Table of Contents

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 Bug 1 - 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.