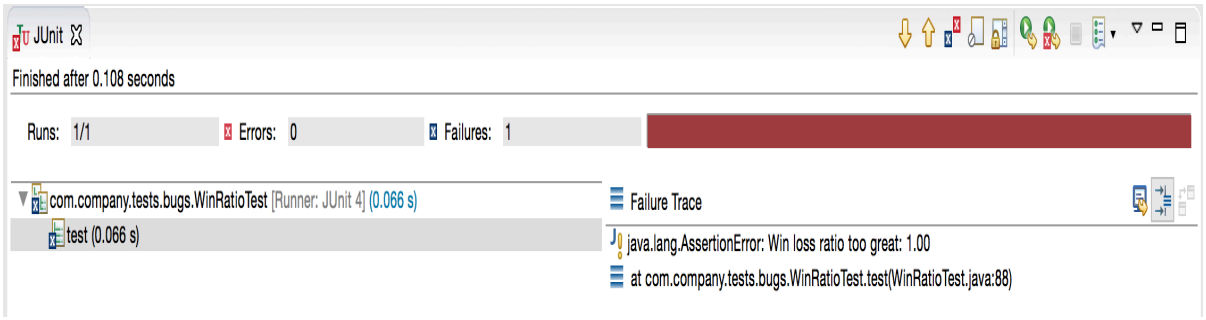
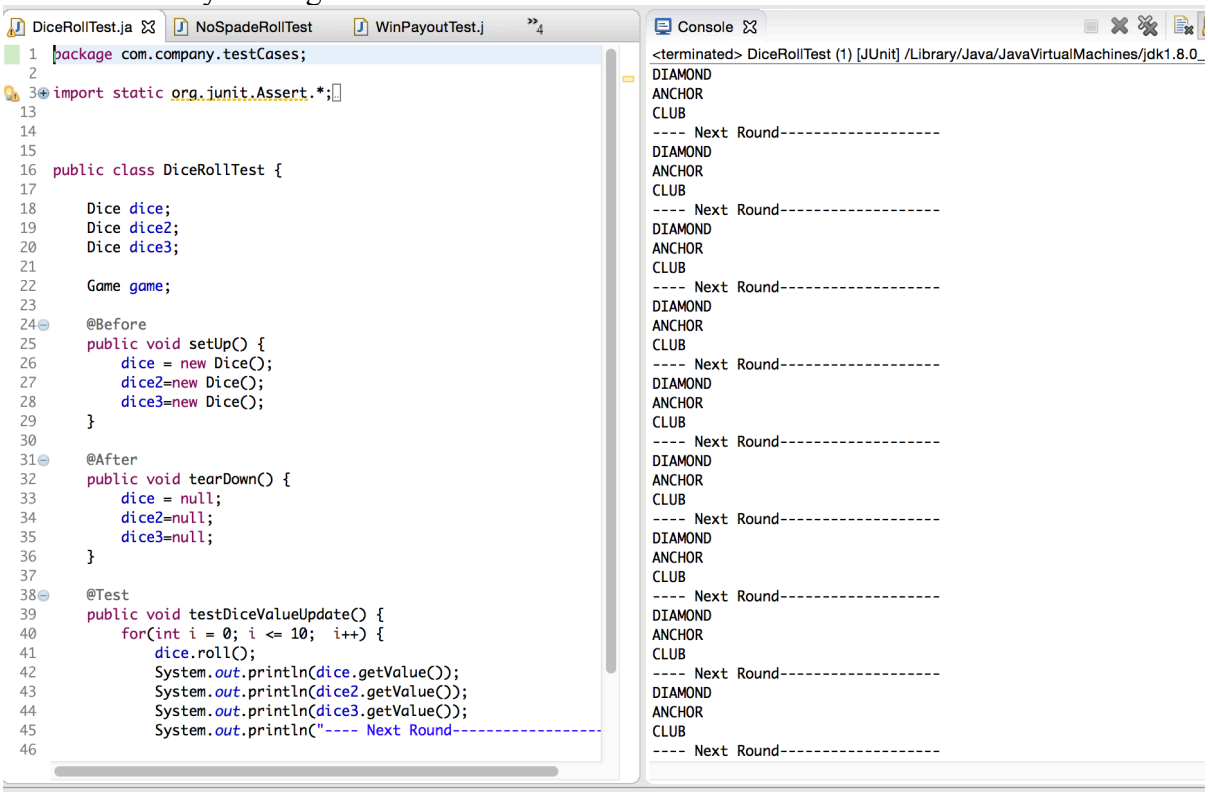


Fig: Screenshot showing the failure of the test.
Above screen shot shows the failure of the test. The failure trace shows the win loss ratio was too great than it was supposed to be. This is why the test has failed.

Bug 4: Dice seem to always roll the same after first roll.

This test is used to test the roll of dice. This test looks into the bug that seems to roll the same after the first roll. In this automated test I have used System.out.Print function to look into the bug. I have simply printed the dice roll for first 10 rounds using the unit test and looked in the console to verify the bug.




```
1  package com.company.testCases;
2
3  import static org.junit.Assert.*;
13
14
15
16 public class DiceRollTest {
17
18     Dice dice;
19     Dice dice2;
20     Dice dice3;
21
22     Game game;
23
24     @Before
25     public void setUp() {
26         dice = new Dice();
27         dice2=new Dice();
28         dice3=new Dice();
29     }
30
31     @After
32     public void tearDown() {
33         dice = null;
34         dice2=null;
35         dice3=null;
36     }
37
38     @Test
39     public void testDiceValueUpdate() {
40         for(int i = 0; i <= 10;  i++) {
41             dice.roll();
42             System.out.println(dice.getValue());
43             System.out.println(dice2.getValue());
44             System.out.println(dice3.getValue());
45             System.out.println("---- Next Round----------
46
```

In the above screen shot, we can see that every round of the play has the same dice roll as the first one.

Bug 5: “SPADE” doesn’t appear in roll.

This test tests for the bug that “Spade” doesn’t occur in the values of the dice rolled. The program tests in such a way that if fails if the dice roll results doesn’t contain “Spade”.

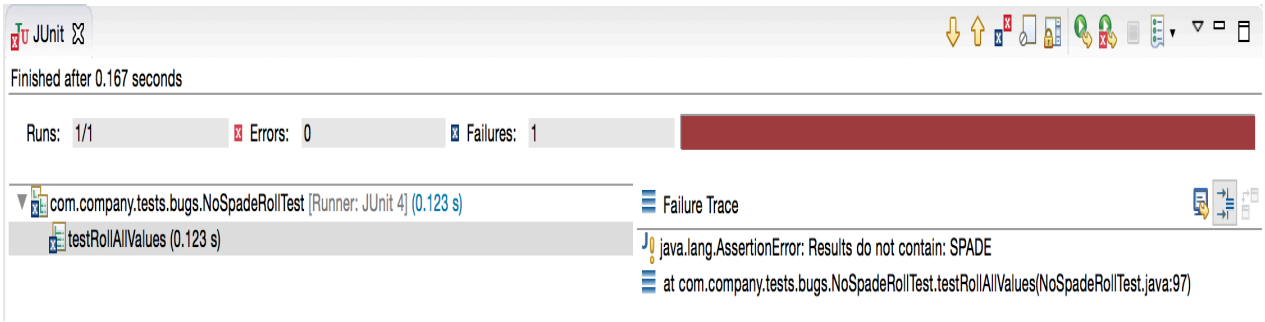


Fig: Screenshot showing the failure of the test.
The above screen shot shows that the test has failed, meaning that the dice roll values doesn't contain the "Spade". The "Failure Trace" section also shows the assertion error that "Result do not contain "Spade"".

Debugging log recording the succession of hypotheses, tests, and results:

Debugging BUG 1: Incorrect Pay

Hypothesis 1:

Hypothesis	The winnings amount must be calculated incorrectly.
Prediction	All the wining amounts must be incorrect hence resulting in paying out incorrect amount.
Experiment	Play the game and observe the winning amount
Observation	correct winning amount was calculated.
Conclusion	Hypothesis was wrong hence the wining calculation in the game were correct.

Hypothesis 2:

Hypothesis	Balance isn't calculated correctly.
Prediction	Balance calculation after adding wining isn't correct.
Experiment	Play the game and observe
Observation	Observation: Balance decreases on loss, balance is same on one win but then balance increases incorrectly in two and three wins.
Conclusion	Balance calculation after win is incorrect, further investigation is required.

Hypothesis 3:

Hypothesis	Bet amount isn't returned to the player.

Prediction	Bet amount isn't refunded back to the player after the win. Only the win amount is returned.
Experiment	Play the game and observe
Observation	The balance is always less by the amount of bet placed.
Conclusion	There is issue in refunding bet. The issue is in Game.java , Game.playRound() method.

After the hypothesis were developed, the junit test was developed and tested to validate the issue and its resolution.

Resolving Bug1- Incorrect Payout

There was a simple resolution of this bug. It is simple that the bet is only taken when the player loses the game but then returned when the player wins the game which is done as follow:

The bug was in Game.playround() method of Game.java , in following part of code.

```
        if (matches > 0) {
                        player.receiveWinnings(winnings);
                }
    return winnings;
                }
```

The received winnings is only the winnings but no bet is returned. Thus Following is the correct bug resolution.

```
        if (matches > 0) {
                        player.receiveWinnings(winnings + bet);
                }
    return winnings;
                }
```

Debugging Bug 2 - Player cannot reach betting limit

Hypothesis1:

Hypothesis	Player.balanceExceedsLimitBy(int amount) has an issue resulting in player not reaching limit
Prediction	Game ends even when the player balance is greater than bet limit.
Experiment	Run game and observe: bet amount 5, limit 0.
Observation	Final balance is 5 which is still equal to bet amount and greater than bet limit which is 0.
Conclusion	There is an issuw with balanceExceedsLimitBy(int amount) method in player class.

A junit test (automated test) was developed and checked which gave back an error which means that there is the bug in the method.

After this run an automated test was developed that plays rounds and loses until the nominated method returns false. Then the remaining balance is checked. This demonstrates the bug. A further simplification is possible though.

Resolving Bug 2: Player cannot reach betting limit

There is the simple resolution of this bug. balanceExceedsLimitBy(int amount) method was inspected and following error was found. It is because we are only checking the balance was greater than limit but not if it was equal to limit.

```
public boolean balanceExceedsLimitBy(int amount) {
    return (balance - amount > limit);
    }
```

This was resolved as follow:

```
public boolean balanceExceedsLimitBy(int amount) {
    return (balance - amount >= limit);
    }
```

After making these changes to the method, the automated test was run and it passed. The UAT also passed.

Debugging Bug 3- Game overall odds appear incorrect

Hypothesis 1:

Hypothesis	Win/loss ratio is different only in small round of game
Prediction	Playing the large number of rounds might converge the win and loss ratio.
Experiment	1000 round of game are run in unit test.
Observation	Win/loss ratio converges to zero or infinity
Conclusion	Wrong hypothesis, win ratio and loss ratio doesn't tend to match

Even though the hypothesis is wrong, I found out that the dices doesn't tend to change after the first round and spade doesn't appear to roll in the dice roll. These bug will be looked into bug 4 and 5 respectively and returned back to this bug after debugging these two issues in dice roll.

Debugging Bug 4- Dice seem to always roll the same after first roll.

Hypothesis 1:

Hypothesis	Issue with Issue with DiceValue.getRandom ().It is not returning different values.
Prediction	The values for the dice roll will be same for a single game. It doesn't change.
Experiment	Run game and observe the three dices.
Observation	Dice did indeed rolled but there was no SPADE which will also a bug which will be dealt in bug 4.
Conclusion	There is issue with DiceValue.getRandom ()method of dice class but not for the bug we are working into.

Hypothesis 2:

Hypothesis	Issue with Dice.getValue().It is not receiving different values.

Prediction	The values for the dice roll will be same for a single game. It doesn't change.
Experiment	Run game and observe the three dices.
Observation	All the dice values are same and doesn't change for the whole game.
Conclusion	Dice.getValue() is the main problem for this bug.

Resolving Bug 4: Dice seem to always roll the same after first roll.

On inspecting dice.getValue(), it is found out that the "value" has been initialized in the constructor but not called in the roll which is the bug. A simple change in the code eliminates the bug which is done as follow:

Previous code with bug:

```
Public DiceValue getValue() {
        return value;
    }
```

Bug free code:

```
Public DiceValue getValue() {
        this.value=DiceValue.getRandom();
        return value:
    }
```

We find the automated test for this method now passes, as does the UAT test.

Debugging Bug 5 – "SPADE" doesn't appear in roll.

Hypothesis 1:

Hypothesis	Issue with DiceValue.getRandom ().It is not returning SPADE.
Prediction	The values for the dice roll will be same for a single game. It doesn't change.
Experiment	Run game and observe the three dices.
Observation	Dice did indeed rolled but there was no SPADE which will also a bug which will be dealt in bug 4.
Conclusion	There is issue getRandom ().method of DiceValue class.

The Dicevalue.getRandom() method uses **RANDOM.nextInt(DiceValue.SPADE.ordinal().** This excludes "SPADE" from being rolled as nextINT(n), that returns random between 0 and n where o being inclusive and n being exclusive.

Resolving Bug 5: "SPADE" doesn't appear in roll.

There is a simple bug in the DiceValue class, getRandom method. The code has been changed as follow to remove the bug.

Previous code with bug:

```
public static DiceValue getRandom() {
        int random=RANDOM.nextInt(DiceValue.SPADE.ordinal());
        return values()[random];
    }
```

Bug free code:

```
public static DiceValue getRandom() {
        int random = RANDOM.nextInt(VALUE_REPR_MAP.size());
        return values()[random];
```

}

Here we just passed the size so that "SPADE" gets inclusive to the roll. The unit test was carried out to test this bug and it passed the test.

Revisiting Bug 3 - Game overall odds appear incorrect

After resolving the dice roll issues, the overall odds are checked, which seems to come down but does align with the expected result as per the rule of the game. The win ratio is in between 42 % to 45 %.

Resolving Bug 3 - Game overall odds appear incorrect

The bug of game overall odd was corrected when the bug on dice roll was corrected.

'Before' and 'after' screen shots identifying the origin of the bug:

Bug 1: Incorrect Payout on win

Before Code:

The screenshot shows with the highlighted are shows the origin of incorrect pay bug. The bug is in "Game.java", playRound() method.

Game.java

```
    private List<Dice> dice;
    private List<DiceValue> values;

    public Game(Dice die1, Dice die2, Dice die3) {
        if (die1 == null || die2 == null || die3 == null) throw new IllegalArgumentException("Dice cannot be null.");
        dice = new ArrayList<Dice>();
        dice.add(die1);
        dice.add(die2);
        dice.add(die3);
        values = new ArrayList<DiceValue>();
    }

    public List<DiceValue> getDiceValues() {
        values.clear();
        for (Dice d : dice) {
            values.add(d.getValue());
        }
        return Collections.unmodifiableList(values);
    }

    public int playRound(Player player, DiceValue pick, int bet ) {
        if (player == null) throw new IllegalArgumentException("Player cannot be null.");
        if (pick == null) throw new IllegalArgumentException("Pick cannot be negative.");
        if (bet < 0) throw new IllegalArgumentException("Bet cannot be negative.");

        player.takeBet(bet);

        int matches = 0;
        for ( Dice d : dice) {
            d.roll();
            if (d.getValue().equals(pick)) {
                matches += 1;
            }
        }

        int winnings = matches * bet;

        if (matches > 0) {
            player.receiveWinnings(winnings);
        }
        return winnings;
    }

}
```

Fig: Screenshot showing the highlighted code that contains error

Before Output:

The following screenshot shows the incorrect output as the result of the bug in the program. It clearly shows the final balance isn't correct on win.

```
Start Game 81:
Fred starts with balance 100, limit 0
Turn 1: Fred bet 5 on CROWN
Rolled CROWN, ANCHOR, CROWN
Fred won 10, balance now 105

Turn 2: Fred bet 5 on ANCHOR
Rolled CROWN, ANCHOR, CROWN
Fred won 5, balance now 105

Turn 3: Fred bet 5 on HEART
Rolled CROWN, ANCHOR, CROWN
Fred lost, balance now 100

Turn 4: Fred bet 5 on DIAMOND
Rolled CROWN, ANCHOR, CROWN
Fred lost, balance now 95

Turn 5: Fred bet 5 on ANCHOR
Rolled CROWN, ANCHOR, CROWN
Fred won 5, balance now 95
```

Fig: screenshot showing incorrect output

After Debugged Code:

The below screenshot shows the highlighted code that has fixed the bug. I have simply returned bet amount back to the player in case of win condition.

Game.java

```
7
8       private List<Dice> dice;
9       private List<DiceValue> values;
10
11      public Game(Dice die1, Dice die2, Dice die3) {
12          if (die1 == null || die2 == null || die3 == null) throw new IllegalArgumentException("Dice cannot be null.");
13          dice = new ArrayList<Dice>();
14          dice.add(die1);
15          dice.add(die2);
16          dice.add(die3);
17          values = new ArrayList<DiceValue>();
18      }
19
20      public List<DiceValue> getDiceValues() {
21          values.clear();
22          for (Dice d : dice) {
23              values.add(d.getValue());
24          }
25          return Collections.unmodifiableList(values);
26      }
27
28      public int playRound(Player player, DiceValue pick, int bet ) {
29          if (player == null) throw new IllegalArgumentException("Player cannot be null.");
30          if (pick == null) throw new IllegalArgumentException("Pick cannot be negative.");
31          if (bet < 0) throw new IllegalArgumentException("Bet cannot be negative.");
32
33          player.takeBet(bet);
34
35          int matches = 0;
36          for ( Dice d : dice) {
37              d.roll();
38              if (d.getValue().equals(pick)) {
39                  matches += 1;
40              }
41          }
42
43          int winnings = matches * bet;
44
45          if (matches > 0) {
46              player.receiveWinnings(winnings+bet);
47          }
48          return winnings;
49      }
50
51  }
```

Fig: Screenshot showing the highlighted debugged code

After Debugged Output:

The following screenshot shows the correct output as the result of fixing of the bug in the program. It clearly shows the final balance is correct on win.

```
Turn 221: Fred bet 5 on DIAMOND
Rolled CROWN, CROWN, HEART
Fred won 5, balance now 25

Turn 222: Fred bet 5 on DIAMOND
Rolled SPADE, HEART, HEART
Fred won 5, balance now 30

Turn 223: Fred bet 5 on ANCHOR
Rolled CROWN, ANCHOR, DIAMOND
Fred lost, balance now 25

Turn 224: Fred bet 5 on CLUB
Rolled HEART, ANCHOR, SPADE
Fred lost, balance now 20

Turn 225: Fred bet 5 on ANCHOR
Rolled DIAMOND, ANCHOR, CLUB
Fred won 5, balance now 25
```

fig: Screenshot showing correct output

BUG 2: Player cannot reach betting limit

Before code:

The following screenshot shows the highlighted code which is the main source of the bug. The bug was present in "Player.java" class, balanceExceedsLimitBy() method. The game only plays if the player balance is greater than bet amount but not equal to bet amount.

Player.java

```
3     private String name;
4     private int balance;
5     private int limit;
6
7     public Player(String name, int balance) {
8         if (name == null || name .isEmpty()) throw new IllegalArgumentException("Name cannot be null or empty");
9         if (balance < 0) throw new IllegalArgumentException("Balance cannot be negative");
10        this.name = name;
11        this.balance = balance;
12        this.limit = 0;
13    }
14
15    public String getName() { return name; }
16    public int getBalance() { return balance; }
17    public int getLimit() { return limit; }
18
19    public void setLimit(int limit) {
20        if (limit < 0) throw new IllegalArgumentException("Limit cannot be negative.");
21        if (limit > balance)  throw new IllegalArgumentException("Limit cannot be greater than balance.");
22        this.limit = limit;
23    }
24
25    public boolean balanceExceedsLimit() {
26        return (balance > limit);
27    }
28
29    public boolean balanceExceedsLimitBy(int amount) {
30        return (balance - amount >limit);
31    }
```

Fig: Screenshot highlighting the source of bug.

Before Output:

The following screenshot shows the incorrect output as the result of the bug in the program. It clearly shows the game ends before the balance is 0.

```
Turn 34: Fred bet 5 on HEART
Rolled CROWN, ANCHOR, CROWN
Fred lost, balance now 20

Turn 35: Fred bet 5 on CLUB
Rolled CROWN, ANCHOR, CROWN
Fred lost, balance now 15

Turn 36: Fred bet 5 on DIAMOND
Rolled CROWN, ANCHOR, CROWN
Fred lost, balance now 10

Turn 37: Fred bet 5 on CLUB
Rolled CROWN, ANCHOR, CROWN
Fred lost, balance now 5

37 turns later.
End Game 99: Fred now has balance 5
```

Fig: Screenshot showing incorrect output as a result of bug.

After Debugging code:

The following screenshot shows the highlighted bug free code The bug was present in "Player.java" class, balanceExceedsLimitBy() method. The game only plays if the player balance is greater than bet amount but not equal to bet amount. So I just modified the code so that the game is on when the player balance equals to or greater than bet amount.

*Player.java

```
3     private String name;
4     private int balance;
5     private int limit;
6
7     public Player(String name, int balance) {
8         if (name == null || name .isEmpty()) throw new IllegalArgumentException("Name cannot be null or empty");
9         if (balance < 0) throw new IllegalArgumentException("Balance cannot be negative");
10        this.name = name;
11        this.balance = balance;
12        this.limit = 0;
13    }
14
15    public String getName() { return name; }
16    public int getBalance() { return balance; }
17    public int getLimit() { return limit; }
18
19    public void setLimit(int limit) {
20        if (limit < 0) throw new IllegalArgumentException("Limit cannot be negative.");
21        if (limit > balance)  throw new IllegalArgumentException("Limit cannot be greater than balance.");
22        this.limit = limit;
23    }
24
25    public boolean balanceExceedsLimit() {
26        return (balance > limit);
27    }
28
29    public boolean balanceExceedsLimitBy(int amount) {
30        return (balance - amount >=limit);
31    }
32
```

Fig: Screenshot highlighting the fixed code for elimination of bug.

After Debugging output:

The following screenshot shows the correct output as the result of fixing the bug in the program. It clearly shows the game ends when the balance is 0.

```
Turn 229: Fred bet 5 on ANCHOR
Rolled HEART, HEART, DIAMOND
Fred won 5, balance now 15

Turn 230: Fred bet 5 on DIAMOND
Rolled ANCHOR, CLUB, ANCHOR
Fred lost, balance now 10

Turn 231: Fred bet 5 on CROWN
Rolled SPADE, CLUB, HEART
Fred lost, balance now 5

Turn 232: Fred bet 5 on CLUB
Rolled CROWN, HEART, ANCHOR
Fred lost, balance now 0

232 turns later.
End Game 99: Fred now has balance 0
```

Fig: Screenshot showing correct output as a result of fixing the bug.

Bug 3: Incorrect Game overall odds

Before Code:

It was found out that this bug is present in the game not because of a single error in the code. It is present because there are two other bugs that results in this bug. The two bugs are looked into bug 4 and bug 5 respectively. Please refer to bug u and bug 5 before code screenshot for further reference.

Before output:

The following screenshot shows the incorrect ratio which is higher than expected (0.42-0.45) as the result of the bug in the program. The ratio here is 0.61 which is too high.

```
Turn 51: Fred bet 5 on ANCHOR
Rolled CLUB, HEART, DIAMOND
Fred lost, balance now 10

Turn 52: Fred bet 5 on ANCHOR
Rolled CLUB, HEART, DIAMOND
Fred lost, balance now 5

52 turns later.
End Game 99: Fred now has balance 5

Win count = 3004, Lose Count = 1900, 0.61
```

Fig: Screenshot showing incorrect output as a result of bug.

After debugging code:

As described above the source of this bug was because of bug 4 and bug 5 which we will look next. The after debugged code of this bug is also the debugged code of bug4 and bug 5 collectively. Please refer to the after debugged code screenshot of bug 4 and bug 5 for further reference.

After debugging output:

The following screenshot shows the correct output as the result of fixing the bug in the program. The win ratio here is 0.42 which is in line of the expected result.

```
Turn 230: Fred bet 5 on DIAMOND
Rolled ANCHOR, CLUB, ANCHOR
Fred lost, balance now 10

Turn 231: Fred bet 5 on CROWN
Rolled SPADE, CLUB, HEART
Fred lost, balance now 5

Turn 232: Fred bet 5 on CLUB
Rolled CROWN, HEART, ANCHOR
Fred lost, balance now 0

232 turns later.
End Game 99: Fred now has balance 0

Win count = 8752, Lose Count = 12177, 0.42
```

Fig: Screenshot showing correct output as a result of fixing the bug.

Bug 4: Dice seem to always roll the same after first roll.

Before Code:

The following screenshot shows the reason of the bug present in the program. The bug is present in Dice.java class, DiceValue getValue() method. It is present there because the method is returing the same values of dice on every round of the play.

*Player.java | DiceValue.java | *Dice.java

```
1  package com.company;
2  public class Dice {
3
4      private DiceValue value;
5
6      public Dice() {
7          value =  DiceValue.getRandom();
8      }
9
10     public DiceValue getValue() {
11
12         return value;
13     }
14
15     public DiceValue roll() {
16         return DiceValue.getRandom();
17     }
18
19     public String toString() {
20         return value.toString();
21     }
22 }
23
```

Fig: Screenshot highlighting the source of bug.

Before Output:

The following screenshot shows same output of the dice roll for every round of the game which is incorrect as the result of the bug in the program. The dice roll value doesn't change at all after the first round of dice roll.

Fig: Screenshot showing incorrect output as a result of bug.

After Debugging Code:

The following screenshot shows the code that fixes the bug. As the DiceValue getValue() was returning the same value of dice, now I have made it to random so it returns the random value.

*Player.java DiceValue.java *Dice.java

```
1  package com.company;
2  public class Dice {
3
4      private DiceValue value;
5
6      public Dice() {
7          value =  DiceValue.getRandom();
8      }
9
10     public DiceValue getValue() {
11         this.value=DiceValue.getRandom();
12
13         return value;
14     }
15
16     public DiceValue roll() {
17         return DiceValue.getRandom();
18     }
19
20     public String toString() {
21         return value.toString();
22     }
23 }
24
```

Fig: Screenshot highlighting the fixed code for elimination of bug.

After Debugging Output:

The following screenshot shows the different output of the dice roll for every round of the game which is correct as the result of fixing the bug in the program. The dice roll value does change at all round of dice roll.

```
Rolled CROWN, CROWN, CLUB
Fred lost, balance now 15

Turn 127: Fred bet 5 on DIAMOND
Rolled CROWN, DIAMOND, HEART
Fred won 5, balance now 20

Turn 128: Fred bet 5 on CLUB
Rolled CROWN, ANCHOR, DIAMOND
Fred won 5, balance now 25

Turn 129: Fred bet 5 on CLUB
Rolled CLUB, CLUB, HEART
Fred lost, balance now 20

Turn 130: Fred bet 5 on DIAMOND
Rolled ANCHOR, DIAMOND, HEART
Fred lost, balance now 15

Turn 131: Fred bet 5 on ANCHOR
Rolled ANCHOR, CLUB, CROWN
Fred won 5, balance now 20

Turn 132: Fred bet 5 on DIAMOND
Rolled CLUB, HEART, CLUB
Fred lost, balance now 15

Turn 133: Fred bet 5 on CLUB
Rolled HEART, HEART, DIAMOND
Fred lost, balance now 10

Turn 134: Fred bet 5 on DIAMOND
Rolled CROWN, CROWN, CROWN
Fred won 10, balance now 20

Turn 135: Fred bet 5 on ANCHOR
Rolled ANCHOR, DIAMOND, ANCHOR
Fred won 5, balance now 25

Turn 136: Fred bet 5 on HEART
Rolled CLUB, ANCHOR, CROWN
Fred lost, balance now 20
```

Fig: Screenshot showing correct output

Bug 5: "SPADE" doesn't appear in roll.

Before Code:

The following Screenshot shows the highlighted portion of the code resulting in the bug with no "spade" appear in dice roll. The bug is present in DiceValue.java class , DiceValue getRandon() method. The random assigned in this method skips the SPADE on every occasion of the roll that results in the bug.

*Player.java | *DiceValue.java | *Dice.java

```
1  package com.company;
2  import java.util.HashMap;
5
6  public enum DiceValue {
7      CROWN, ANCHOR, HEART, DIAMOND, CLUB, SPADE;
8
9      private static Random RANDOM = new Random();
10
11     private static final Map<DiceValue, String> VALUE_REPR_MAP= new HashMap<DiceValue, String>();
12     static {
13         VALUE_REPR_MAP.put(DiceValue.CROWN, "Crown");
14         VALUE_REPR_MAP.put(DiceValue.ANCHOR, "Anchor");
15         VALUE_REPR_MAP.put(DiceValue.HEART, "Heart");
16         VALUE_REPR_MAP.put(DiceValue.DIAMOND, "Diamond");
17         VALUE_REPR_MAP.put(DiceValue.CLUB, "Club");
18         VALUE_REPR_MAP.put(DiceValue.SPADE, "Spade");
19     }
20
21     public String toString(DiceValue value) {
22         return VALUE_REPR_MAP.get(value);
23     }
24
25     public static DiceValue getRandom() {
26         int random = RANDOM.nextInt(DiceValue.SPADE.ordinal());
27         return values()[random];
28     }
29
30 }
```

Fig: Screenshot highlighting the source of bug.

Before Output:

The following output shows no occurrence of "Spade" as the result of bug. Looking into all the rounds of play there is not even a single occurrence of "spade" which is incorrect.

```
Fred starts with balance 100, limit 0
Turn 1: Fred bet 5 on CLUB
Rolled CLUB, CROWN, CLUB
Fred lost, balance now 95

Turn 2: Fred bet 5 on ANCHOR
Rolled CROWN, CLUB, ANCHOR
Fred won 10, balance now 100

Turn 3: Fred bet 5 on HEART
Rolled HEART, CROWN, HEART
Fred won 5, balance now 100

Turn 4: Fred bet 5 on CLUB
Rolled DIAMOND, ANCHOR, ANCHOR
Fred lost, balance now 95

Turn 5: Fred bet 5 on ANCHOR
Rolled CROWN, DIAMOND, CLUB
Fred won 5, balance now 95

Turn 6: Fred bet 5 on ANCHOR
Rolled HEART, CROWN, CLUB
Fred won 5, balance now 95

Turn 7: Fred bet 5 on ANCHOR
Rolled HEART, HEART, HEART
Fred won 5, balance now 95

Turn 8: Fred bet 5 on CROWN
Rolled ANCHOR, DIAMOND, CROWN
Fred won 5, balance now 95

Turn 9: Fred bet 5 on HEART
Rolled CLUB, ANCHOR, CLUB
Fred lost, balance now 90

Turn 10: Fred bet 5 on HEART
Rolled HEART, CLUB, HEART
Fred lost, balance now 85

Turn 11: Fred bet 5 on HEART
Rolled DIAMOND, ANCHOR, DIAMOND
Fred lost, balance now 80
```

Fig: Screenshot showing no “spade” in output as a result of bug.

After debugging code:

The following screenshot shows the highlighted portion of the code that has been debugged to eliminate the "**spade doesn't occurs in dice roll"** bug. I have simply changed the code so that random occurs as per the size of the map that holds the value of the dice rather than the dice Value that was skipping SPADE.

*Player.java | DiceValue.java | *Dice.java

```
1  package com.company;
2  import java.util.HashMap;
5
6  public enum DiceValue {
7      CROWN, ANCHOR, HEART, DIAMOND, CLUB, SPADE;
8
9      private static Random RANDOM = new Random();
10
11     private static final Map<DiceValue, String> VALUE_REPR_MAP= new HashMap<DiceValue, String>();
12     static {
13         VALUE_REPR_MAP.put(DiceValue.CROWN, "Crown");
14         VALUE_REPR_MAP.put(DiceValue.ANCHOR, "Anchor");
15         VALUE_REPR_MAP.put(DiceValue.HEART, "Heart");
16         VALUE_REPR_MAP.put(DiceValue.DIAMOND, "Diamond");
17         VALUE_REPR_MAP.put(DiceValue.CLUB, "Club");
18         VALUE_REPR_MAP.put(DiceValue.SPADE, "Spade");
19     }
20
21     public String toString(DiceValue value) {
22         return VALUE_REPR_MAP.get(value);
23     }
24
25     public static DiceValue getRandom() {
26         //int random = RANDOM.nextInt(DiceValue.SPADE.ordinal());
27         int random = RANDOM.nextInt(VALUE_REPR_MAP.size());
28         return values()[random];
29     }
30
31 }
32
```

Fig: Screenshot highlighting the fixed code for elimination of bug.

After Debugging Output:

The below screenshot shows the output after fixing the bug. The output clearly shows the occurrence of the "spade" in the dice roll values.

```
Turn 168: Fred bet 5 on CROWN
Rolled ANCHOR, HEART, HEART
Fred won 10, balance now 20

Turn 169: Fred bet 5 on ANCHOR
Rolled DIAMOND, DIAMOND, CLUB
Fred lost, balance now 15

Turn 170: Fred bet 5 on ANCHOR
Rolled SPADE, ANCHOR, CLUB
Fred lost, balance now 10

Turn 171: Fred bet 5 on DIAMOND
Rolled SPADE, CROWN, CROWN
Fred won 5, balance now 15

Turn 172: Fred bet 5 on CROWN
Rolled ANCHOR, CROWN, HEART
Fred won 5, balance now 20

Turn 173: Fred bet 5 on SPADE
Rolled CROWN, HEART, DIAMOND
Fred lost, balance now 15

Turn 174: Fred bet 5 on HEART
Rolled HEART, DIAMOND, CROWN
Fred lost, balance now 10

Turn 175: Fred bet 5 on DIAMOND
Rolled DIAMOND, HEART, HEART
Fred lost, balance now 5

Turn 176: Fred bet 5 on CROWN
Rolled ANCHOR, SPADE, CROWN
Fred lost, balance now 0
```

Fig: Screenshot showing the occurrence of “spade” in dice roll

Output from the automated test demonstrating correct operation after resolution of each bug:

Bug 1: Incorrect Payout on win

Following is the screenshot of the output of the automated test (unit test) that has passed after fixing the bug. The test was check against all the zero, one , two and three matched condition.

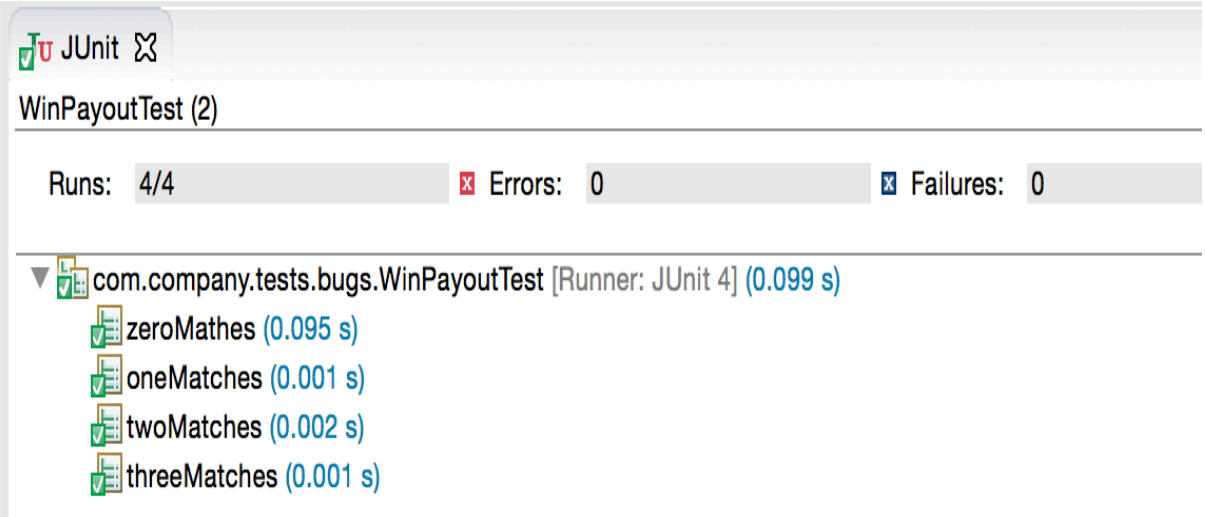


Fig: Screenshot showing the automated test has passed

All the four tests have passed after fixing the bug. There is 0 failure.

BUG 2: Player cannot reach betting limit

Following is the screenshot of the output of the automated test (unit test) that has passed after fixing the bug. The test was carried out in the test pass condition that the player reaches 0 to end the game.

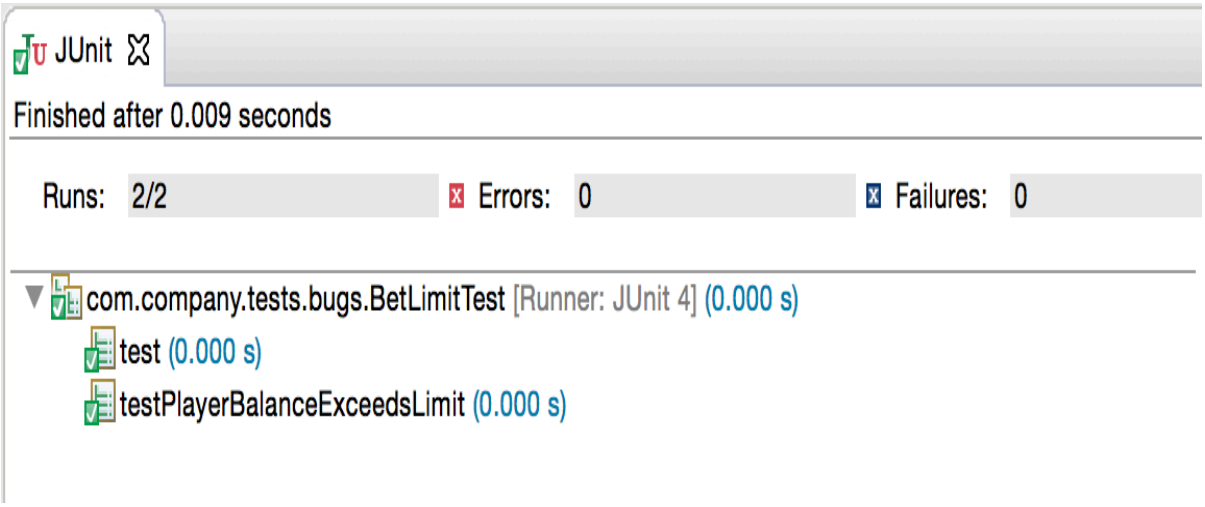


Fig: Screenshot showing the automated test has passed

Both the tests have passed after fixing the bug. There is 0 failure in the test.

Bug 3: Game overall odds

Following is the screenshot shows the automated test for overall game odds has passed after resolving the bug. The test was carried out to test the overall win ratio was between 0.42 to 0.45.

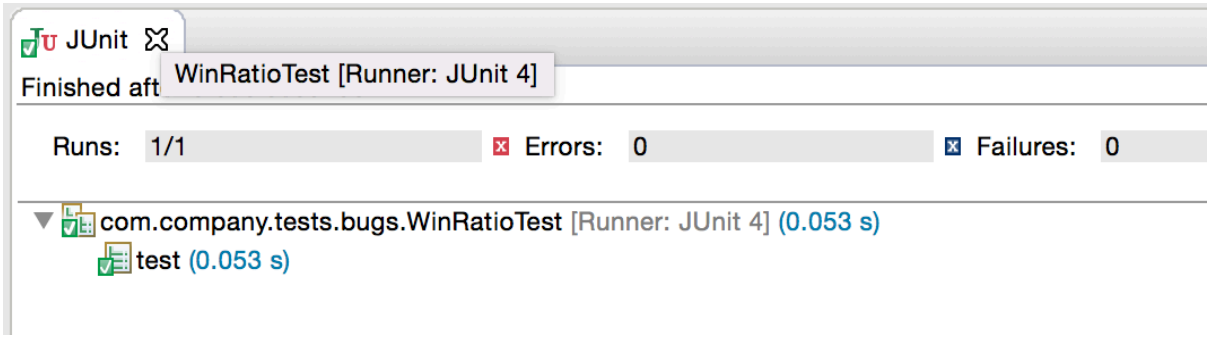


Fig: Screenshot showing the automated test has passed

Bug 4: Dice seem to always roll the same after first roll.

The following screenshot is the "system out" testing method. In this method the results are printed through the unit test and analyzed to validate the bug fix.

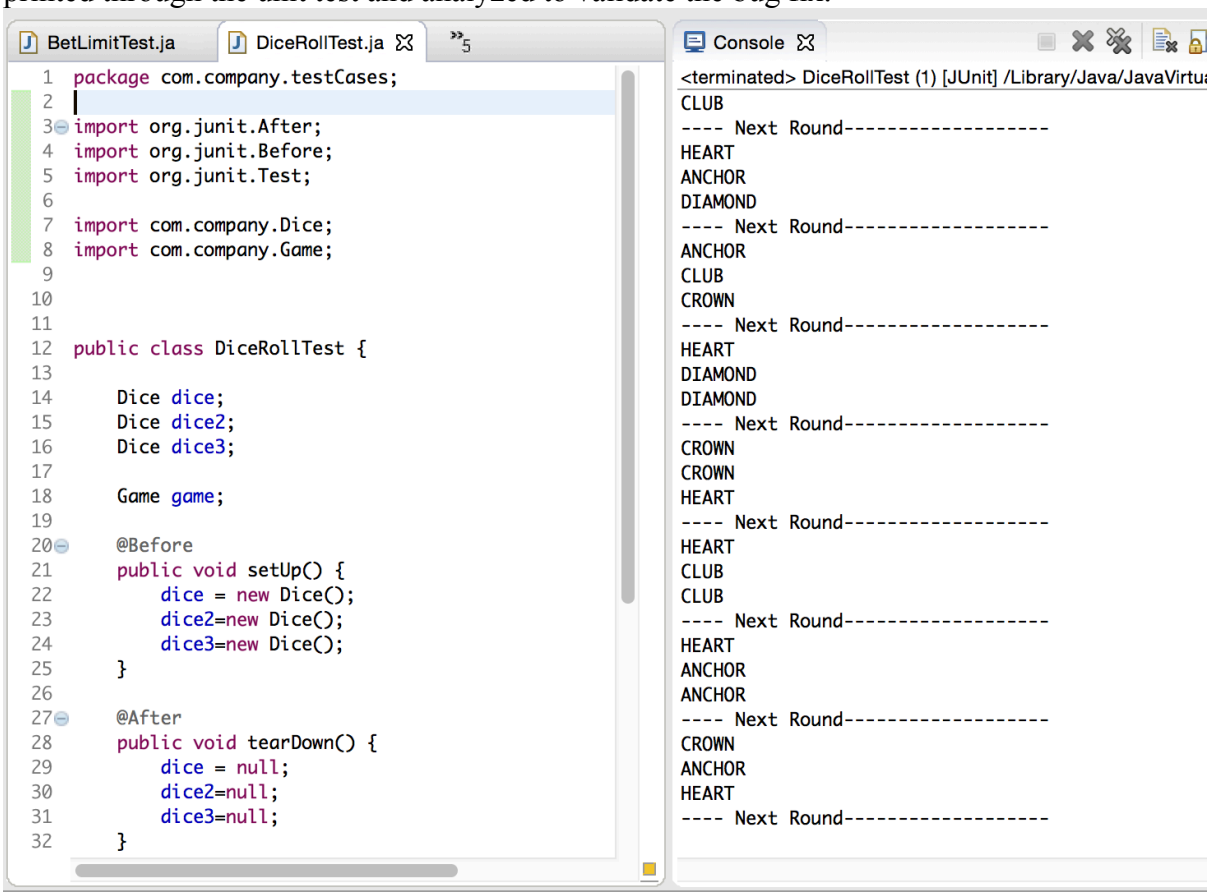


Fig: Screenshot showing the automated test has passed

In the above screenshot, the result shows the different value of dice roll for different rounds of play which means the test has passed.

Bug 5: "SPADE" doesn't appear in roll.

Following is the screenshot of the automated test that was testing the "spade" not showing in the dice roll. The screenshot shows that the automated test has passed after the bug was fixed, hence no failure occurs in the test. The test was carried out to test if the "spade" comes out in any of the round while rolling the dice.

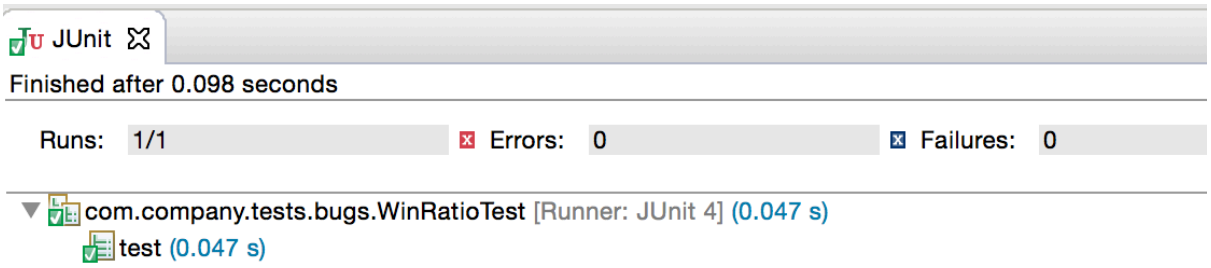


Fig: Screenshot showing the automated test has passed

Conclusion:

Hence the five bugs were identified through UAT and unit tests. Hypothesis were developed for each bug and they were applied and tested to fix the bug. All the bugs were fixed and the UAT and unit test were re applied to test the debugged code to verify the bug fix. All these procedures were documented and all the actions were updated to the Github repository.