**Bug 1 - Player Loses Double**

Point of Failure: At line 80 in ‘InteractiveGame’, punter.getBalance() is incorrect.

Hypothesis 1: The bug is in the presentation class where the punters balance is displayed.

Test: Visually inspect the execution of punter.getBalance() at line 80.

Prediction: Line 80 has “punter.getBalance() - bet”

Result: Hypothesis 1 is false: punter.getBalance() has no additional calculations on line 80.

Hypothesis 2: The bug is in the entity class ‘Punter’ where the punters balance is returned.

Test: Visually inspect the method getBalance() in the ‘Punter’ class.

Prediction: The getBalance() method returns “balance - bet”

Result: Hypothesis 2 is false: The balance of the punter is returned with no alteration.

Hypothesis 3: The bug is in the presentation class ‘InteractiveGame’ at the line where “Round.play()” is executed on line 69.

Prediction:

Test: Check parameters of “play()” method.

punter == (name==”John”, balance= 100, limit == 10)

dice == List<Die>(new Die(), new Die(),new Die())

pick = Face.FISH

bet == 5

Result: Hypothesis 3 is false: The all the parameters are sane so the bug must be in the “Round.play()” method.

**NOTE: After I finished this bug and moved to the next one I realised that I had started going down the wrong path from this point on. The correct path is on the next page.**

Hypothesis 4: The bug is in the control class ‘Round’ in the method “play()”.

Test: Watch the variables while stepping through the execution of the “play()” method.

Prediction: Variables of this method should show some unexpected behaviour..

Result: Hypothesis 4 is true: The bug was not directly found in the “Round” class, however method “placeBet()” in the Punter class changed the Punter object unexpectedly .

Hypothesis 5: The bug is in the entity class ‘Punter’ in the method “placeBet()”.

Test: Step through the method “placeBet()” in the ‘Punter’ class and write a test to assert the punter.

Prediction: Punter object is altered by “placeBet()” method.

Result: Hypothesis 5 is true: The “placeBet()” method changed the Punter object.

Hypothesis 6: A line in the “Punter” class is the cause of the bug and is unnesserary and should be removed.

Test: Inspect the “placeBet()” method to see if any lines are unnesserary.

Prediction: when Line 74 is removed the bug should be fixed.

Result: Test case runs successfully. Bug is removed.

**Correct path**

Hypothesis 4: The bug is in the control class ‘Round’ in the method “play()”.

Test: Watch the variables while stepping through the execution of the “play()” method.

Prediction: Variables of this method should show some unexpected behaviour..

Result: Hypothesis 4 is true: The bug was not directly found in the “Round” class, however method “loseBet()” in the Punter class changed the Punter object unexpectedly.

Hypothesis 5: The bug is in the entity class ‘Punter’ in the method “loseBet()”.

Test: Step through the method “loseBet()” in the ‘Punter’ class and write a test to assert the punter.

Prediction: Punter object is altered by “loseBet()” method.

Result: Hypothesis 5 is true: The “loseBet()” method changed the Punter object.

Hypothesis 6: A line in the “Punter” class is the cause of the bug and is unnesserary and should be removed.

Test: Inspect the “loseBet()” method to see if any lines are unnesserary.

Prediction: when Line 104 is removed the bug should be fixed.

Result: Test case runs successfully. Bug is removed.

**Bug 2 - Player doesn’t receive any winnings.**

Point of Failure: Line 76, First evidance of bug

Hypothesis 1: Incorrect parameters entering “Round.play()” method

Test: Inspect parameters of “Round.play()” method to see is they are sane.

Prediction: Parameters of “Round.play()” are sane.

Result: Hypothesis false, parameters are sane

punter == (name==”John”, balance= 100, limit == 10)

dice == List<Die>(new Die(), new Die(),new Die())

pick = Face.FISH

bet == 5

Hypothesis 2: Punters balance is not processed correctly by “Round.play()”

Test: Inspect and walkthrough the “Round.play()” method to see how punter is altered.

Prediction: Punters winnings are not added to balance.

Result: Hypothesis false

Punters bet is returned

Punters does not recieve winnings from “punter.recieveWinnings()” method

Hypothesis 3: Punters winnings are not added in “recieveWinnings()” method

Test: Inspect and walkthrough the “recieveWinnings()” method to see what happens to punter winnings.

Prediction: Punters winnings are not added to balance.

Result: Hypothesis 3 true

Punter does not get winnings added to balance, because State of punter is “NOT\_BETTING” instead of “RECEIVING\_WINNINGS”

Hypothesis 4: State of Punter is changed to incorrect value, which is the cause of the bug

Test: change the state that the “returnBet()” method changes the punter to from “NOT\_BETTING” to “RECEIVING\_WINNINGS”

Prediction: Bug will be fixed.

Result: Hypothesis 4 true

With the state changed, the bug is fixed and the test case is successful.

**Bug 3 - Player cannot reach betting limit**

Point of Failure: Line 61 in “InteractiveGame” class, unnessersary message printed.

Hypothesis 1: Check parameters entering “balanceExceedsLimitBy()” method to ensure that they are sane.

Test: Use debugger tools to “Watch” the variables

Prediction: Variable will be sane

Result: Parameters are sane

Bet == 5

Hypothesis 2 : The “balanceExceedsLimitBy()” method is returning an incorrect result

Test: Use debugger to step through the methods execution

Prediction: Method is not checking if values are equal to limit

Result Hypothesis is true.

“balanceExceedsLimitBy()” is only checking if balance remains above limit, not equal to the limit

Hypothesis 3 : Changing the “balanceExceedsLimitBy()” method to check if the balance is higher or equal to the limit will fix the bug

Test: Change the “>” to a “>=” in the “balanceExceedsLimitBy()” method

Prediction: The typo of “>” instead “>=” was the cause of the bug

Result Hypothesis is true.

“balanceExceedsLimitBy()” returns true.

**Bug 4 - Odds in the game are incorrect.**

Point of Failure: Line 90 in “BatchMode”, where the ratio is displayed

Hypothesis 1: The mechinism when rolling the dice during a game, is flawed.

Test: Inspect the “Round” class to see how the face get chosen.

Prediction: The method of choosing the face is preventing the win ratio from being 0.42.

Result:

Hypothesis 1 is correct.

The faces on the dice are not being changed each round, therefore the odds are not correct.

Hypothesis 2: The mechinism for choosing a random Face on the dice is preventing the ratio from being closer to 0.42.

Test: Step through the “Face.getRandom()” method to see how the random face get chosen.

Prediction: The function to choose the random number is giving 0.50 odds.

Result:

The steps to choose a random face seem to give just under 0.50 odds.

Hypothesis 3: A change in the “Face.getRandom()” method to choose the random face will bring the odds closer to 0.42.

Test: Modify the “Face.getRandom()” method to control the odds of winning.

Prediction: A new “Face.getRandom()” method will fix the bug

Result:

With the addition of the “Face.getWeightedResult()” method, now the odds of the game are 0.42 (+/- 0.01) and the test passes.