Debugging Log:

Bug 1: Game not paying out at correct level.

Code Run Through with Debugger 1:

|  |  |  |
| --- | --- | --- |
| File | Line | Possible Problem |
| Game.java | 34 | Player.takeBet(bet) method is called on every game round.  Player objects balance is reduced by the bet amount before the game round is played.  The player object is sane, until this method is called:  e.g. Player “Fred” has a balance of 95 entering the game round:  After this step, the players balance has decreased when it should not have changed until after the game round completed:  The players balance is now infected. |
| Game.java | 44 | Line where winnings are calculated. Appears the winnings amount is calculated correctly. |
| Game.java | 47 | Line where winnings are added to the player object. Noticed that the balance is already balance - bet here so, the winnings (5) are added to a balance which has already had the bet taken from it. The balance value is infected because of the issue at line 34. |

Hypothesis 1: The player.takeBet(bet) method should only be called if the player does not have any matches. The player.takeBet(bet) method should be called if the number of matches is 0.

Test 1:

Add an else clause to the if statement at line 46 in Game.java so that the method call player.takeBet(bet) only occurs when the player loses the game. Remove line 34 from program.

Ran automated test and found that the automated test no longer produced results which are valid.

Ran the main.java test cases and found that they produced the correct values.

Issue Resolved: Y

Bug 2: Player never reaches betting limit.

Code Run Through with Debugger 1:

|  |  |  |
| --- | --- | --- |
| File | Line | Reason |
| Player.java | 27 | return (balance > limit); The balance should be allowed to be >= limit? This would make sense with the reported bug. |
| Player.java | 31 | return (balance - amount > limit); As above. E.g. amount = 5, balance = 5, limit = 0  Debugging: Set variable balance to 10:   This is before the method balanceExceedsLimitBy(int amount) returns an infected value (i.e. is sane).  This is where the the method returns an infected value: |

Hypothesis 1: Changing line 27 and line 31 in player to >= would allow the limit to be reached.

Test 1:

Changed lines as required in hypothesis and then ran through with debugger until balance = bet and ran through the test to ensure that the bet would be taken. The function no longer returned an infected value.

Ran the automated test case and collected results. Saved as Output runWinbettingLimitTest AFTER FIX.txt.

Issue Resolved: Y

Bug 3: Application odds incorrect.

Code Run Through with Debugger 1:

|  |  |  |
| --- | --- | --- |
| File | Line | Reason |
| Dice.java | 16 | The roll() method doesn’t update DiceValue value variable in class Dice. This means a call to value doesn’t return the last value of the dice when it was rolled. |
| Game.java | 21-27 | The code at this location shows that the game.getDiceValues() method simply calls dice.getValue() on each dice to return their present values. |
| Game.java | 37-41 | The number of matches the user makes on the die is done based on the dice.getValue() method which is not updated by a call to roll, which is done in line 38.  Seen with debugger. The value of d.getValue() doesn’t appear to change after a d.roll().  e.g. Before Roll:  After Roll:    The value doesn’t change.  Variable “value” in dice.java is sane before roll() and is infected after roll(). |

Hypothesis 1: The method dice.roll() should update the variable value to ensure that the dice values change.

Test 1: Set d.roll() method to update the variable:  
  
Added to line 16 of Dice.java:

value = DiceValue.getRandom();

return value;

Ran debugger and found that the variable was no longer infected.

Ran the automated test case and collected results. Saved as Output runOddsCheck AFTER FIX.txt.

Issue Resolved: Y

Bug4: Application never produces a SPADE.

Code Run Through with Debugger 1:

|  |  |  |
| --- | --- | --- |
| File | Line | Reason |
| DiceValue | 27 | int random = RANDOM.nextInt(DiceValue.SPADE.ordinal()); The “nextInt” method “Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.” Which means spade can never be drawn.  This means each call to getRandom() will return an infected value. This was only detected by looking at the output of many games and noticing that the value was missing. See SpadeNeverDrawn.txt in git repository for output of buggy behaivour.  It was observed over several runs that the random generator never returned a 6 when it was expected to return it 1 in 6 times.            Here is an observation showing the infection:  The ordinal value returned is 5, but the random int returned is 0, so the only values that can be returned by this method by definition are 0,1,2,3,4 inclusive. |

Hypothesis 1: We need to be inserting into the nextInt() method the size of the hashmap, rather than the numeric value of the last value of the map. We should change line 27 to

Test 1:

Ran through the main method again and found that spades were now present in the output of the main method. Output saved as SpadeNowDrawn.txt

Issue Resolved: Y

Template:

Code Run Through with Debugger X:

|  |  |  |
| --- | --- | --- |
| File | Line | Reason |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Hypothesis X:

Test 1:

Test 2:

Test 3:

Issue Resolved: Y/N