***BUG 1***

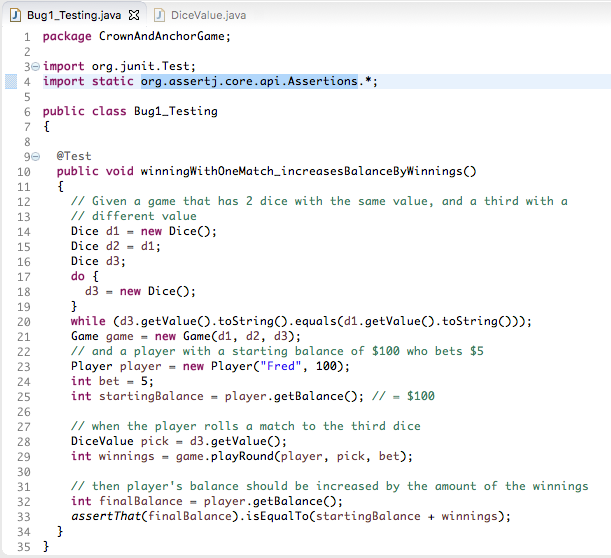
**Step 1: Replication**

|  |  |  |  |
| --- | --- | --- | --- |
| **Bug No.** | | Bug no. 1 | |
| **Detailed description** | | Balance does not increase when player wins. | |
| **Reproduction Scenario** | | Run the app, player’s balance still remains the same even after winning. | |
|  | |  | |
|  | |  | |
|  | **TEST STEP** | | **PROPOSED TEST OUTPUTS** |
|  | Run the Main Class | | The application runs normally and displays 100 turns and their status |
|  | Inspect the output | | Actual output: |
|  | Look for the time a player wins the bet | | Find the case when the bet on is rolled once |
|  | Find the difference between the balance when just won a bet and the balance prior to the bet. | | The balance stay the same even though it should be increased (because player keeps winning) |

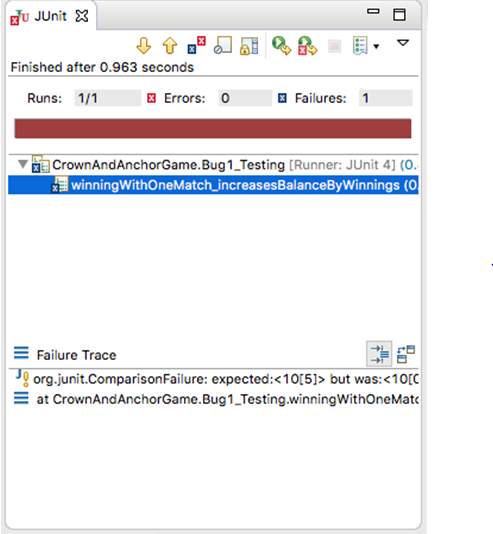
**Step 2: Simplification**

|  |  |
| --- | --- |
| **Test class name** | Bug1\_Testing |
| **Bug Description** | When rolled the dices, if one of the die is the same as a player’s bet, final balance should increase by the won bet amount |
| **Test Output** | Final balance stays the same after a won bet and does not take in winnings |

Bug1\_testing.java



JUnit Test screenshots



**Step 3: Tracing**

**Hypothesis 1**

|  |  |
| --- | --- |
| **Assumption** | The invocation of the action game.playRound() at line 29 deducts an incorrect amount from the player’s balance (more than the actual bet). |
| **Prediction** | player.takeBet(); in game.playRound()will result in balance equating to 90. |
| **Investigation** | Player’s balance is correct (== 95) after invocation of player.takeBet()  Before the bet was taken:    After the bet was taken: |
| **Conclusion** | Hypothesis is discarded |

**Hypothesis 2**

|  |  |
| --- | --- |
| **Assumption** | The method on line 29 in the test does not return the correct winning amount |
| **Prediction** | winnings will equate to zero after execution of line 29 |
| **Investigation** | winnings equates to 5 that is the correct amount  Before origin action at line 29 starting balance is on line 25 is logical (== 100)  After origin final balance on line 32 is not logical (remains == 100) although winnings is logical (== 5) |
| **Conclusion** | The hypothesis is discarded. |

**Hypothesis 3**

|  |  |
| --- | --- |
| **Assumption** | The invocation of the action game.playRound() at line 29 returns the initial bet along with the winnings. |
| **Prediction** | player.recieveWinnings(); in game.playRound()will result in balance equating to 105. |
| **Investigation** | Player’s balance is incorrect (== 100) after invocation of player.recieveWinnings()  First screenshot below shows player’s balance is logical (== 95) after the bet has been taken AND that bet == 5, winnings == 5  Second screenshot shows player’s balance is not logical (== 100) after the winnings have been returned  Third screenshot simply shows there are no more changes to player’s balance after this invocation. |
| **Conclusion** | Hypothesis is discarded and that player.recieveWinnings() must return the original bet amount along with the won amount (see line 43).  Game.java    Player.java    Bug1\_Testing.java |

**Step 4: Resolution**

|  |  |
| --- | --- |
| **Design** | In order to resolve this bug, I wrote a new method called ***refundBet().*** This method is only invoked when a player wins the bet and when it is invoked, it will return the bet amount back to the player balance |
| **Confirmation of Junit test** | Bug1\_Testing now passes after making those changes |

|  |  |
| --- | --- |
| **Confirmation of the successful resolution** | The player balance is adjusted according to the winnings amout. |

***BUG 2***

**Step 1: Replication**

|  |  |  |  |
| --- | --- | --- | --- |
| **Bug No.** | | Player cannot reach betting limit. | |
| **Detailed description** | | Limit set to 0, but game ends with player still with 5 (dollars) remaining. | |
| **Reproduction Scenario** | | Run simulation, find instance where player wins on one match and compare balance to previous balance. | |
|  | **TEST STEP** | | **PROPOSED TEST OUTPUTS** |
|  | Run application (Main Class) | | The application runs & prints 100 games and stats on the console. |
|  | Output in console. | | Application Output:  While the limit is declared as 0, the game ends while the payer’s balance is still 5: |

**Step 2: Simplification**

|  |  |
| --- | --- |
| **Test class name** | Bug2\_Testing2 |
| **Bug Description** | Given a player with a limit of zero, when he plays a round with three dice with a different value to his pick, then the player should end with a balance of zero (equal to his limit). |
| **Test Output** | An exception is thrown: “Placing bet would go below limit.” |

**Step 3: Tracing**

**Hypothesis 1**

|  |  |
| --- | --- |
| **Assumption** | The limit setter in line 24 does not actually set the player’s limit to zero |
| **Prediction** | player.setLimit will not be set to zero after execution of line 24 |
| **Investigation** | player.setLimit is in fact set to 0 (the specified limit)  See screenshots below:  Before origin limit variable (line 23) is logical (== 0)  After origin player’s limit (line 24) is also logical (== 0) |
| **Conclusion** | The hypothesis is discarded. |

**Hypothesis 2**

|  |  |
| --- | --- |
| **Assumption** | During invocation of game.playRound(),player.takeBet(bet) is called with invalid / incorrect parameter |
| **Prediction** | player.takeBet()will be called with an incorrect argument |
| **Investigation** | player.takeBet()is in fact called with the correct parameters (bet is correct value)  See screenshots below:  Before invocation, parameter is logical and correct (first screenshot)    However, within player.takeBet()there is a call to player.balanceExceedsLimitBy()that causes the exception to be thrown |
| **Conclusion** | The hypothesis is discarded but the source of the bug is located |

**Step 4: Resolution**

|  |  |
| --- | --- |
| **Design** | player.balanceExceedsLimitBy() should return true if the balance is equal to or greater than limit plus the argument. |
| **Confirmation of Junit test** | Bug2\_Testing now passes after making that changes |
| **Confirmation of successful resolution** | The game ends when the player balance is equals to zero |

***BUG 3***

**Step 1: Replication**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bug No.** | | Odds within the app seem to be incorrect. | | |
| **Detailed description** | | Crown and Anchor games have an approximate 8% bias to the house. So the win : (win+lose) ratio should approximately equal 0.42. This does not appear to be the case. | | |
| **Reproduction Scenario** | | When player wins on one match and compare balance to previous balance. | | |
|  | **TEST STEP** | | **PROPOSED TEST OUTPUTS** | |
|  | Run application Main.java | | The application runs & prints 100 games and stats on the console |
|  | Output in console. | | Application Output – Last line showing the win count, lose count and win+lose ratio. |

**Step 2: Simplification**

Note – rather than jumping to conclusions, here I decided to just quickly test 2 different aspects – first consistency of the win ratio across games, and then the distribution of the values rolled

|  |  |
| --- | --- |
| **Test class name** | Bug3\_Testing  ratioShouldBeConsistent() |
| **Bug Description** | Given many games with many rounds, when we evaluate the win ration then the ratio should be consistent across the games. |
| **Test Output** | The test fails as the win ratio is still higher than expected |

**Step 3: Tracing**

**Hypothesis 1**

|  |  |
| --- | --- |
| **Assumption** | The invocation of d1.roll() in line 64 does not return a new value, and thus does not assign a new value to dv |
| **Prediction** | dv remains the same through all of the iterations, |
| **Investigation** | After a couple of iterations, it is clear that d1.roll() does in fact update the value of dv |
| **Conclusion** | The hypothesis is discarded. |

**Step 4: Resolution**

|  |  |
| --- | --- |
| **Design** | The expression  int random = RANDOM.nextInt(DiceValue.SPADE.ordinal());  Cannot produce the value 5, which in turn means that DiceValue.getRandom() cannot return the value ‘SPADE’    by implementing the size of the Map, which will store the values |
| **Confirmation of Junit test** | Bug3\_Testing now pass after making that changes |
| **Confirmation of successful resolution** | Now shows that the win ratio is no longer one of a small set of values (i.e. 0.40 or 0.60). |