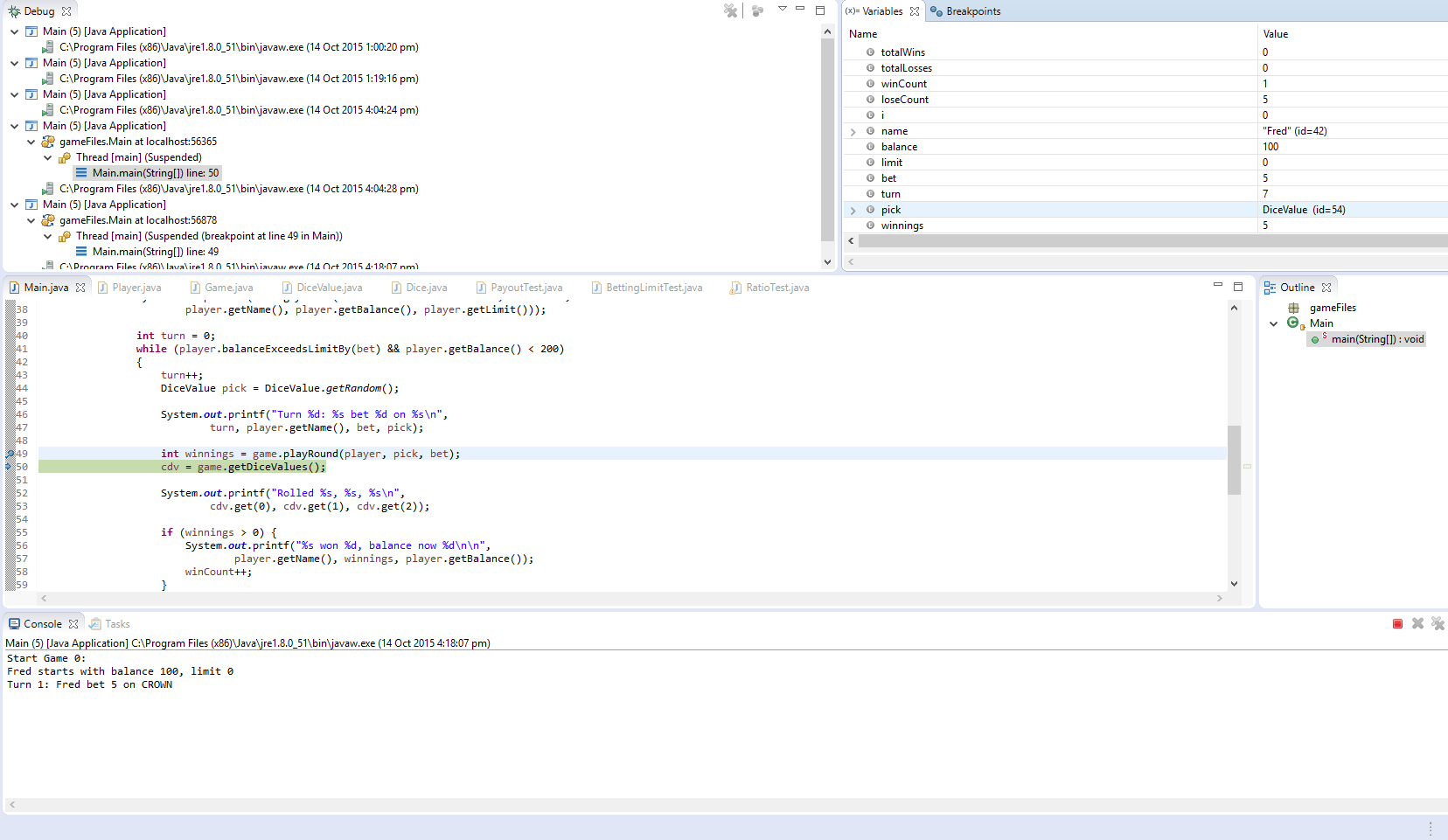
**Debugging log**

**Payout problem**

Hypothesis 1: In a winning bet with one match the value of the winnings variable should be twice the bet that was set down, as part of the game rules, therefore if the bet is 5 then this variable should be equal to 10. The problem is likely to arise when the winnings variable is set by calling game.playRound.

Set break point on the line that assigns the value of winnings. Stepping through this with a winning bet shows that this value is 5 and not 10 so the infection arises in the playRound method of the Game class.

Figure 1. debugging window showing the winnings value equal to the bet value. (infected)

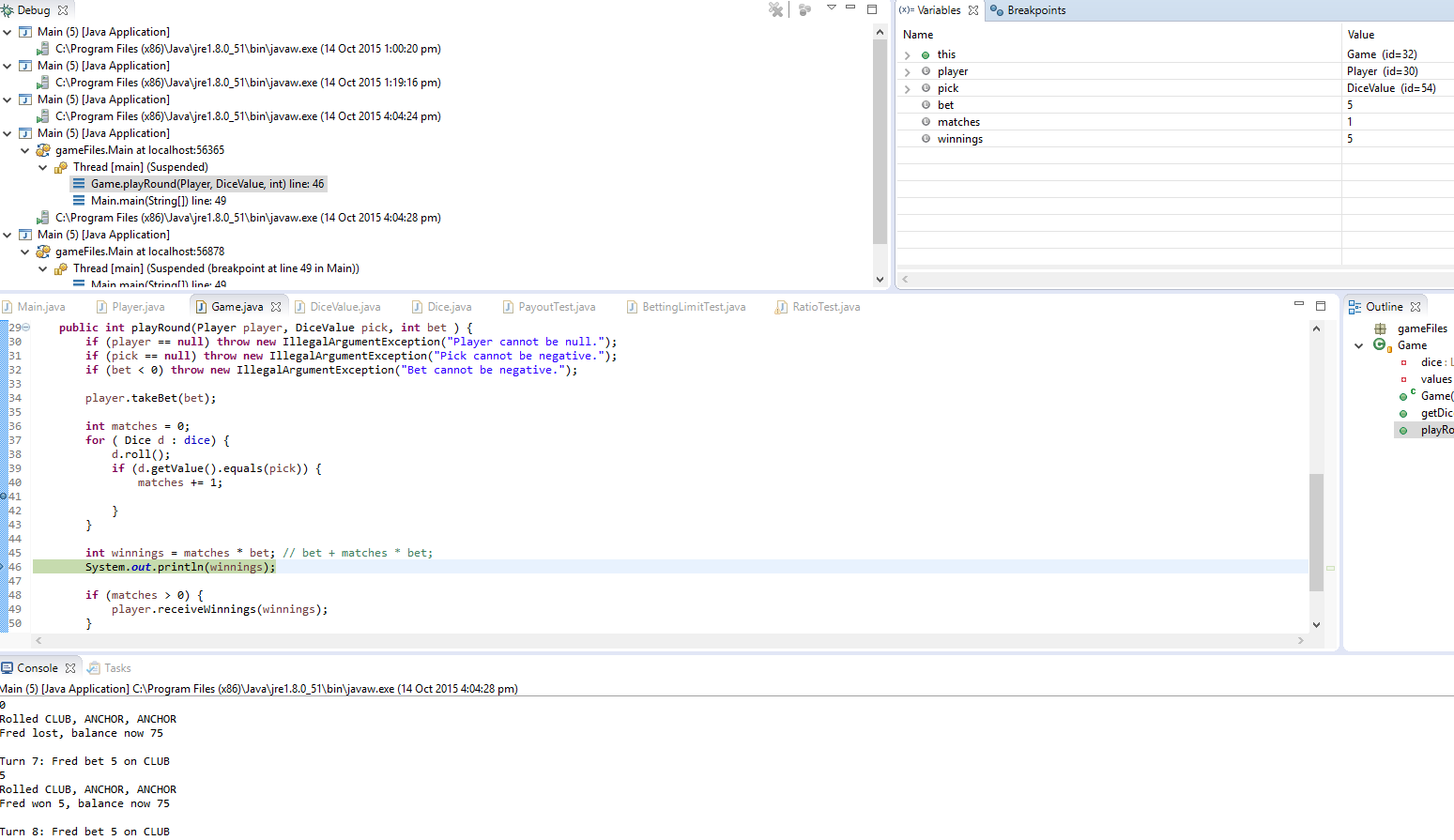


Hypothesis 2: the cause of the infection is likely to come from the playRound method.

Running a debugging on the playRound method shows that the line stating, int winnings = matches \*bet, results in the three variables: bet = 5, matches = 1, and winnings = 5.

The winnings variable is not setting the value correctly as the result should be winnings = 10 and as this is the line where the infection first occurs (calls no other methods) it is likely the cause of the infection.

Figure 2: debugging window showing the cause of the infection

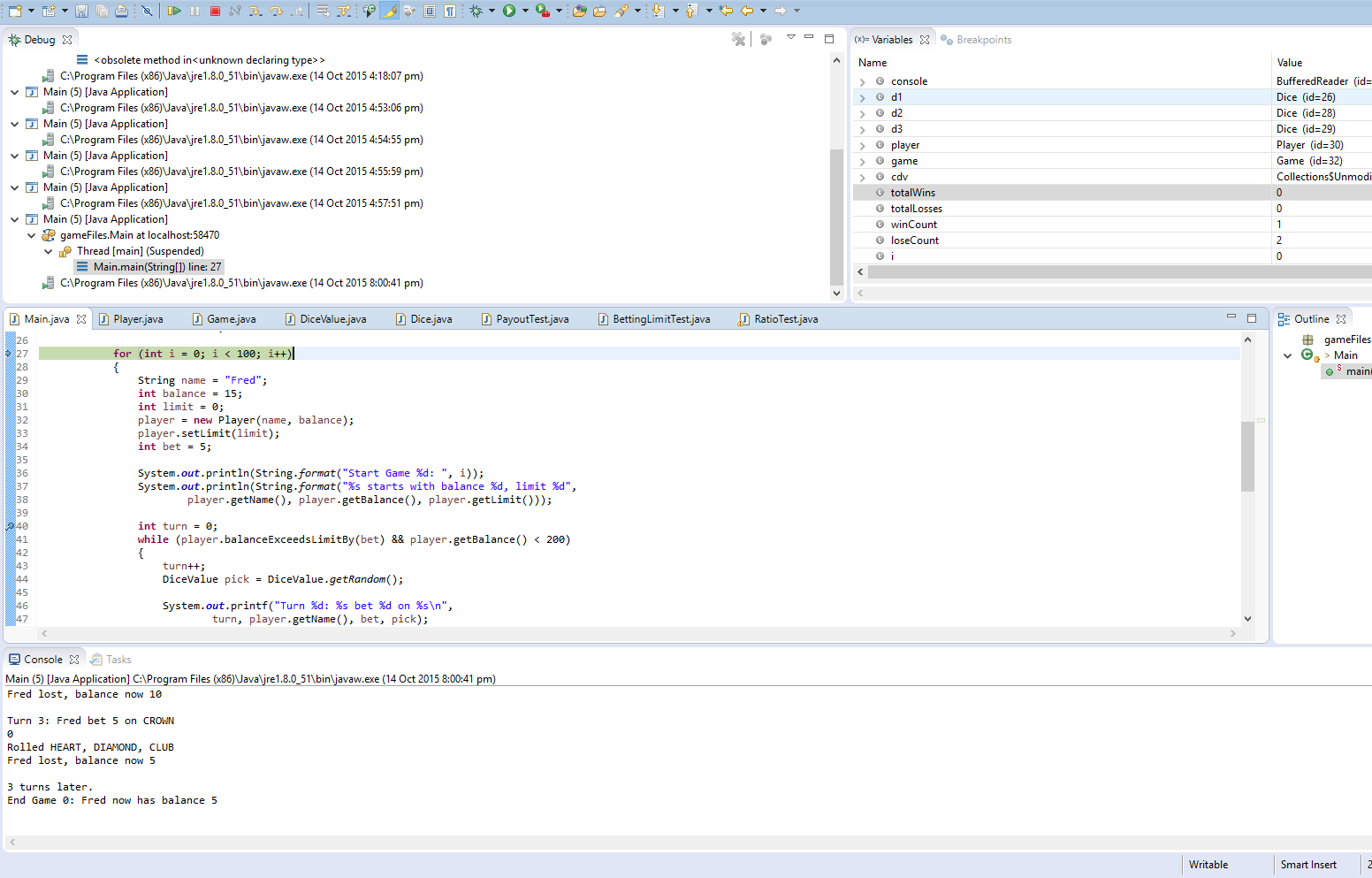


**Betting limit debugging**

Hypothesis 1: the problem likely arises in the while statement which fails to execute the last time when the balance is only one bet more than the limit.

Putting a break point before this line and stepping the program until balance = limit + bet results in the program exiting the while statement instead of running it one last time.

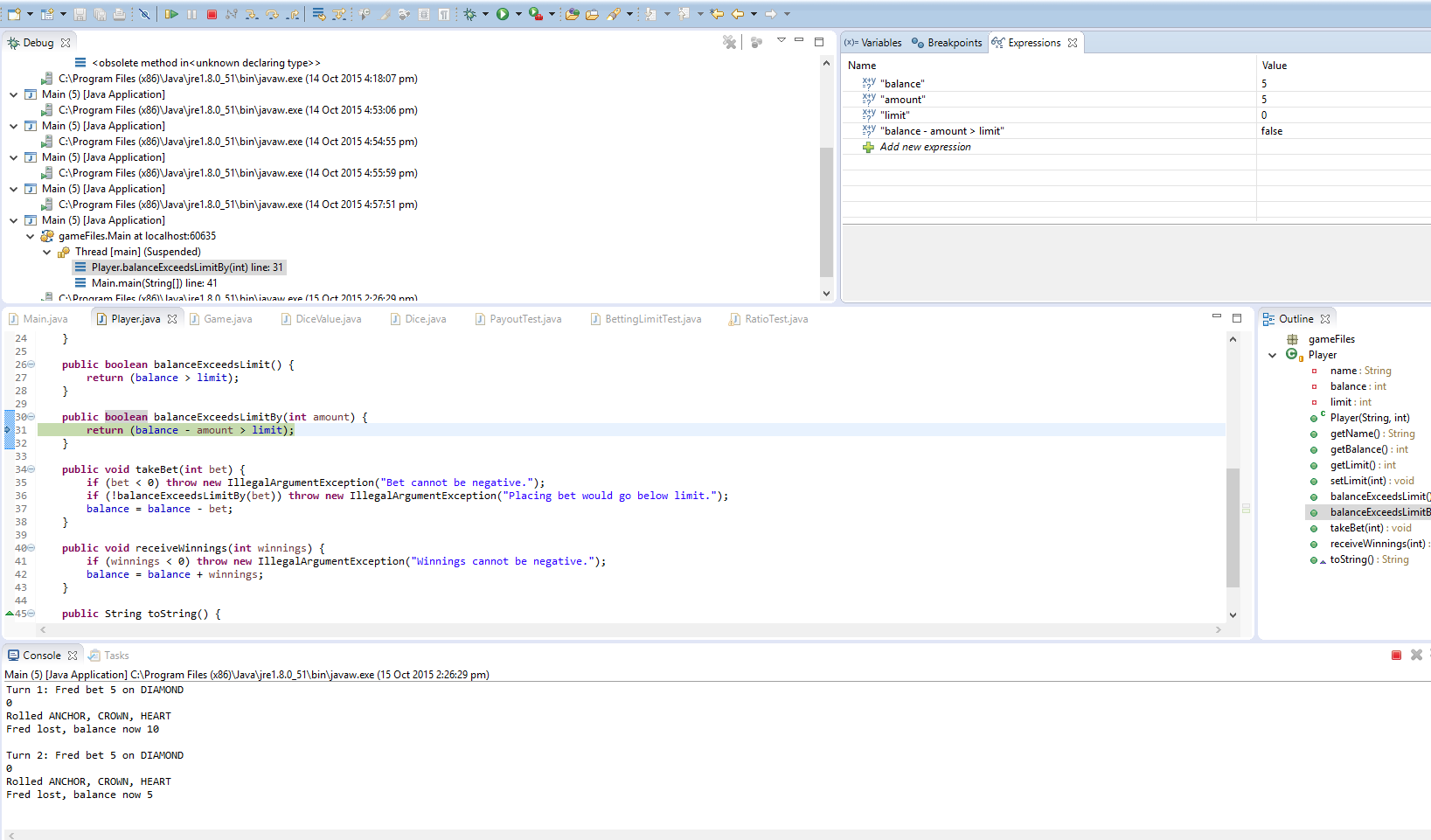
Figure 3: output showing the program leaving the while statement early



Hypothesis 2: the first Boolean in the while state is returning false when it should be true, i.e player.balanceExceedsLimitBy(bet) is not performing correctly and should be tested.

When this method runs it takes the bet(amount) and runs the statement, return (balance – amount > limit). When this statement is run for the last time, the values shown are, balance = 5, amount = 5, limit = 0 and the statement returns false. Therefore, the bug originates from this statement as it need to return true at this point.

Figure 4: debug window showing the origin of the infection.



**Non Varied Roll debugging:**

Hypothesis 1: my theory is that after the initial roll the dice values are never changed again, meaning that all of the games are played with the same dice rolls. Test this by setting a break point before the initial variable declarations, and track these to find out if they change at any point.

Figure 5: variable values after initialization.

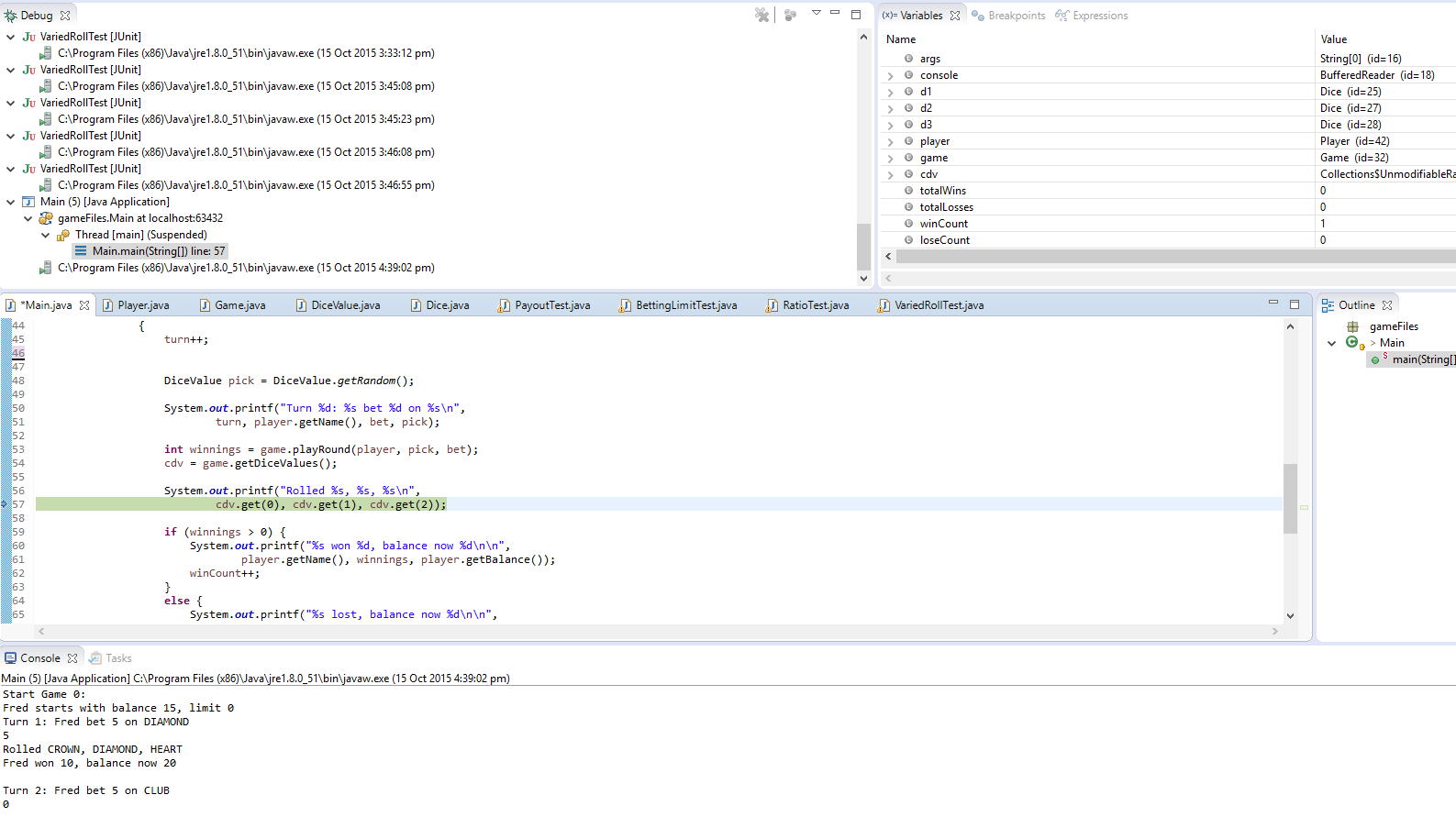
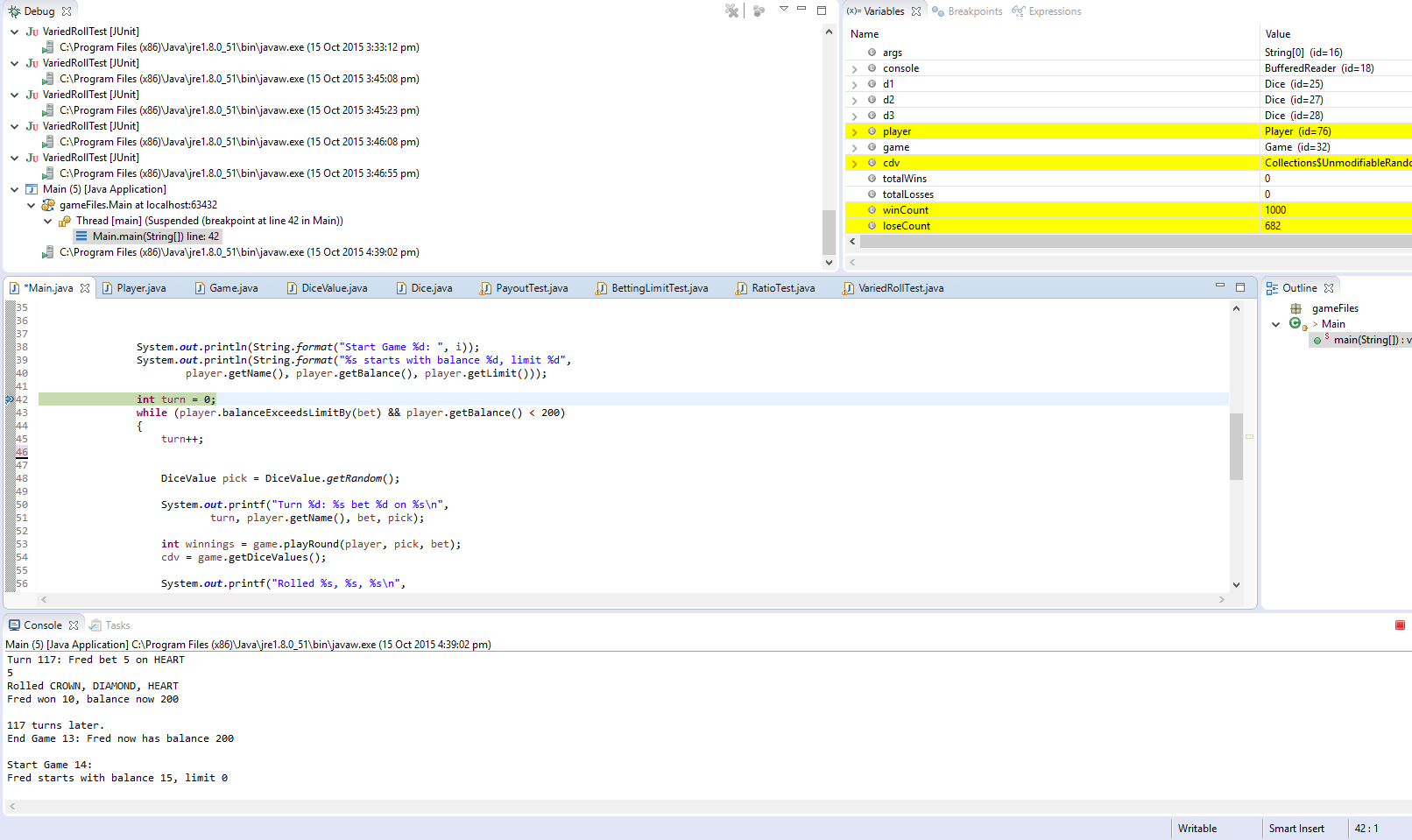


Figure 6: debugging window after a number of games have been run.



As the values for these values never change this means that the infection arises from the fact that the variables are never reassigned during the running or the program and the original hypothesis is correct.

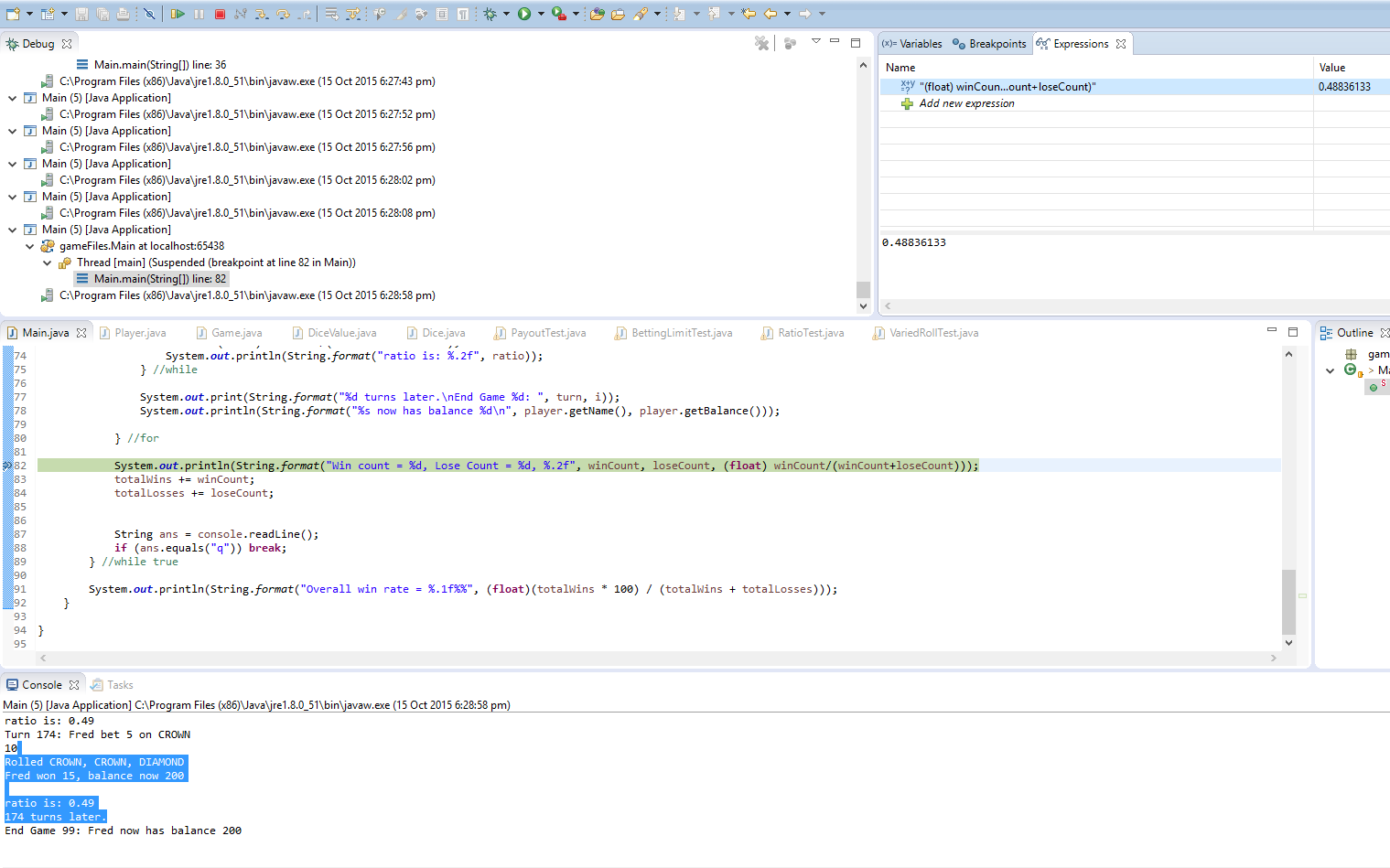
**Incorrect Ratio Debugging:**

Hypothesis 1: the first hypothesis is that when the game picks the winning rolls it does not factor in the current ratio at all and therefore there is no actual way of tracking the ratio and using it to determine the chances of winning the next game.

To test this a new variable will be introduced to track the current ratio on each roll and determine if the program brings the win rate to an average 0.42.

By running the program with the new variable to trace the ratio it can be seen that the ratio will never be 0.42. Setting a break point on each change to the ratio variable will show that the local value for this variable will rarely equal 0.42 and setting a break point after the program has run 10000+ games shows that the overall ratio will never equal 0.42.

Figure 7: debugging output showing that the ratio is not equal to 0.42.

****

Given this information the bug arises from the programs lack of tracking this ratio and using it in any calculations.