

ITC515 - Assignment item 4

DEBUGGING

STUDENT NAME: HAI HA DO
STUDENT NUMBER: 11635010

Contents

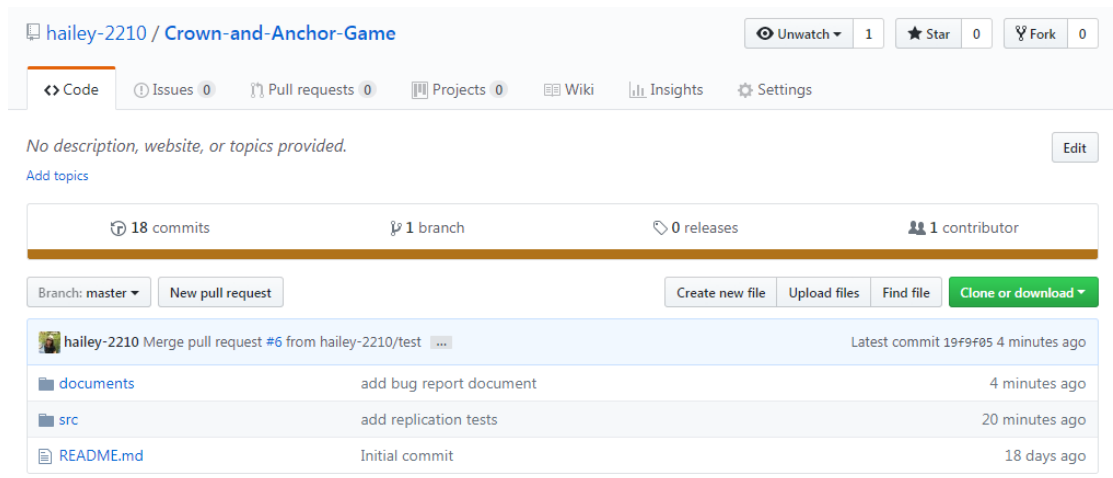
1	Repository Information.....	3
1.1	Location of Files in Repository	3
2	Bug 1 - The player is not paid out correctly	5
2.1	Replication	5
2.2	Simplification	7
2.3	Tracing.....	10
2.4	Hypothesis.....	12
2.5	Resolution	13
2.6	Result	15
2.6.1	Bug1Test.java	15
2.6.2	Console from Main.....	16
3	Bug 2 - Player cannot reach betting limit.....	17
3.1	Replication	17
3.2	Simplification	18
3.3	Tracing.....	20
3.4	Hypothesis.....	21
3.5	Resolution	22
3.6	Result	22
3.6.1	Bug2Test.java	22
3.6.2	Console from Main.....	23
4	Bug 3 - The DiceValues are the same for each game.....	24
4.1	Replication	24
4.2	Simplification	25
4.3	Tracing.....	26
4.4	Hypotheses	28
4.5	Resolution	29
4.6	Result	30
4.6.1	Bug3Test.....	30
4.6.2	Console from Main.....	31
5	Bug 4 - SPADE is ever rolled or guessed.....	32
5.1	Replication	32

5.2	Simplification	33
5.3	Tracing.....	35
5.4	Hypotheses	36
5.5	Resolution	38
5.6	Result	39
5.6.1	Bug4Test.....	39
5.6.2	Console from Main.....	40
6	Bug 5 - Odds of game are incorrect	41
6.1	Replication	41
6.2	Simplification	42
6.3	Tracing.....	45
6.4	Hypothesis.....	46
6.5	Resolution	46
6.6	Result	46
6.6.1	Bug5Test.....	46
6.6.2	Main Console	47

1 Repository Information

The URL for the assignment 4 repository is:

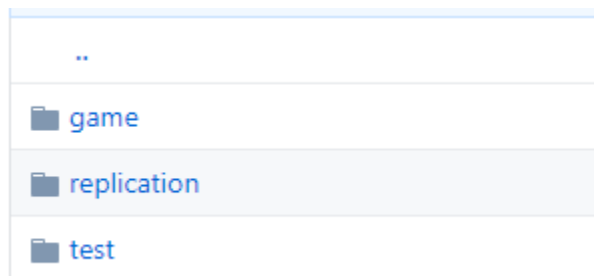
<https://github.com/hailey-2210/Crown-and-Anchor-Game.git>



1.1 Location of Files in Repository






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/documents/

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

..
 Bug 1- Incorrect Pay Out
 Bug 2- Cant Reach Bet Limit
 Bug 3- Same DiceValue
 Bug 4- No SPADE
 Bug 5- Incorrect Odds
 Bug Report.pdf

2 Bug 1 - The player is not paid out correctly

2.1 Replication

Test Name	Test balance increase correctly after winning
Use Case Tested:	Crown and Anchor Game
Test Description:	<p>Test whether the player is paid the correct amount</p> <p>In particular:</p> <ul style="list-style-type: none">- 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
1.	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
2.	Look at each turn and identify one where the player has made matches.	A turn should be presented	Pass
3.	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
4.	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
<p>Fred lost, balance now 70</p> <p>Turn 13: Fred bet 5 on ANCHOR Rolled HEART, HEART, ANCHOR Fred won 5, balance now 70</p>	<p>Initial balance: 70</p> <p>Balance after turn: 70</p> <p>Expected: 75</p> <p>Result: FAIL</p>
<p>Fred lost, balance now 95</p> <p>Turn 2: Fred bet 5 on HEART Rolled DIAMOND, DIAMOND, HEART Fred won 5, balance now 95</p>	<p>Initial balance: 95</p> <p>Balance after turn: 95</p> <p>Expected: 100</p> <p>Result: FAIL</p>
<p>Start Game</p> <p>Fred starts with balance 100, limit 0</p> <p>Turn 1: Fred bet 5 on CROWN Rolled HEART, CROWN, DIAMOND Fred won 5, balance now 100</p>	<p>Initial balance: 100</p> <p>Balance after turn: 100</p> <p>Expected: 105</p> <p>Result: FAIL</p>
<p>Fred lost, balance now 90</p> <p>Turn 3: Fred bet 5 on HEART Rolled HEART, HEART, ANCHOR Fred won 10, balance now 95</p>	<p>Initial balance: 90</p> <p>Balance after turn: 95</p> <p>Expected: 100</p> <p>Result: FAIL</p>
<p>Fred won 5, balance now 85</p> <p>Turn 8: Fred bet 5 on DIAMOND Rolled DIAMOND, DIAMOND, HEART Fred won 10, balance now 90</p>	<p>Initial balance: 85</p> <p>Balance after turn: 90</p> <p>Expected: 95</p> <p>Result: FAIL</p>
<p>Fred lost, balance now 95</p> <p>Turn 2: Fred bet 5 on DIAMOND Rolled DIAMOND, DIAMOND, HEART Fred won 10, balance now 100</p>	<p>Initial balance: 95</p> <p>Balance after turn: 100</p> <p>Expected: 105</p> <p>Result: FAIL</p>

2.2 Simplification

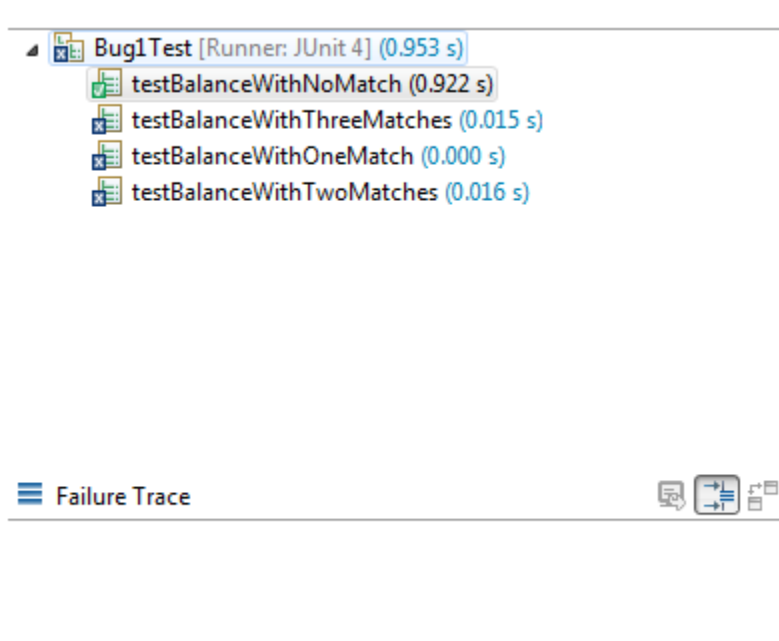
Test Name	Test balance increase correctly after winning
Use Case Tested:	Automate the testing of errors in UAT Test 1
Test Description:	<p>Test whether the player is paid the correct amount</p> <p>In particular:</p> <ul style="list-style-type: none">- If he makes one match, he should get the initial 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	<p>Single player "October" created, balance = 100, bet = 5</p> <p>Each run to use a single value "HEART" as the player's pick.</p> <p>Run game for 4 given combination of dice values.</p>
Post-conditions	n/a

	TEST STEP	EXPECTED TEST RESULTS	RESULT
1.	Run Bug1Test.java	JUnit test and Console are opened.	Pass
2.	Check result of testBalanceWithNoMatch in Failure Trace	JUnit test should be no error and no failure	Pass
3.	Check result of testBalanceWithNoMatch in Console	Balance = 95 Winnings = 0	Pass
4.	Check result of testBalanceWithOneMatch in Failure Trace	JUnit test should be no error and no failure	Fail
5.	Check result of testBalanceWithOneMatch in Console	Balance = 105 Winnings = 5	Fail
6.	Check result of testBalanceWithTwoMatches in Failure Trace	JUnit test should be no error and no failure	Fail

	TEST STEP	EXPECTED TEST RESULTS	RESULT
7.	Check result of testBalanceWithTwoMatches in Console	Balance = 110 Winnings = 10	Fail
8.	Check result of testBalanceWithThreeMatches in Failure Trace	JUnit test should be no error and no failure	Fail
9.	Check result of testBalanceWithThreeMatches in Console	Balance = 115 Winnings = 15	Fail

Result

1. No match: PASS



Failure Trace

```

Automated Test for Bug 1: The player is not paid out correctly
Start game
October starts with balance 100, limit 0

Rolled ANCHOR, ANCHOR, DIAMOND
October bet 5 on HEART
balance now 95

Winnings: 0

```

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

Bug1Test [Runner: JUnit 4] (0.953 s)

- testBalanceWithNoMatch (0.922 s)
- testBalanceWithThreeMatches (0.015 s)
- testBalanceWithOneMatch (0.000 s)
- testBalanceWithTwoMatches (0.016 s)

Failure Trace

java.lang.AssertionError: expected:<105> but was:<100>
at Bug1Test.testBalanceWithOneMatch(Bug1Test.java:83)

Automated Test for Bug 1: The player is not paid out correctly
Start game
October starts with balance 100, limit 0

Rolled HEART, ANCHOR, DIAMOND
October bet 5 on HEART
balance now 100

Winnings: 5

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

Bug1Test [Runner: JUnit 4] (0.953 s)

- testBalanceWithNoMatch (0.922 s)
- testBalanceWithThreeMatches (0.015 s)
- testBalanceWithOneMatch (0.000 s)
- testBalanceWithTwoMatches (0.016 s)

Failure Trace

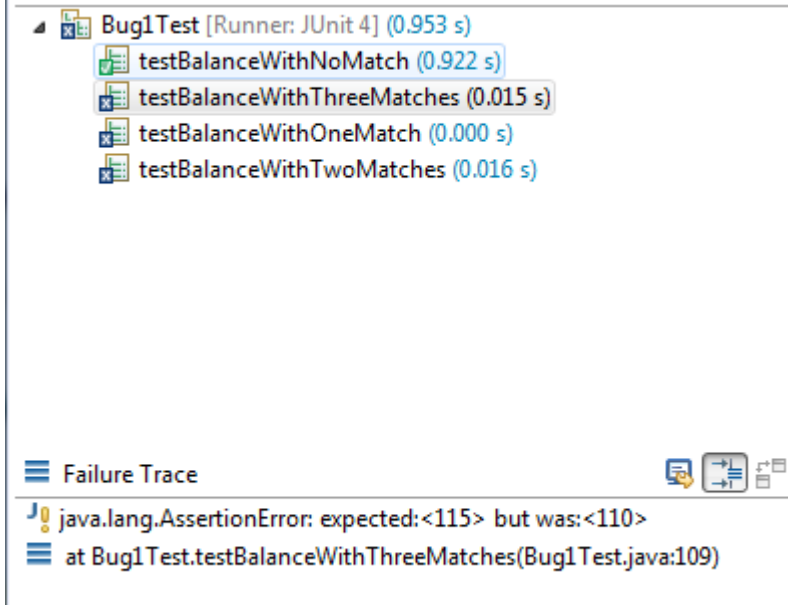
java.lang.AssertionError: expected:<110> but was:<105>
at Bug1Test.testBalanceWithTwoMatches(Bug1Test.java:96)

```
Automated Test for Bug 1: The player is not paid out correctly
Start game
October starts with balance 100, limit 0
```

```
Rolled HEART, HEART, DIAMOND
October bet 5 on HEART
balance now 105
```

```
Winnings: 10
```

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



Bug1Test [Runner: JUnit 4] (0.953 s)

- testBalanceWithNoMatch (0.922 s)
- testBalanceWithThreeMatches (0.015 s)
- testBalanceWithOneMatch (0.000 s)
- testBalanceWithTwoMatches (0.016 s)

Failure Trace

java.lang.AssertionError: expected:<115> but was:<110>
at Bug1Test.testBalanceWithThreeMatches(Bug1Test.java:109)

```
Automated Test for Bug 1: The player is not paid out correctly
Start game
October starts with balance 100, limit 0
```

```
Rolled HEART, HEART, HEART
October bet 5 on HEART
balance now 110
```

```
Winnings: 15
```

2.3 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**).

```

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

System.out.printf("Rolled %s, %s, %s\n",
    cdv.get(0), cdv.get(1), cdv.get(2));

if (winnings > 0) {
    System.out.printf("%s won %d, balance now %d\n\n",
        player.getName(), winnings, player.getBalance());
    winCount++;
}
else {
    System.out.printf("%s lost, balance now %d\n\n",
        player.getName(), player.getBalance());
    loseCount++;
}

```

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

```

public void takeBet(int bet) {
    if (bet < 0) throw new IllegalArgumentException("Bet cannot be zero or negative.");
    if (!balanceExceedsLimitBy(bet)) throw new IllegalArgumentException("Placing bet would go below limit.");
    balance = balance - bet;
}

public void receiveWinnings(int winnings) {
    if (winnings < 0) throw new IllegalArgumentException("Winnings cannot be negative.");
    balance = balance + winnings;
}

```

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

```

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 ++;
        }
    }

    int winnings = matches * bet;

    if (matches > 0) {
        player.receiveWinnings(winnings); // refund the bet
    }
    return winnings;
}

```

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: $balance = (initial) balance - bet$
- If one match, $balance = balance + bet = (initial) balance$ instead of $(initial) balance + bet$
- If two matches, $balance = balance + 2 * bet = (initial) balance + bet$ instead of $(initial) balance + 2 * bet$
- If three matches, $balance = balance + 3 * bet = (initial) balance + 2 * bet$ instead of $(initial) balance + 3 * bet$

2.4 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

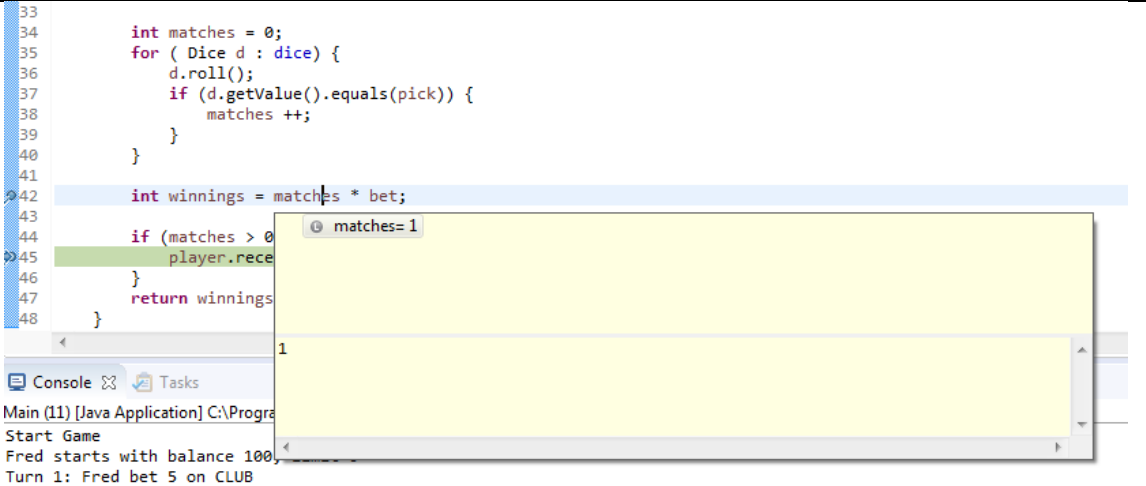
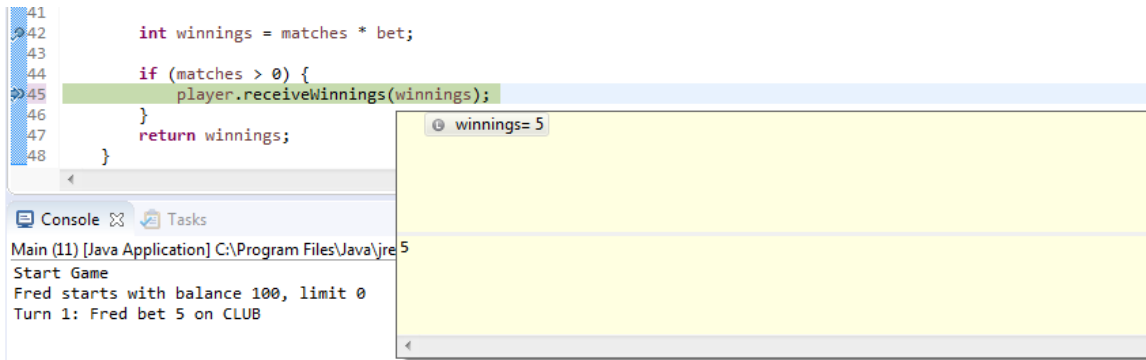
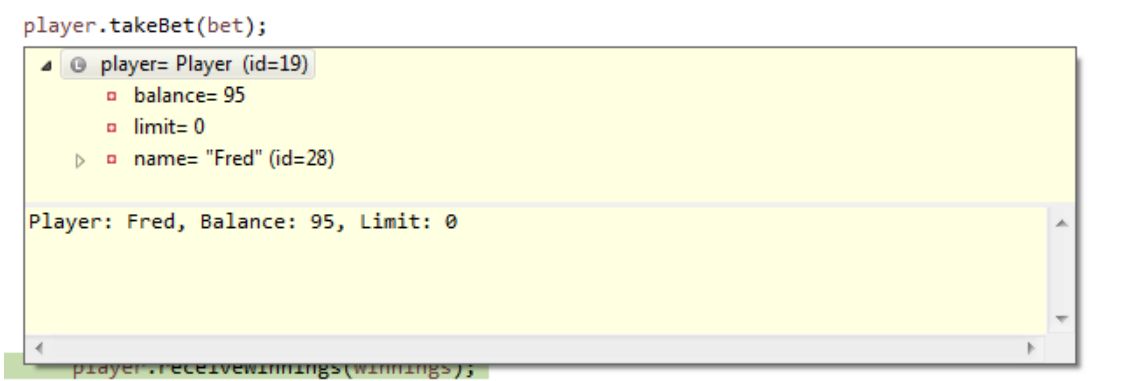
- (1) `int winnings = matches * bet;`
- (2) `player.receiveWinnings(winnings);`

The debugging shows the results for the hypotheses as below:

- Result from the console:

```
Start Game
Fred starts with balance 100, limit 0
Turn 1: Fred bet 5 on CLUB
Rolled DIAMOND, CLUB, CROWN
Fred won 5, balance now 100
```

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.

	
<p>Hypothesis 2: The winnings are added correctly</p>	<p>At the stage of calculating winnings. So (2) is true.</p> 
<p>Hypothesis 3: The balance becomes incorrect after taking the bet</p>	<p>Moving to the stage of take bet: the balance of player is wrong. It should be 100. So (3) is true.</p> 

2.5 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:

```
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 ++;
        }
    }

    int winnings = matches * bet;

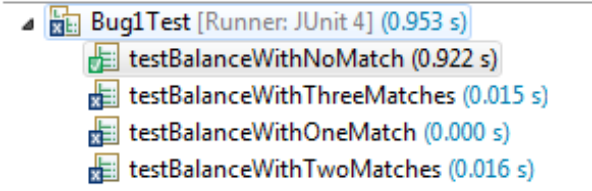

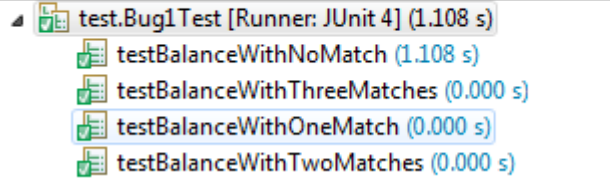
    if (matches > 0) {
        player.receiveWinnings(winnings);
    }

    else player.takeBet(bet);
    return winnings;
}
```

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.

2.6 Result

2.6.1 Bug1Test.java

BEFORE	AFTER
	<p>Runs: 4/4 Errors: 0 Failures:</p>  
<p>One match</p> <p>Automated Test for Bug 1: The player is not pa Start game October starts with balance 100, limit 0</p> <p>Rolled HEART, ANCHOR, DIAMOND October bet 5 on HEART balance now 100</p> <p>Winnings: 5</p>	<p>One match</p> <p>Automated Test for Bug 1: The player is not pa Start game October starts with balance 100, limit 0</p> <p>Rolled HEART, ANCHOR, DIAMOND October bet 5 on HEART balance now 105</p> <p>Winnings: 5</p>
<p>Two match</p> <p>Automated Test for Bug 1: The player is not pa Start game October starts with balance 100, limit 0</p> <p>Rolled HEART, HEART, DIAMOND October bet 5 on HEART balance now 105</p> <p>Winnings: 10</p>	<p>Two match</p> <p>Automated Test for Bug 1: The player is not pa Start game October starts with balance 100, limit 0</p> <p>Rolled HEART, HEART, DIAMOND October bet 5 on HEART balance now 110</p> <p>Winnings: 10</p>
<p>Three match</p>	<p>Three match</p>

BEFORE	AFTER
<p>Automated Test for Bug 1: The player is not pa</p> <p>Start game</p> <p>October starts with balance 100, limit 0</p> <p>Rolled HEART, HEART, HEART</p> <p>October bet 5 on HEART</p> <p>balance now 110</p> <p>Winnings: 15</p>	<p>Automated Test for Bug 1: The player is not</p> <p>Start game</p> <p>October starts with balance 100, limit 0</p> <p>Rolled HEART, HEART, HEART</p> <p>October bet 5 on HEART</p> <p>balance now 115</p> <p>Winnings: 15</p>

2.6.2 Console from Main

Result from the console shows no error as well:

BEFORE	AFTER
<p>Fred lost, balance now 70</p> <p>Turn 13: Fred bet 5 on ANCHOR</p> <p>Rolled HEART, HEART, ANCHOR</p> <p>Fred won 5, balance now 70</p>	<p>Start Game</p> <p>Fred starts with balance 100, limit 0</p> <p>Turn 1: Fred bet 5 on CLUB</p> <p>Rolled CROWN, CROWN, CLUB</p> <p>Fred won 5, balance now 105</p> <p>Turn 2: Fred bet 5 on CROWN</p> <p>Rolled CROWN, CROWN, CLUB</p> <p>Fred won 10, balance now 115</p> <p>Turn 3: Fred bet 5 on CROWN</p> <p>Rolled CROWN, CROWN, CLUB</p> <p>Fred won 10, balance now 125</p> <p>Turn 4: Fred bet 5 on CLUB</p> <p>Rolled CROWN, CROWN, CLUB</p> <p>Fred won 5, balance now 130</p> <p>Turn 5: Fred bet 5 on DIAMOND</p> <p>Rolled CROWN, CROWN, CLUB</p> <p>Fred lost, balance now 125</p> <p>Turn 6: Fred bet 5 on HEART</p> <p>Rolled CROWN, CROWN, CLUB</p> <p>Fred lost, balance now 120</p>

3 Bug 2 - Player cannot reach betting limit

3.1 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
1.	Run Main.java with player details: Player name = "Fred" Balance = 100 Limit = 0	Console opens and the results of games are displayed.	Pass
2.	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"
3.	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
Turn 58: Fred bet 5 on CLUB Rolled DIAMOND, ANCHOR, DIAMOND Fred lost, balance now 5 58 turns later. End Game 99: Fred now has balance 5	Result: FAIL
Turn 443: Fred bet 5 on ANCHOR Rolled CROWN, DIAMOND, CROWN Fred lost, balance now 5 443 turns later. End Game 99: Fred now has balance 5	Result: FAIL

Turn 117: Fred bet 5 on ANCHOR Rolled DIAMOND, CROWN, CROWN Fred lost, balance now 5 117 turns later. End Game 99: Fred now has balance 5	Result: FAIL
---	--------------




3.2 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
1.	Run Bug1Test.java	Junit test and Console are opened.	Pass
2.	Check result of testPlayWithBalanceExceedsBet in Failure Trace	JUnit test should be no error and no failure	Pass
3.	Check result of testPlayWithBalanceExceedsBet in Console	Balance = 5 Limit = 0	Pass
4.	Check result of testPlayWithBalanceEqualsBet in Failure Trace	JUnit test should be no error and no failure	Fail
5.	Check result of testPlayWithBalanceEqualsBet in Console	Balance = 0 Limit = 0	Fail

Result

1. When balance exceeds bet: PASS

▲  Bug2Test [Runner: JUnit 4] (1.030 s)
  testPlayWithBalanceExceedsBet (1.014 s)
  testPlayWithBalanceEqualsBet (0.016 s)

 Failure Trace

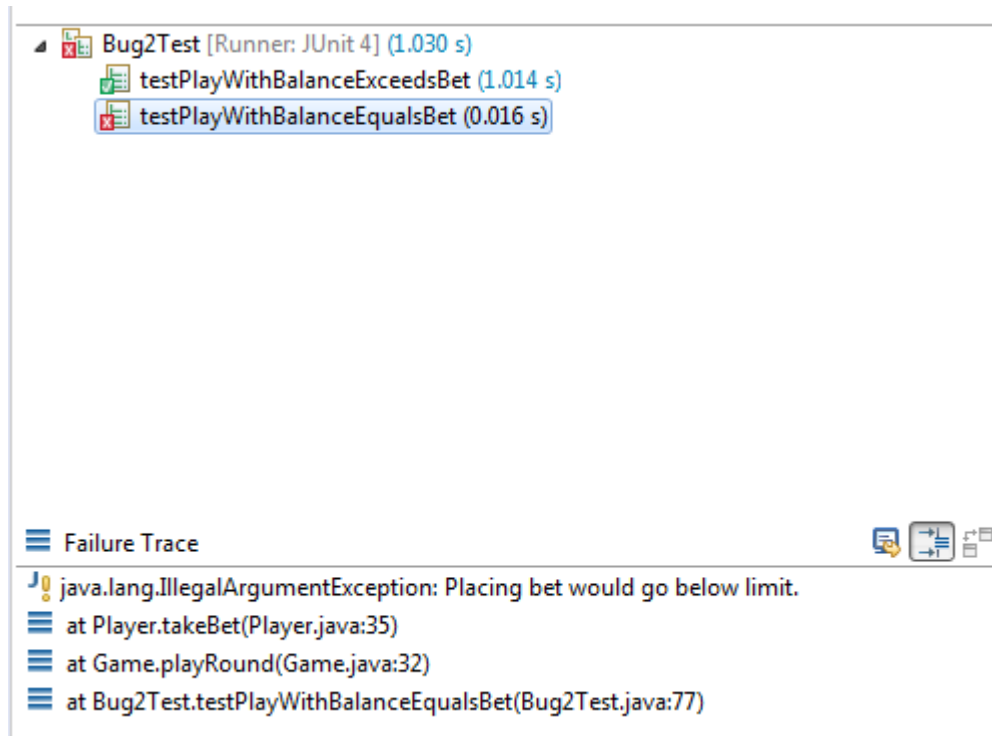


Automated Test for Bug 2: Player cannot reach betting limit
Start Game

October starts with balance 6, limit 0
Rolled ANCHOR, ANCHOR, ANCHOR
October bet 5 on HEART
Balance now 1 | Limit: 0

Winnings for this bet:0

2. When balance equals bet: FAIL



Automated Test for Bug 2: Player cannot reach betting limit
Start Game

October starts with balance 5, limit 0
Rolled ANCHOR, ANCHOR, ANCHOR
October bet 5 on HEART

3.3 Tracing

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

```
while (player.balanceExceedsLimitBy(bet) && player.getBalance() < 200)  
{  
    turn++;  
    DiceValue pick = DiceValue.getRandom();
```

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:

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

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.

3.4 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: <i>setLimit()</i> method correctly assigns limit as zero.	<p>(1) is correct</p> <pre> public void setLimit(int limit) { if (limit < 0) throw new IllegalArgumentException("Limit cannot be negative."); if (limit > balance) throw new IllegalArgumentException("Limit cannot be greater than balance."); this.limit = limit; } </pre> 
Hypothesis 2: The game ends (negatively) when the limit is reached.	<p>After run, the game as at balance = 5, limit = 0 so (2) is verified.</p> 
Hypothesis 3: <i>balanceExceedsLimitBy</i> does not include the minimum limit.	<p>This is apparently true</p> <pre> public boolean balanceExceedsLimitBy(int amount) { return (balance - amount > limit); } </pre>

3.5 Resolution

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

```
public boolean balanceExceedsLimit() {
    return (balance > limit);
}

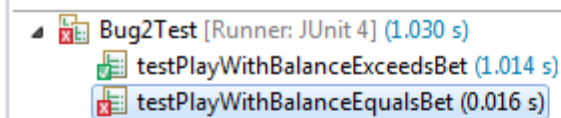
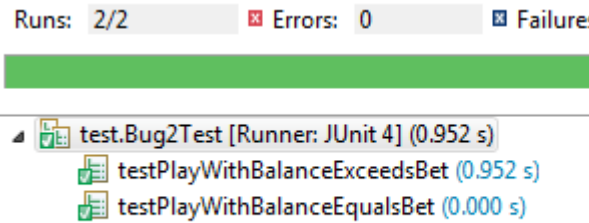
public boolean balanceExceedsLimitBy(int amount) {
    return (balance - amount >= limit);
}
```

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 (`balanceExceedsLimitBy(bet) = false`), the exception would be called, and it is not correct. So if changing the code, this may fix potential problem.

```
public void takeBet(int bet) {
    if (bet < 0) throw new IllegalArgumentException("Bet cannot be zero or negative.");
    if (!balanceExceedsLimitBy(bet)) throw new IllegalArgumentException("Placing bet would go below limit.");
    balance = balance - bet;
}
```

3.6 Result

3.6.1 Bug2Test.java

BEFORE	AFTER
 <p>Bug2Test [Runner: JUnit 4] (1.030 s)</p> <ul style="list-style-type: none">testPlayWithBalanceExceedsBet (1.014 s)testPlayWithBalanceEqualsBet (0.016 s) <p>Failure Trace</p> <pre>java.lang.IllegalArgumentException: Placing bet would go at Player.takeBet(Player.java:35) at Game.playRound(Game.java:32) at Bug2Test.testPlayWithBalanceEqualsBet(Bug2Test.java:</pre>	 <p>Runs: 2/2 Errors: 0 Failures:</p> <p>test.Bug2Test [Runner: JUnit 4] (0.952 s)</p> <ul style="list-style-type: none">testPlayWithBalanceExceedsBet (0.952 s)testPlayWithBalanceEqualsBet (0.000 s)

BEFORE	AFTER
<p>Automated Test for Bug 2: Player cannot reach b Start Game</p> <p>October starts with balance 6, limit 0 Rolled ANCHOR, ANCHOR, ANCHOR October bet 5 on HEART Balance now 1 Limit: 0</p> <p>Winnings for this bet:0</p> <p>Automated Test for Bug 2: Player cannot reach b Start Game</p> <p>October starts with balance 5, limit 0 Rolled ANCHOR, ANCHOR, ANCHOR October bet 5 on HEART</p>	<p>Automated Test for Bug 2: Player cannot rea Start Game</p> <p>Test with balance exceeds bet October starts with balance 6, limit 0 Rolled ANCHOR, ANCHOR, ANCHOR October bet 5 on HEART Balance now 1 Limit: 0</p> <p>Winnings for this bet:0</p> <p>Automated Test for Bug 2: Player cannot rea Start Game</p> <p>Test with balance equals bet October starts with balance 5, limit 0 Rolled ANCHOR, ANCHOR, ANCHOR October bet 5 on HEART Balance now 0 Limit: 0</p> <p>Winnings for this bet:0</p>

3.6.2 Console from Main

BEFORE	AFTER
<p>Turn 58: Fred bet 5 on CLUB Rolled DIAMOND, ANCHOR, DIAMOND Fred lost, balance now 5</p> <p>58 turns later. End Game 99: Fred now has balance 5</p>	<p>Turn 373: Fred bet 5 on CROWN Rolled ANCHOR, HEART, HEART Fred lost, balance now 5</p> <p>Turn 374: Fred bet 5 on CLUB Rolled ANCHOR, HEART, HEART Fred lost, balance now 0</p> <p>374 turns later. End Game 99: Fred now has balance 0</p>

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

4.1 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
1.	Run Main.java with player details: Player name = "Fred" Balance = 100 Limit = 0	Console opens and results for games are displayed in it.	Pass
2.	Look at each individual line of rolls	There are at minimum two different rolls.	Fail
3.	Repeat Steps 1-2.	Same as Steps 1-2.	Fail

Examples of bugs

<p>Run 1</p> <pre>Start Game Fred starts with balance 100, limit 0 Turn 1: Fred bet 5 on CLUB Rolled CROWN, CROWN, CLUB Fred won 5, balance now 105 Turn 2: Fred bet 5 on CROWN Rolled CROWN, CROWN, CLUB Fred won 10, balance now 115 Turn 3: Fred bet 5 on CROWN Rolled CROWN, CROWN, CLUB Fred won 10, balance now 125 Turn 4: Fred bet 5 on CLUB Rolled CROWN, CROWN, CLUB Fred won 5, balance now 130</pre>	<p>Run 2</p> <pre>Start Game Fred starts with balance 100, limit 0 Turn 1: Fred bet 5 on HEART Rolled ANCHOR, HEART, HEART Fred won 10, balance now 110 Turn 2: Fred bet 5 on CLUB Rolled ANCHOR, HEART, HEART Fred lost, balance now 105 Turn 3: Fred bet 5 on CROWN Rolled ANCHOR, HEART, HEART Fred lost, balance now 100 Turn 4: Fred bet 5 on HEART Rolled ANCHOR, HEART, HEART Fred won 10, balance now 110</pre>
---	--

4.2 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
1.	Run Bug3Test.java	Console opens and results for games are displayed in it.	Pass
2.	Look at the result of 10 turns rolling	There are at minimum two different rolls.	Fail
3.	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

```
Test Dice roll values of three dices
Original dice values: CLUB , CLUB , HEART
Results of 10 turns rolling:
Turn 1: CLUB , CLUB , HEART
Turn 2: CLUB , CLUB , HEART
Turn 3: CLUB , CLUB , HEART
Turn 4: CLUB , CLUB , HEART
Turn 5: CLUB , CLUB , HEART
Turn 6: CLUB , CLUB , HEART
Turn 7: CLUB , CLUB , HEART
Turn 8: CLUB , CLUB , HEART
Turn 9: CLUB , CLUB , HEART
Turn 10: CLUB , CLUB , HEART
```

Run 2: Fail

```
Test Dice roll values of three dices
Original dice values: CROWN , CLUB , CLUB
Results of 10 turns rolling:
Turn 1: CROWN , CLUB , CLUB
Turn 2: CROWN , CLUB , CLUB
Turn 3: CROWN , CLUB , CLUB
Turn 4: CROWN , CLUB , CLUB
Turn 5: CROWN , CLUB , CLUB
Turn 6: CROWN , CLUB , CLUB
Turn 7: CROWN , CLUB , CLUB
Turn 8: CROWN , CLUB , CLUB
Turn 9: CROWN , CLUB , CLUB
Turn 10: CROWN , CLUB , CLUB
```

Run 3: Fail

```
Test Dice roll values of three dices
Original dice values: CLUB , CLUB , DIAMOND
Results of 10 turns rolling:
Turn 1: CLUB , CLUB , DIAMOND
Turn 2: CLUB , CLUB , DIAMOND
Turn 3: CLUB , CLUB , DIAMOND
Turn 4: CLUB , CLUB , DIAMOND
Turn 5: CLUB , CLUB , DIAMOND
Turn 6: CLUB , CLUB , DIAMOND
Turn 7: CLUB , CLUB , DIAMOND
Turn 8: CLUB , CLUB , DIAMOND
Turn 9: CLUB , CLUB , DIAMOND
Turn 10: CLUB , CLUB , DIAMOND
```

4.3 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:

```
BufferedReader console = new BufferedReader(new InputStreamReader(System.in));

Dice d1 = new Dice();
Dice d2 = new Dice();
Dice d3 = new Dice();

Player player = new Player("Fred", 100);
Game game = new Game(d1, d2, d3);
List<DiceValue> cdv = game.getDiceValues();
```

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

```

while (player.balanceExceedsLimitBy(bet) && player.getBalance() < 200)
{
    turn++;
    DiceValue pick = DiceValue.getRandom();

    System.out.printf("Turn %d: %s bet %d on %s\n",
        turn, player.getName(), bet, pick);

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

    System.out.printf("Rolled %s, %s, %s\n",
        cdv.get(0), cdv.get(1), cdv.get(2));

    if (winnings > 0) {
        System.out.printf("%s won %d, balance now %d\n\n",
            player.getName(), winnings, player.getBalance());
        winCount++;
    }
}

```

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

```

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 ++;
        }
    }

    int winnings = matches * bet;

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

    else player.takeBet(bet);
    return winnings;
}

```

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

```

public Dice() {
    value = DiceValue.getRandom();
}

public DiceValue getValue() {
    return value;
}

public DiceValue roll() {
    return DiceValue.getRandom();
}

```

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.

4.4 Hypotheses

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

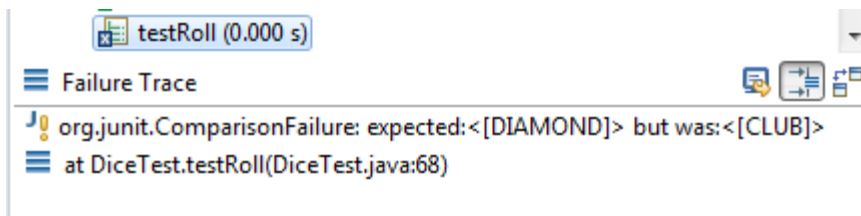
- **Hypothesis 1:** the *roll()* method does not change the value of DiceValue (incorrect).
- **Hypothesis 2:** the result of each turn is decided by comparing the pick with the *value* of the dice (correct).
- **Hypothesis 3:** three dices are assigned with each value and reused throughout the game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

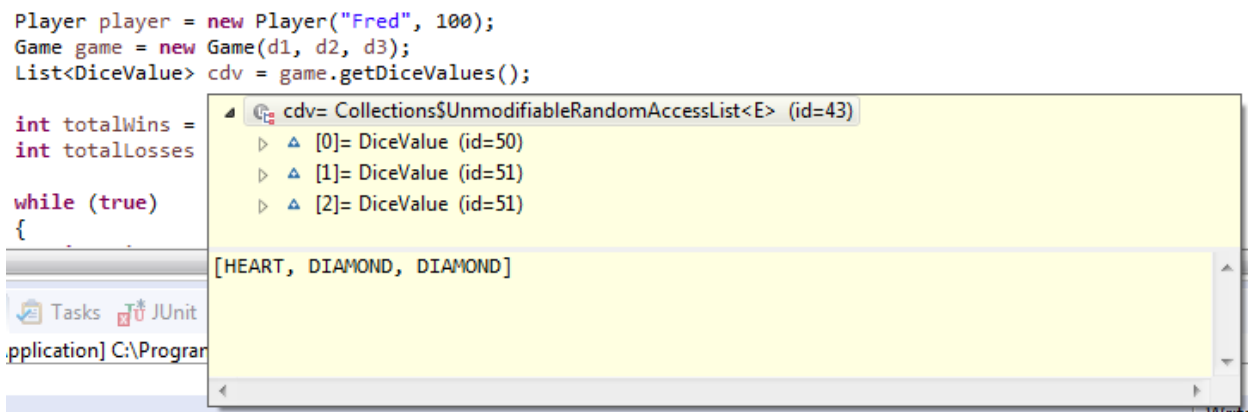
```

@Test
public void testRoll() {
    System.out.println("\nTest Dice roll values ");
    dice = new Dice();
    value = dice.roll().toString();
    System.out.println("Expected: " + value + " | Actual result: " + dice.getValue().toString());
    assertEquals(value, dice.getValue().toString());
}

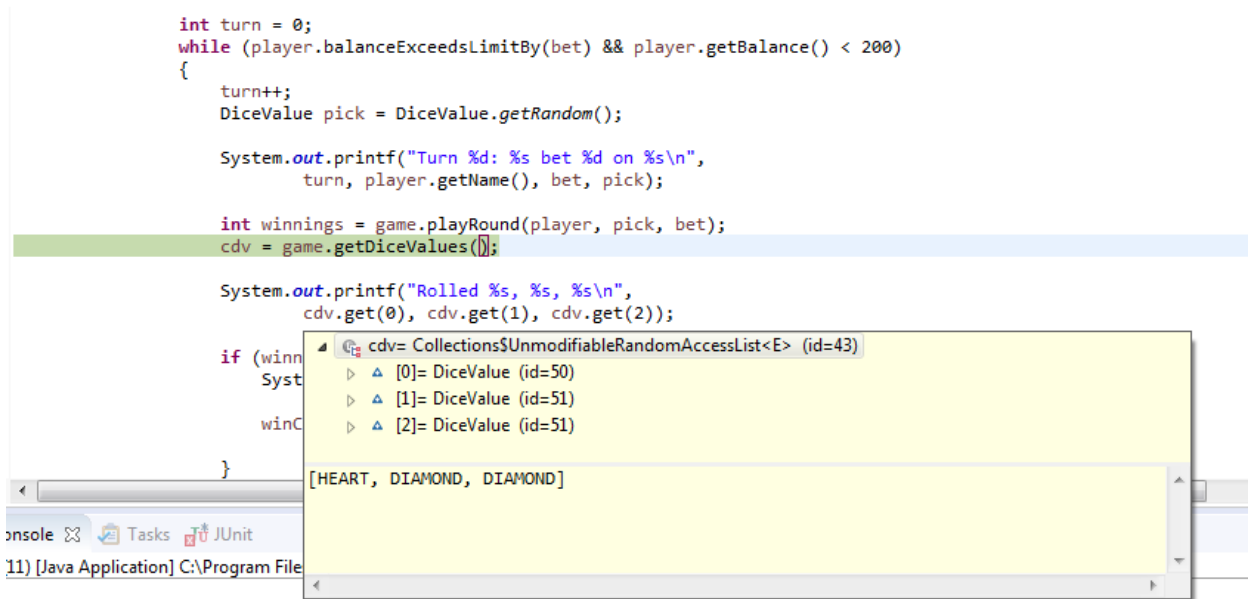
```



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:



4.5 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.

```

public class Dice {
    private DiceValue value;

    public Dice() {
        value = DiceValue.getRandom();
    }

    public DiceValue getValue() {
        return value;
    }

    public DiceValue roll() {
        // return DiceValue.getRandom();
        value = DiceValue.getRandom();
        return value;
    }

    public String toString() {
        return value.toString();
    }
}

```

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.

4.6 Result

4.6.1 Bug3Test

Different results are showed

BEFORE	AFTER
Test Dice roll values of three dices Original dice values: CLUB , CLUB , HEART Results of 10 turns rolling: Turn 1: CLUB , CLUB , HEART Turn 2: CLUB , CLUB , HEART Turn 3: CLUB , CLUB , HEART Turn 4: CLUB , CLUB , HEART Turn 5: CLUB , CLUB , HEART Turn 6: CLUB , CLUB , HEART Turn 7: CLUB , CLUB , HEART Turn 8: CLUB , CLUB , HEART Turn 9: CLUB , CLUB , HEART Turn 10: CLUB , CLUB , HEART	Test Dice roll values of three dices Original dice values: DIAMOND , CROWN , CLUB Results of 10 turns rolling: Turn 1: ANCHOR , CROWN , CROWN Turn 2: ANCHOR , CROWN , HEART Turn 3: CLUB , CLUB , DIAMOND Turn 4: DIAMOND , HEART , ANCHOR Turn 5: CLUB , HEART , ANCHOR Turn 6: DIAMOND , CROWN , CROWN Turn 7: CLUB , CROWN , ANCHOR Turn 8: CROWN , CROWN , ANCHOR Turn 9: CLUB , CROWN , CROWN Turn 10: DIAMOND , CROWN , ANCHOR

4.6.2 Console from Main

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

BEFORE	AFTER
<pre>Start Game Fred starts with balance 100, limit 0 Turn 1: Fred bet 5 on CLUB Rolled CROWN, CROWN, CLUB Fred won 5, balance now 105 Turn 2: Fred bet 5 on CROWN Rolled CROWN, CROWN, CLUB Fred won 10, balance now 115 Turn 3: Fred bet 5 on CROWN Rolled CROWN, CROWN, CLUB Fred won 10, balance now 125 Turn 4: Fred bet 5 on CLUB Rolled CROWN, CROWN, CLUB Fred won 5, balance now 130</pre>	<pre>Start Game Fred starts with balance 100, limit 0 Turn 1: Fred bet 5 on CROWN Rolled DIAMOND, DIAMOND, CLUB Fred lost, balance now 95 Turn 2: Fred bet 5 on DIAMOND Rolled DIAMOND, DIAMOND, DIAMOND Fred won 15, balance now 110 Turn 3: Fred bet 5 on DIAMOND Rolled DIAMOND, CLUB, DIAMOND Fred won 10, balance now 120 Turn 4: Fred bet 5 on DIAMOND Rolled HEART, DIAMOND, DIAMOND Fred won 10, balance now 130 Turn 5: Fred bet 5 on ANCHOR Rolled HEART, ANCHOR, HEART Fred won 5, balance now 135 Turn 6: Fred bet 5 on DIAMOND Rolled HEART, CROWN, CROWN Fred lost, balance now 130</pre>

5 Bug 4 - SPADE is ever rolled or guessed

5.1 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
1.	Run Main.java with player details: Player name = "Fred" Balance = 100 Limit = 0	Console opens and results for games are displayed in it.	Pass
2.	Look at each individual line of rolls	There are at minimum two SPADE.	Fail
3.	Repeat Steps 1-2.	Same as Steps 1-2.	Fail

Examples of bugs

Run 1	Run 2
<p>Start Game Fred starts with balance 100, limit 0 Turn 1: Fred bet 5 on CROWN Rolled DIAMOND, DIAMOND, CLUB Fred lost, balance now 95</p> <p>Turn 2: Fred bet 5 on DIAMOND Rolled DIAMOND, DIAMOND, DIAMOND Fred won 15, balance now 110</p> <p>Turn 3: Fred bet 5 on DIAMOND Rolled DIAMOND, CLUB, DIAMOND Fred won 10, balance now 120</p> <p>Turn 4: Fred bet 5 on DIAMOND Rolled HEART, DIAMOND, DIAMOND Fred won 10, balance now 130</p> <p>Turn 5: Fred bet 5 on ANCHOR Rolled HEART, ANCHOR, HEART Fred won 5, balance now 135</p> <p>Turn 6: Fred bet 5 on DIAMOND Rolled HEART, CROWN, CROWN Fred lost, balance now 130</p>	<p>Start Game Fred starts with balance 100, limit 0 Turn 1: Fred bet 5 on CROWN Rolled ANCHOR, ANCHOR, CROWN Fred won 5, balance now 105</p> <p>Turn 2: Fred bet 5 on HEART Rolled CLUB, CROWN, CROWN Fred lost, balance now 100</p> <p>Turn 3: Fred bet 5 on CLUB Rolled CLUB, HEART, HEART Fred won 5, balance now 105</p> <p>Turn 4: Fred bet 5 on ANCHOR Rolled DIAMOND, CLUB, DIAMOND Fred lost, balance now 100</p> <p>Turn 5: Fred bet 5 on CROWN Rolled DIAMOND, DIAMOND, CROWN Fred won 5, balance now 105</p> <p>Turn 6: Fred bet 5 on DIAMOND Rolled CROWN, ANCHOR, CLUB Fred lost, balance now 100</p>

5.2 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
1.	Run Bug4Test.java	Console opens and results for games are displayed in it.	Pass
2.	Check result of testPickSpade in Failure Trace	JUnit test should be no error and no failure	Fail
3.	Check result of testPickSpade in Console	Some Spades	Fail

	TEST STEP	EXPECTED TEST RESULTS	RESULT
4.	Check result of testRollSpade in Failure Trace	JUnit test should be no error and no failure	Fail
5.	Check result of testRollSpadeh in Console	Some Spades	Fail



Results

1. Pick Space: FAIL

 Bug4Test [Runner: JUnit 4] (0.015 s)
 testPickSpade (0.015 s)
 testRollSpade (0.000 s)

 Failure Trace

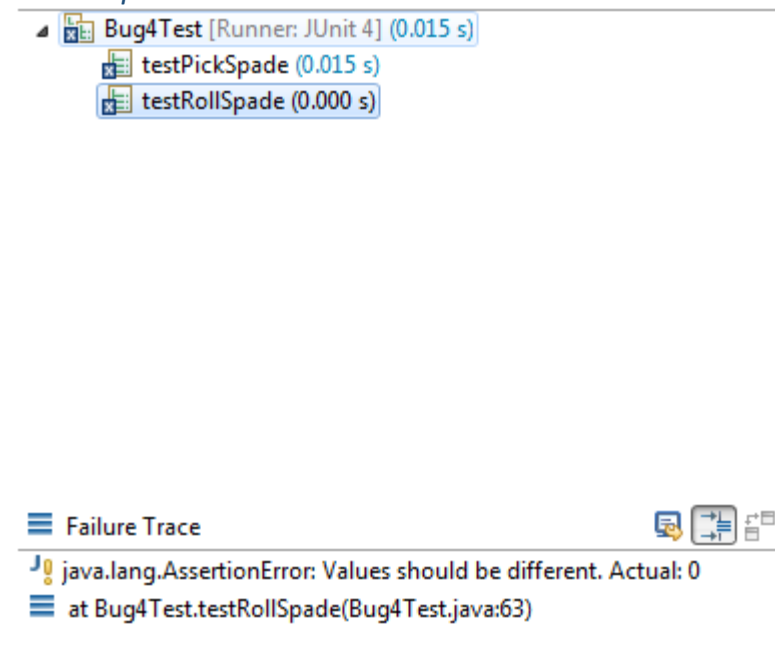


 java.lang.AssertionError: Values should be different. Actual: 0
 at Bug4Test.testPickSpade(Bug4Test.java:42)

```

Test pick SPADE
Turn 1: ANCHOR
Turn 2: ANCHOR
Turn 3: CROWN
Turn 4: CLUB
Turn 5: CROWN
Turn 6: ANCHOR
Turn 7: HEART
Turn 8: CLUB
Turn 9: CLUB
Turn 10: CLUB
Turn 11: HEART
Turn 12: ANCHOR
Turn 13: CLUB
Turn 14: CROWN
Turn 15: CLUB
Turn 16: DIAMOND
Turn 17: CROWN
Turn 18: CROWN
Turn 19: DIAMOND
Turn 20: CROWN
  
```

2. Roll Space: FAIL



Bug4Test [Runner: JUnit 4] (0.015 s)

- testPickSpade (0.015 s)
- testRollSpade (0.000 s)

Failure Trace

java.lang.AssertionError: Values should be different. Actual: 0
at Bug4Test.testRollSpade(Bug4Test.java:63)

```
Test roll SPADE
Turn 1: ANCHOR , ANCHOR , CLUB
Turn 2: CLUB , CLUB , DIAMOND
Turn 3: CROWN , CROWN , CLUB
Turn 4: DIAMOND , ANCHOR , ANCHOR
Turn 5: CROWN , CLUB , ANCHOR
Turn 6: CROWN , DIAMOND , CLUB
Turn 7: HEART , ANCHOR , CLUB
Turn 8: CROWN , ANCHOR , HEART
Turn 9: DIAMOND , CLUB , CLUB
Turn 10: HEART , ANCHOR , HEART
Turn 11: DIAMOND , CROWN , DIAMOND
Turn 12: ANCHOR , DIAMOND , CLUB
Turn 13: CLUB , CLUB , ANCHOR
Turn 14: CROWN , CROWN , DIAMOND
Turn 15: ANCHOR , CLUB , CLUB
Turn 16: ANCHOR , DIAMOND , HEART
Turn 17: DIAMOND , CROWN , DIAMOND
Turn 18: ANCHOR , HEART , CROWN
Turn 19: ANCHOR , HEART , HEART
Turn 20: ANCHOR , ANCHOR , CLUB
```

5.3 Tracing

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

```

public class Dice {

    private DiceValue value;

    public Dice() {
        value = DiceValue.getRandom();
    }

    public DiceValue getValue() {
        return value;
    }

    public DiceValue roll() {
        // return DiceValue.getRandom();
        value = DiceValue.getRandom();
        return value;
    }

    public String toString() {
        return value.toString();
    }
}

```

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

```

while (player.balanceExceedsLimitBy(bet) && player.getBalance() < 200)
{
    turn++;
    DiceValue pick = DiceValue.getRandom();
}

```

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

```

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

```

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.

```

public enum DiceValue {
    CROWN, ANCHOR, HEART, DIAMOND, CLUB, SPADE;
}

```

5.4 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 *nextInt()* of *getRandom()* method.

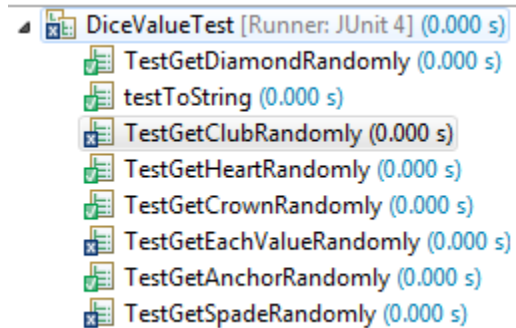
The testing of the hypothesis can also be done by replicating the bugs if changing the codes:

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.

```
public enum DiceValue {  
    CROWN, ANCHOR, HEART, DIAMOND, SPADE, CLUB;  
}
```

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

```
Start Game  
Fred starts with balance 100, limit 0  
Turn 1: Fred bet 5 on ANCHOR  
Rolled HEART, HEART, HEART  
Fred lost, balance now 95  
  
Turn 2: Fred bet 5 on HEART  
Rolled DIAMOND, CROWN, CROWN  
Fred lost, balance now 90  
  
Turn 3: Fred bet 5 on HEART  
Rolled CROWN, DIAMOND, HEART  
Fred won 5, balance now 95  
  
Turn 4: Fred bet 5 on CROWN  
Rolled HEART, CROWN, DIAMOND  
Fred won 5, balance now 100  
  
Turn 5: Fred bet 5 on HEART  
Rolled ANCHOR, ANCHOR, CROWN  
Fred lost, balance now 95  
  
Turn 6: Fred bet 5 on ANCHOR  
Rolled DIAMOND, CROWN, DIAMOND  
Fred lost, balance now 90
```



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.

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

The console confirms the error:

```

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

Turn 2: Fred bet 5 on CROWN
Rolled CROWN, CROWN, CROWN
Fred won 15, balance now 120

Turn 3: Fred bet 5 on ANCHOR
Rolled ANCHOR, CROWN, ANCHOR
Fred won 10, balance now 130

Turn 4: Fred bet 5 on ANCHOR
Rolled CROWN, CROWN, CROWN
Fred lost, balance now 125

Turn 5: Fred bet 5 on CROWN
Rolled ANCHOR, ANCHOR, ANCHOR
Fred lost, balance now 120

Turn 6: Fred bet 5 on ANCHOR
Rolled ANCHOR, CROWN, ANCHOR
Fred won 10, balance now 130

Turn 7: Fred bet 5 on ANCHOR
Rolled ANCHOR, ANCHOR, ANCHOR
Fred won 15, balance now 145

```

5.5 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:

```

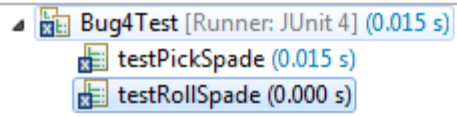
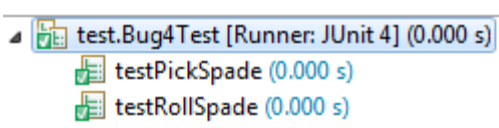
public static DiceValue getRandom() {
    int random = RANDOM.nextInt(DiceValue.SPADE.ordinal() + 1);
    return values()[random];
}

```

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).

5.6 Result

5.6.1 Bug4Test

BEFORE	AFTER
 <p>Failure Trace</p> <pre> java.lang.AssertionError: Values should be different. A at Bug4Test.testRollSpade(Bug4Test.java:63) </pre>	
<p>Roll SPADE</p> <p>Test roll SPADE</p> <p>Turn 1: ANCHOR , ANCHOR , CLUB</p> <p>Turn 2: CLUB , CLUB , DIAMOND</p> <p>Turn 3: CROWN , CROWN , CLUB</p> <p>Turn 4: DIAMOND , ANCHOR , ANCHOR</p> <p>Turn 5: CROWN , CLUB , ANCHOR</p> <p>Turn 6: CROWN , DIAMOND , CLUB</p> <p>Turn 7: HEART , ANCHOR , CLUB</p> <p>Turn 8: CROWN , ANCHOR , HEART</p> <p>Turn 9: DIAMOND , CLUB , CLUB</p> <p>Turn 10: HEART , ANCHOR , HEART</p> <p>Turn 11: DIAMOND , CROWN , DIAMOND</p> <p>Turn 12: ANCHOR , DIAMOND , CLUB</p> <p>Turn 13: CLUB , CLUB , ANCHOR</p> <p>Turn 14: CROWN , CROWN , DIAMOND</p> <p>Turn 15: ANCHOR , CLUB , CLUB</p> <p>Turn 16: ANCHOR , DIAMOND , HEART</p> <p>Turn 17: DIAMOND , CROWN , DIAMOND</p> <p>Turn 18: ANCHOR , HEART , CROWN</p> <p>Turn 19: ANCHOR , HEART , HEART</p> <p>Turn 20: ANCHOR , ANCHOR , CLUB</p>	<p>Roll SPADE</p> <p>Automated Test for Bug 4: SPADE is never picked</p> <p>Test roll SPADE</p> <p>Turn 1: DIAMOND , ANCHOR , CROWN</p> <p>Turn 2: ANCHOR , CLUB , SPADE</p> <p>Turn 3: CROWN , HEART , DIAMOND</p> <p>Turn 4: DIAMOND , SPADE , CLUB</p> <p>Turn 5: SPADE , SPADE , CLUB</p> <p>Turn 6: ANCHOR , CROWN , CLUB</p> <p>Turn 7: HEART , ANCHOR , HEART</p> <p>Turn 8: SPADE , DIAMOND , ANCHOR</p> <p>Turn 9: DIAMOND , ANCHOR , ANCHOR</p> <p>Turn 10: SPADE , CLUB , HEART</p> <p>Turn 11: ANCHOR , SPADE , SPADE</p> <p>Turn 12: SPADE , HEART , CROWN</p> <p>Turn 13: CROWN , SPADE , DIAMOND</p> <p>Turn 14: DIAMOND , DIAMOND , CROWN</p> <p>Turn 15: CROWN , CROWN , CROWN</p> <p>Turn 16: ANCHOR , SPADE , CROWN</p> <p>Turn 17: CLUB , DIAMOND , SPADE</p> <p>Turn 18: SPADE , HEART , SPADE</p> <p>Turn 19: ANCHOR , HEART , DIAMOND</p> <p>Turn 20: CLUB , ANCHOR , HEART</p>

BEFORE	AFTER
<p>Pick SPADE</p> <p>Test pick SPADE</p> <p>Turn 1: ANCHOR</p> <p>Turn 2: ANCHOR</p> <p>Turn 3: CROWN</p> <p>Turn 4: CLUB</p> <p>Turn 5: CROWN</p> <p>Turn 6: ANCHOR</p> <p>Turn 7: HEART</p> <p>Turn 8: CLUB</p> <p>Turn 9: CLUB</p> <p>Turn 10: CLUB</p> <p>Turn 11: HEART</p> <p>Turn 12: ANCHOR</p> <p>Turn 13: CLUB</p> <p>Turn 14: CROWN</p> <p>Turn 15: CLUB</p> <p>Turn 16: DIAMOND</p> <p>Turn 17: CROWN</p> <p>Turn 18: CROWN</p> <p>Turn 19: DIAMOND</p> <p>Turn 20: CROWN</p>	<p>Pick SPADE</p> <p>Automated Test for Bug 4: SPADE is never picked</p> <p>Test pick SPADE</p> <p>Turn 1: DIAMOND</p> <p>Turn 2: HEART</p> <p>Turn 3: SPADE</p> <p>Turn 4: CROWN</p> <p>Turn 5: CROWN</p> <p>Turn 6: SPADE</p> <p>Turn 7: CROWN</p> <p>Turn 8: SPADE</p> <p>Turn 9: CROWN</p> <p>Turn 10: CLUB</p> <p>Turn 11: CROWN</p> <p>Turn 12: DIAMOND</p> <p>Turn 13: SPADE</p> <p>Turn 14: CROWN</p> <p>Turn 15: CROWN</p> <p>Turn 16: CLUB</p> <p>Turn 17: CROWN</p> <p>Turn 18: HEART</p> <p>Turn 19: CLUB</p> <p>Turn 20: CROWN</p>

5.6.2 Console from Main

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

BEFORE	AFTER
<p>Start Game</p> <p>Fred starts with balance 100, limit 0</p> <p>Turn 1: Fred bet 5 on CROWN</p> <p>Rolled DIAMOND, DIAMOND, CLUB</p> <p>Fred lost, balance now 95</p> <p>Turn 2: Fred bet 5 on DIAMOND</p> <p>Rolled DIAMOND, DIAMOND, DIAMOND</p> <p>Fred won 15, balance now 110</p> <p>Turn 3: Fred bet 5 on DIAMOND</p> <p>Rolled DIAMOND, CLUB, DIAMOND</p> <p>Fred won 10, balance now 120</p> <p>Turn 4: Fred bet 5 on DIAMOND</p> <p>Rolled HEART, DIAMOND, DIAMOND</p> <p>Fred won 10, balance now 130</p> <p>Turn 5: Fred bet 5 on ANCHOR</p> <p>Rolled HEART, ANCHOR, HEART</p> <p>Fred won 5, balance now 135</p> <p>Turn 6: Fred bet 5 on DIAMOND</p> <p>Rolled HEART, CROWN, CROWN</p> <p>Fred lost, balance now 130</p>	<p>Start Game</p> <p>Fred starts with balance 100, limit 0</p> <p>Turn 1: Fred bet 5 on ANCHOR</p> <p>Rolled CLUB, DIAMOND, CROWN</p> <p>Fred lost, balance now 95</p> <p>Turn 2: Fred bet 5 on SPADE</p> <p>Rolled HEART, CROWN, SPADE</p> <p>Fred won 5, balance now 100</p> <p>Turn 3: Fred bet 5 on CROWN</p> <p>Rolled DIAMOND, SPADE, CLUB</p> <p>Fred lost, balance now 95</p> <p>Turn 4: Fred bet 5 on HEART</p> <p>Rolled CLUB, CLUB, DIAMOND</p> <p>Fred lost, balance now 90</p> <p>Turn 5: Fred bet 5 on SPADE</p> <p>Rolled CLUB, CLUB, CROWN</p> <p>Fred lost, balance now 85</p> <p>Turn 6: Fred bet 5 on DIAMOND</p> <p>Rolled CROWN, HEART, SPADE</p> <p>Fred lost, balance now 80</p>

6 Bug 5 - Odds of game are incorrect

6.1 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
1.	Undo the changing of SPADE as in Bug 4	Only bugs 1, 2, 3 are fixed.	Pass
2.	Run Main.java	Console opens and results for games are displayed in it.	Pass
3.	Look at the win count line	There should be ratio of 0.42	Fail (ratio =0.48)
4.	Repeat Steps 2-3.	Same as Steps 2-3.	Fail

Examples of bugs

Run 1: FAIL

```
113 turns later.  
End Game 99: Fred now has balance 200  
  
Win count = 9039, Lose Count = 9320, 0.49
```

Run 2: FAIL

```
188 turns later.  
End Game 99: Fred now has balance 200  
  
Win count = 9906, Lose Count = 10538, 0.48
```

Run 3: FAIL

349 turns later.

End Game 99: Fred now has balance 200

Win count = 9084, Lose Count = 9370, 0.49

6.2 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
1.	Run Bug4Test.java	Console opens and results for games are displayed in it.	Pass
2.	Check win ratio in Console	Should be between 41 - 43%	Fail
3.	Check Failure Trace	JUnit should show no error or failure	Fail
4.	Rerun the test 2 times	Same like step 2-3	Fail

Result

Run 1: FAIL

Bug5Test [Runner: JUnit 4] (0.673 s)
testIncorrectOddsRatio (0.673 s)

Failure Trace

java.lang.AssertionError
at Bug5Test.testIncorrectOddsRatio(Bug5Test.java:108)

Turn 999
Rolled HEART, CROWN, HEART
October bet 5 on CROWN
Balance now 3185 | Limit: 0
Winnings for this bet: 5
October won 5, balance now 3185

Turn 1000
Rolled DIAMOND, CLUB, CLUB
October bet 5 on CROWN
Balance now 3180 | Limit: 0
Winnings for this bet: 0
October lost, balance now 3180

GAME OVER
Total Win count = 509, Total Loss Count = 491
Overall win rate = 50.9%

Run 2: FAIL

Bug5Test [Runner: JUnit 4] (0.500 s)
testIncorrectOddsRatio (0.500 s)

Failure Trace

java.lang.AssertionError
at Bug5Test.testIncorrectOddsRatio(Bug5Test.java:108)

Turn 999
Rolled DIAMOND, CROWN, CLUB
October bet 5 on CROWN
Balance now 3135 | Limit: 0
Winnings for this bet: 5
October won 5, balance now 3135

Turn 1000
Rolled CROWN, ANCHOR, ANCHOR
October bet 5 on CROWN
Balance now 3135 | Limit: 0
Winnings for this bet: 5
October won 5, balance now 3135

GAME OVER
Total Win count = 505, Total Loss Count = 495
Overall win rate = 50.5%

Run 3: FAIL

Runs: 1/1 Errors: 0 Failures: 1

Bug5Test [Runner: JUnit 4] (0.610 s)
testIncorrectOddsRatio (0.610 s)

Failure Trace

java.lang.AssertionError
at Bug5Test.testIncorrectOddsRatio(Bug5Test.java:108)

```
Turn 999
Rolled CROWN, CROWN, CLUB
October bet 5 on CROWN
Balance now 3140 | Limit: 0
Winnings for this bet: 10
October won 10, balance now 3140
```

```
Turn 1000
Rolled ANCHOR, CLUB, ANCHOR
October bet 5 on CROWN
Balance now 3135 | Limit: 0
Winnings for this bet: 0
October lost, balance now 3135
```

```
GAME OVER
Total Win count = 504, Total Loss Count = 496
Overall win rate = 50.4%
```

6.3 Tracing

The tracing of this issue should first start at considering the winning ratio of n values in the game. There are n^3 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 $(n - 1)^3$. The winning ratio is calculated as:

$$\text{winning ratio} = 1 - \frac{(n-1)^3}{n^3}$$

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

$$\text{winning ratio} = 1 - \frac{(6-1)^3}{6^3} = 1 - \frac{125}{216} = 0.4212$$

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.

$$\text{winning ratio} = 1 - \frac{(5-1)^3}{5^3} = 1 - \frac{64}{125} = 0.488$$

6.4 Hypothesis

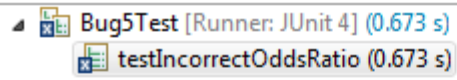
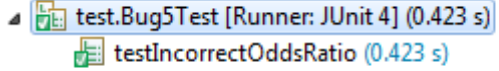
The root of this bug is the same as bug 4 above - no SPADE in the game. So by adding the SPADE, the winning will be resolved.

6.5 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.

6.6 Result

6.6.1 Bug5Test

BEFORE	AFTER
 <pre> Failure Trace java.lang.AssertionError at Bug5Test.testIncorrectOddsRatio(Bug5Test.java:10) </pre>	

BEFORE	AFTER
Turn 999 Rolled HEART, CROWN, HEART October bet 5 on CROWN Balance now 3185 Limit: 0 Winnings for this bet: 5 October won 5, balance now 3185 Turn 1000 Rolled DIAMOND, CLUB, CLUB October bet 5 on CROWN Balance now 3180 Limit: 0 Winnings for this bet: 0 October lost, balance now 3180 GAME OVER Total Win count = 509, Total Loss Count = 491 Overall win rate = 50.9%	Turn 998 Rolled CROWN, DIAMOND, CLUB October bet 5 on CROWN Balance now 99715 Limit: 0 Winnings for this bet: 5 October won 5, balance now 99715 Turn 999 Rolled DIAMOND, ANCHOR, SPADE October bet 5 on CROWN Balance now 99710 Limit: 0 Winnings for this bet: 0 October lost, balance now 99710 Turn 1000 Rolled DIAMOND, DIAMOND, DIAMOND October bet 5 on CROWN Balance now 99705 Limit: 0 Winnings for this bet: 0 October lost, balance now 99705 GAME OVER Total Win count = 430, Total Loss Count = 570 Overall win rate = 43.0%

6.6.2 Main Console

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

Run 1:

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

Win count = 8700, Lose Count = 12054, 0.42
```

Run 2

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

Win count = 9993, Lose Count = 13496, 0.43
```

Run 3

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

Win count = 10012, Lose Count = 13714, 0.42
```