Bug reporting and investigation

Refer to program output: Sample1.txt (“sample 1”), Sample2.txt (“sample 2”), and Sample3.txt (“sample 3”).

Bugs

[1. One match pays out 0 instead of the expected equal to the bet 2](#_Toc463620726)

[Summary 2](#_Toc463620727)

[Examples 2](#_Toc463620728)

[Hypotheses 2](#_Toc463620729)

[Investigation 2](#_Toc463620730)

[2. Two matches pays out 1X bet instead of the expected 2X bet 2](#_Toc463620731)

[Summary 2](#_Toc463620732)

[Examples 2](#_Toc463620733)

[Hypotheses 3](#_Toc463620734)

[Investigation 3](#_Toc463620735)

[3. Three matches pays out 2X bet instead of the expected 3X bet 3](#_Toc463620736)

[Summary 3](#_Toc463620737)

[Examples 3](#_Toc463620738)

[Hypotheses 3](#_Toc463620739)

[Investigation 3](#_Toc463620740)

[4. All rolls are the same in each run through 3](#_Toc463620741)

[Summary 3](#_Toc463620742)

[Examples 3](#_Toc463620743)

[Replication 3](#_Toc463620744)

[Hypotheses 3](#_Toc463620745)

[Resolution 4](#_Toc463620746)

[5. Spades are never rolled 4](#_Toc463620747)

[Summary 4](#_Toc463620748)

[Examples 4](#_Toc463620749)

[Hypotheses 4](#_Toc463620750)

[6. Fred never guesses “Spade” 4](#_Toc463620751)

[Summary 4](#_Toc463620752)

[Examples 4](#_Toc463620753)

[Hypotheses 4](#_Toc463620754)

# One match pays out 0 instead of the expected equal to the bet

## Summary

* Game pays out incorrect winnings.
* Initially found through unit testing the Game class.
* Also Bug 1 in initial bug report provided with assignment.

## Examples

1. Displayed in sample 1 on turn 1:

*Fred starts with balance 100, limit 0*

*Turn 1: Fred bet 5 on CROWN*

*Rolled HEART, HEART, CROWN*

*Fred won 5, balance now 100*

Fred got one match, so should win $5. He started with $100, so should have $105. But he only has $100 still.

1. Fails GameTest unit testing in test TestPlayRoundOneMatch.

## Replication

It isn’t really feasible to make a dedicated replication program for this bug. It would require too much manipulation of main, which has the risk of introducing other defects (for example, maybe we would have condition to only print to screen when there was exactly one match). Other bugs have replication because it is just a matter of turning messages on or off to see the results more clearly, or just adding up the wins and losses over many runs of the program. This isn’t the case here.

However, we can run the Game class unit test (GameTest.java) to simulate one match and verify that the player gets the wrong amount in their balance as a result.

Please see Bug 1 UAT Replication.docx for details.

## Hypotheses

* Same cause as bugs 2 and 3.
* The bet is being removed from the balance before the winnings are added

## Investigation

Please see: Bugs 1 2 and 3 Investigation.docx

# Two matches pays out 1X bet instead of the expected 2X bet

## Summary

* Game pays out incorrect winnings
* Initially found through unit testing the Game class (GameTest: TestPlayRoundTwoMatches())

## Examples

1. Displayed in the sample 1 on turn 3:

*Fred lost, balance now 95*

*Turn 3: Fred bet 5 on HEART*

*Rolled HEART, HEART, CROWN*

*Fred won 10, balance now 100*

Since Fred got two matches, he wins $10. He had $95 before, so, he should have $105 now. But he only has $100.

1. Fails GameTest unit testing in test TestPlayRoundTwoMatches.

## Replication

It isn’t really feasible to make a dedicated replication program for this bug. It would require too much manipulation of main, which has the risk of introducing other defects (for example, maybe we would have condition to only print to screen when there was exactly one match). Other bugs have replication because it is just a matter of turning messages on or off to see the results more clearly, or just adding up the wins and losses over many runs of the program. This isn’t the case here.

However, we can run the Game class unit test (GameTest.java) to simulate two matches and verify that the player gets the wrong amount in their balance as a result.

Please see Bug 2 UAT Replication.docx for details.

## Hypotheses

* Same cause as bugs 1 and 3.
* The bet is being removed from the balance before the winnings are added

## Investigation

Please see Bugs 1 2 and 3 Investigation.docx

# 3. Three matches pays out 2X bet instead of the expected 3X bet

## Summary

* Game pays out incorrect winnings.
* Initially found through unit testing the Game class (GameTest: TestPlayRoundThreeMatches())

## Examples

1. Fails GameTest unit testing in test TestPlayRoundThreeMatches.

## Replication

It isn’t really feasible to make a dedicated replication program for this bug. It would require too much manipulation of main, which has the risk of introducing other defects (for example, maybe we would have condition to only print to screen when there was exactly one match). Other bugs have replication because it is just a matter of turning messages on or off to see the results more clearly, or just adding up the wins and losses over many runs of the program. This isn’t the case here.

However, we can run the Game class unit test (GameTest.java) to simulate three matches and verify that the player gets the wrong amount in their balance as a result.

Please see Bug 3 UAT Replication.docx for details.

## Hypotheses

* Same cause as bugs 1 and 2.
* The bet is being removed from the balance before the winnings are added

## Investigation

Please see Bugs 1 2 and 3 Investigation.docx

# All rolls are the same in each run through

## Summary

* In each run through, the rolls are all the same.
* Initially found by running the program and observing the results.

## Examples

1. In sample 1, all the rolls are Heart, Heart, Crown (38 rolls).
2. In sample 2, all the rolls are Diamond, Diamond, Club (69 rolls).
3. In sample 3, the rolls are all Anchor, Heart, Anchor (45 rolls).

## Replication

Please see Bug 4 UAT Replication.docx

“Bug4Replication.java” executes **Main**’s **main** but takes out all the peripheral comments, leaving just the rolls so you can see more at a glance, and not see anything that isn’t important.

## Hypotheses

1. **Dice** are created only once per run of the program, and then reused for each **Game**.
2. The **value** is invariant over the life of any particular instance of **Dice**.
3. The **value** of the **Dice** is what is used as each roll & compared to the pick to determine if the player wins or not.

All hypotheses tested and found to be verified (please see Bug 4 Investigation.docx for details).

## Resolution

The **roll** method in the **Dice** class was edited. This successfully resolved the bug.

# Spades are never rolled

## Summary

* Spades are never rolled during the game.
* Initially found during unit testing of DiceValue (after 100 rolls, Spades are not produced using the function getRandom).

## Examples

1. Fails DiceValueTest unit testing in test TestGetRandomsProducesSpade
2. No spades rolled in same 1, sample 2, or sample 3.

## Hypotheses

1. As Spades are the highest ordinal in the enum DiceValue, this is probably an “out by 1” programming error. I.e. the getRandom function gets a random int up to, but not including Spades.
2. The fix will probably be to add “+ 1” in the function.
3. Idea for test: change order of enum values – if the new highest ordinal never gets rolled, this will confirm Hypothesis 1.

# Player never guesses “Spade”

## Summary

* Player never picks “Spade” to bet on.

## Examples

1. Player never picks “Spade” to bet on in sample 1, sample 2, or sample 3.

## Hypotheses

1. Same cause as bug 5.

# Odds of game incorrect

## Summary

Odds appear to be 48.8% win rate instead of the expected 42.0%.

## Examples

## Hypotheses

1. Same cause as bug 5.