

It has taken a bit of mental confusion but it was eventually realised that any violation on the row requires different treatment.

Whether:

uniqueEntry 1 and 2, (we overwrite uniqueEntry 2)

At this point, the column checking would serve no purpose anyhow....The best technique is to maximise loop variables in which it processes the column stack.

uniqueEntry 1 and 3 (we overwrite uniqueEntry 3)

At this point, the column checking would serve no purpose anyhow....The best technique is to maximise loop variables in which it processes the column stack.

uniqueEntry 2 and 3 (we overwrite uniqueEntry 3)

At this point, the column checking would serve no purpose anyhow....The best technique is to maximise loop variables in which it processes the column stack.

uniqueEntry 1 and 2 and 3 (we overwrite UniqueEntry 2)

At this point, the column checking would serve no purpose anyhow....The best technique is to maximise loop variables in which it processes the column stack.

TEST CASE: Setting variable conditions to bypass main checking permanent selections against column violated block

```
..//I need to effectively change the highest index copyStoreRetrieved3x3GridColumn[2]=0;-  
..//We know since it has effectively removed on affected row, it should in practice not find a violation for the two grids.  
..//So we need to prevent it from doing the column check since it is repeat.-  
..//best way is to max out h and u variables in that section-  
  
..h=positionColumn;-  
..u=blockedPermutationNumberColumnSequence[0].length;-
```

TEST CASE: Ensuring a row violation performs a single track back and also skips the column checking

*****IMPORTANT INFORMATION ROW*****: Number attempts: 1

ROW SEGMENT: [10, 2, 0, 0, 0, 0, 0, 0, 0]

Violating RULE[h]: [10, 2, 0, 0, 0, 0, 0, 0, 0]

h=5 numberMatches=2 uniqueEntries=2

Grid number: 1

Location of zero (unique entry): 3 //This tells us that there are only two uniqueEntries filled and it is correct

blockedPermutationNumberSequence[h][uniqueEntries]==0: 0

2

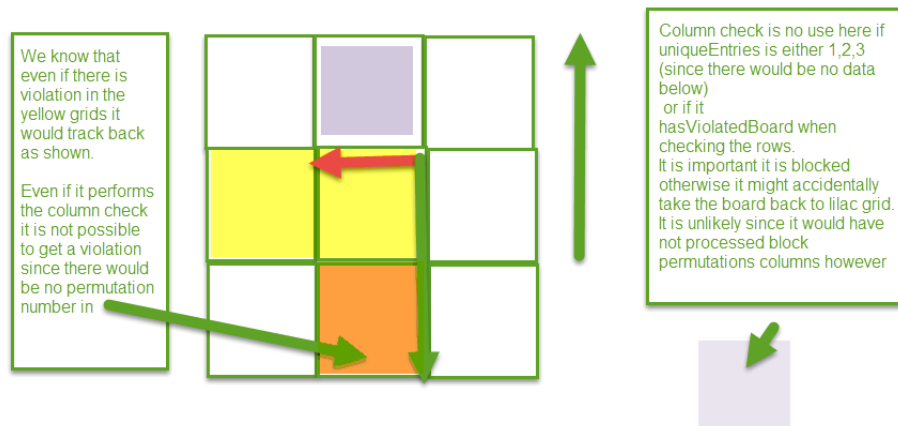
2

[10, 2, 0, 0, 0, 0, 0, 0, 0] is in full violation with [10, 2, 0, 0, 0, 0, 0, 0, 0]

This row permutation selection violates the board:

****BACKTRACKING TO: 1 //It currently backtracks to 1

*****IMPORTANT INFORMATION COLUMN*****: Number attempts: 1 //This is all excessive information, we are not concerned with checking the column segment if uniqueEntries is 1,2,3 //Furthermore if the row violates, we would be backtracking, so if the scenario was to happen further on the board....



COL SEGMENT: [2, 0, 0, 0, 0, 0, 0, 0, 0] //EXCESSIVE

Violating RULE[h]: [7, 9, 0, 0, 0, 0, 0, 0, 0] //EXCESSIVE

h=0 numberMatchesCol=0 uniqueEntries=1 //EXCESSIVE

Grid number: 1 //EXCESSIVE

Location of zero (unique entry): 1 //EXCESSIVE

blockedPermutationNumberSequence[h][uniqueEntries]==0: 7 //EXCESSIVE

*****IMPORTANT INFORMATION COLUMN*****: Number attempts: 1

COL SEGMENT: [2, 0, 0, 0, 0, 0, 0, 0, 0]

Violating RULE[h]: [9, 7, 0, 0, 0, 0, 0, 0, 0]

h=1 numberMatchesCol=0 uniqueEntries=1

Grid number: 1

Location of zero (unique entry): 1

blockedPermutationNumberSequence[h][uniqueEntries]==0: 8

*****IMPORTANT INFORMATION COLUMN*****: Number attempts: 1

COL SEGMENT: [2, 0, 0, 0, 0, 0, 0, 0, 0]

Violating RULE[h]: [1, 9, 0, 0, 0, 0, 0, 0, 0]

h=2 numberMatchesCol=0 uniqueEntries=1

Grid number: 1

Location of zero (unique entry): 1

blockedPermutationNumberSequence[h][uniqueEntries]==0: 6

*****IMPORTANT INFORMATION COLUMN*****: Number attempts: 1

COL SEGMENT: [2, 0, 0, 0, 0, 0, 0, 0, 0]

Violating RULE[h]: [9, 1, 0, 0, 0, 0, 0, 0, 0]

h=3 numberMatchesCol=0 uniqueEntries=1

Grid number: 1

Location of zero (unique entry): 1

blockedPermutationNumberSequence[h][uniqueEntries]==0: 8

REACHED END OF WHILE: 1 [10, 0, 0, 0, 0, 0, 0, 0, 0]

Starting to check violations at start first row: [10,7,0]

SEEING IF BACKTRACKING IS CORRECT (ROW VIOLATION):

It is filling unique entry: [10, 7, 0, 0, 0, 0, 0, 0, 0]

//This is perfectly fine

we had this situation initially

[10, 2, 0, 0, 0, 0, 0, 0, 0]is in full violation with [10, 2, 0, 0, 0, 0, 0, 0, 0]

And we have overwriten the 2

TEST CASE: FIXING THE NOISE ABOVE

```
if ((uniqueEntries>3 && uniqueEntries<9) && !hasViolatedBoard)
{
    beforeUniqueEntryEqualNine =(numberMatchesCol==uniqueEntries && hasMatchSeco
    && blockedPermutationNumberSequence[h][uniqueEntries]==0);

    System.out.println("*****IMPORTANT INFORMATION COLUMN*****: " +
    System.out.println("COL SEGMENT: " + Arrays.toString(copyStoreRetrieved3x3Gr
    System.out.println("Violating RULE[h]:      " + Arrays.toString(blockedPermut
    System.out.println("h=" + h + " numberMatchesCol=" + numberMatchesCol + " un
    System.out.println("Grid number: " + gridNumber);
```

OUTPUT

*****IMPORTANT INFORMATION ROW*****: Number attempts: 7

ROW SEGMENT: [3, 10, 8, 0, 0, 0, 0, 0, 0]

Violating RULE[h]: [3, 10, 8, 0, 0, 0, 0, 0, 0]

h=75 numberMatches=3 uniqueEntries=3

Grid number: 1

Location of zero (unique entry): 4

blockedPermutationNumberSequence[h][uniqueEntries] ==0: 0

[3, 10, 8, 0, 0, 0, 0, 0, 0]is in full violation with [3, 10, 8, 0, 0, 0, 0, 0, 0]

This row permutation selection violates the board:

****BACKTRACKING TO: 2

REACHED END OF WHILE: 2 [3, 10, 0, 0, 0, 0, 0, 0, 0]

REACHED END OF WHILE: 2 [3, 10, 0, 0, 0, 0, 0, 0, 0]

Starting to check violations at start first row: [3,10,5]

SEEING IF BACKTRACKING IS CORRECT (ROW VIOLATION):

It is filling unique entry: [3, 10, 5, 0, 0, 0, 0, 0, 0] //we can see that this appears incorrect

I think it is important that in the important information, I present the

System.out.println("Permutations selected: " + Arrays.toString(storeRetrieved3x3Grid));

I am in a position in my code in which I can complete lots executions on a full IDE.

It is just a case of hitting all the row scenarios described above and see how it makes sense of the backtracking...

Firstly I will also include the above observation on the outputs:

TEST CASE: Executing the code and setting system.exit(0) at point where it simply has performed backtracking and inputted another permutation number. It has not analysed replacement yet.

```
//System.out.println("*****NEW ENTRY: "
storeRetrieved3x3Grid[uniqueEntries]=randomNumber;
uniqueEntries++;

if (numFullViolationColumn>0)
{
    System.out.println("SEEING IF BACKTRACKING IS CORRECT (COLUMN VIOLATION): ");
    //System.out.println("This is current storeRetrieved3x3Grid: " + Arrays.asList(storeRetrieved
    System.out.println("It is filling unique entry: " + Arrays.toString(storeRetrieved3x3Grid));
    System.exit( status: 0);
}

if (numFullViolation>0)
{
    System.out.println("SEEING IF BACKTRACKING IS CORRECT (ROW VIOLATION):");
    //System.out.println("This is current storeRetrieved3x3Grid: " + Arrays.asList(storeRetrieved
    System.out.println("It is filling unique entry: " + Arrays.toString(storeRetrieved3x3Grid));
    System.exit( status: 0);
}
```

I configured this code once the code had looped any added another entry in

*****|IMPORTANT INFORMATION ROW*****: Number attempts: 3

Permutations selected: [4, 1, 0, 0, 0, 0, 0, 0, 0] //This is now clear information on state of play

ROW SEGMENT: [4, 1, 0, 0, 0, 0, 0, 0, 0] //this is information stored in
copyStoreRetrieved3x3Grid

Violating RULE[h]: [4, 1, 0, 0, 0, 0, 0, 0, 0]

h=39 numberMatches=2 uniqueEntries=2

Grid number: 1

Location of zero (unique entry): 3

blockedPermutationNumberSequence[h][uniqueEntries] ==0: 0

2

2

[4, 1, 0, 0, 0, 0, 0, 0, 0] is in full violation with [4, 1, 0, 0, 0, 0, 0, 0, 0]

This row permutation selection violates the board:

****BACKTRACKING TO: 1

REACHED END OF WHILE: 1 [4, 0, 0, 0, 0, 0, 0, 0, 0]

Starting to check violations at start first row: [4,6,0]

SEEING IF BACKTRACKING IS CORRECT (ROW VIOLATION):

It is filling unique entry: [4, 6, 0, 0, 0, 0, 0, 0, 0] //this is all fine for two uniqueEntries

TEST CASE: Running execution again

*****IMPORTANT INFORMATION ROW*****: Number attempts: 1

Permutations selected: [9, 8, 5, 3, 7, 0, 0, 0, 0] //now we can see the benefit of the full representation here since it would otherwise show [3,7,0.....] and it would be very confusing

ROW SEGMENT: [3, 7, 0, 0, 0, 0, 0, 0, 0]

Violating RULE[h]: [3, 7, 0, 0, 0, 0, 0, 0, 0]

h=0 numberMatches=5 uniqueEntries=5

Grid number: 1

Location of zero (unique entry): 6

blockedPermutationNumberSequence[h][uniqueEntries] ==0: 0

5

5

[9, 8, 5, 3, 7, 0, 0, 0, 0] is in full violation with [3, 7, 0, 0, 0, 0, 0, 0, 0]

This row permutation selection violates the board:

****BACKTRACKING TO: 4

REACHED END OF WHILE: 4 [9, 8, 5, 3, 0, 0, 0, 0, 0]

Starting to check violations at start second row: [3,4,0]

SEEING IF BACKTRACKING IS CORRECT (ROW VIOLATION):

It is filling unique entry: [9, 8, 5, 3, 4, 0, 0, 0, 0] //this is fine, it is similar to last test case but incident happening on second row

I have now kept executing the code until I get violation involving three unique entries..
We know I would have blocked this on basis that occurrenceNumber=3
But it now raises a very interesting question.....
Why did it not block against Violating RULE[h]:5,9
since it would have registered this as a blocked violation.
And also, is there a real purpose for three occurrences since it would have registered
the first two blocks For my previous test cases above, we can see that it recorded two
blocks violating in a row

*****IMPORTANT INFORMATION ROW*****: Number attempts: 2

Permutations selected: [5, 9, 10, 0, 0, 0, 0, 0, 0]

ROW SEGMENT: [5, 9, 10, 0, 0, 0, 0, 0, 0]

Violating RULE[h]: [5, 9, 10, 0, 0, 0, 0, 0, 0]

h=21 numberMatches=3 uniqueEntries=3

Grid number: 1

Location of zero (unique entry): 4

blockedPermutationNumberSequence[h][uniqueEntries] ==0: 0

3

3

[5, 9, 10, 0, 0, 0, 0, 0, 0]is in full violation with [5, 9, 10, 0, 0, 0, 0, 0, 0]

This row permutation selection violates the board:

****BACKTRACKING TO: 2

REACHED END OF WHILE: 2 [5, 9, 0, 0, 0, 0, 0, 0, 0]

Starting to check violations at start first row: [5,9,8]

SEEING IF BACKTRACKING IS CORRECT (ROW VIOLATION):

It is filling unique entry: [5, 9, 8, 0, 0, 0, 0, 0, 0] //and in this circumstance, it requires an additional
backtrack. I can perhaps set something in the rules section that if uniqueEntries%3==0
then perform uniqueEntries - 2

I implemented the following:

```
if (uniqueEntries%3==0)
{
    uniqueEntries=uniqueEntries-2;
}
else
{
    uniqueEntries = uniqueEntries - 1;
}
```

I think its most important that I produce screen output for current rows violated that are in the array. It is only way I can be sure of activity.

-----CURRENT violating permutation row sequences: [3, 1, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [1, 3, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [1, 4, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [4, 1, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [8, 7, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [7, 8, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [8, 7, 6, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [8, 6, 7, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [7, 8, 6, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [7, 6, 8, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [6, 7, 8, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [6, 8, 7, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [8, 6, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [6, 8, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [7, 6, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [6, 7, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [10, 5, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [5, 10, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [10, 5, 9, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [10, 9, 5, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [5, 10, 9, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [5, 9, 10, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [9, 5, 10, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [9, 10, 5, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [5, 9, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [9, 5, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [10, 9, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [9, 10, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [10, 7, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [7, 10, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [10, 3, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [3, 10, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [7, 3, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [3, 7, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [8, 1, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [1, 8, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [1, 6, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [6, 1, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [8, 6, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [6, 8, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [8, 1, 6, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [8, 6, 1, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [1, 8, 6, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [1, 6, 8, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [6, 1, 8, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [6, 8, 1, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [1, 5, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [5, 1, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [10, 5, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [5, 10, 0, 0, 0, 0, 0, 0, 0]

And it does make this comparison but it does not find it as a full violation...

*****IMPORTANT INFORMATION ROW*****: Number attempts: 2

Permutations selected: [5, 9, 0, 0, 0, 0, 0, 0, 0]

ROW SEGMENT: [5, 9, 0, 0, 0, 0, 0, 0, 0]

Violating RULE[h]: [5, 9, 0, 0, 0, 0, 0, 0, 0]

h=24 numberMatches=**1** uniqueEntries=**2**

Grid number: 1

Location of zero (unique entry): 1

blockedPermutationNumberSequence[h][uniqueEntries] ==0: 0

We know to enter the section of code:

1 2

beforeUniqueEntryEqualNine =(numberMatches==uniqueEntries &&

hasMatchSecondPermutationNumber

&& blockedPermutationNumberSequence[h][uniqueEntries]==0);

*****IMPORTANT INFORMATION ROW*****: Number attempts: 2

Permutations selected: [5, 9, 10, 0, 0, 0, 0, 0, 0]

ROW SEGMENT: [5, 9, 10, 0, 0, 0, 0, 0, 0]

Violating RULE[h]: [5, 9, 10, 0, 0, 0, 0, 0, 0]

h=21 numberMatches=3 uniqueEntries=3

Grid number: 1

Location of zero (unique entry): 4

blockedPermutationNumberSequence[h][uniqueEntries] ==0: 0

3

3

[5, 9, 10, 0, 0, 0, 0, 0, 0]is in full violation with [5, 9, 10, 0, 0, 0, 0, 0, 0]

I have a massive gut feeling it is related to swapping the order of the following code, I performed this operation

```
else
{
    //System.out.println("*****NEW ENTRY: " + randomNu
    storeRetrieved3x3Grid[uniqueEntries]=randomNumber;
    uniqueEntries++;
}
```

before:

```
...//if (uniqueEntries==1 || uniqueEntries==4 || uniqueEntries==7)-
...if (uniqueEntries==4 || uniqueEntries==7)-
...{
...    copyStoreRetrieved3x3GridColumn=new int[9];-
...    copyStoreRetrieved3x3GridColumn[0]=storeRetrieved3x3Grid[0];-
...    copyStoreRetrieved3x3GridColumn[1]=storeRetrieved3x3Grid[3];-
...    copyStoreRetrieved3x3GridColumn[2]=storeRetrieved3x3Grid[6];-
...}-
...//if (uniqueEntries==2 || uniqueEntries==5 || uniqueEntries==8)-
...if (uniqueEntries==5 || uniqueEntries==8)-
...{
...    copyStoreRetrieved3x3GridColumn=new int[9];-
...    copyStoreRetrieved3x3GridColumn[0]=storeRetrieved3x3Grid[1];-
```

I have switched it around. This will impact all my testing above. So I will try again to execute the code.

And also we realised that having three permutation numbers in a block is actually not feasible. If it hits this, then it is likely something else is wrong in the checking...

So this also means the current technique to backtrack ($\text{uniqueEntries} - 1$ is sufficient). The guard that I kept above is something that should not occur in practice.

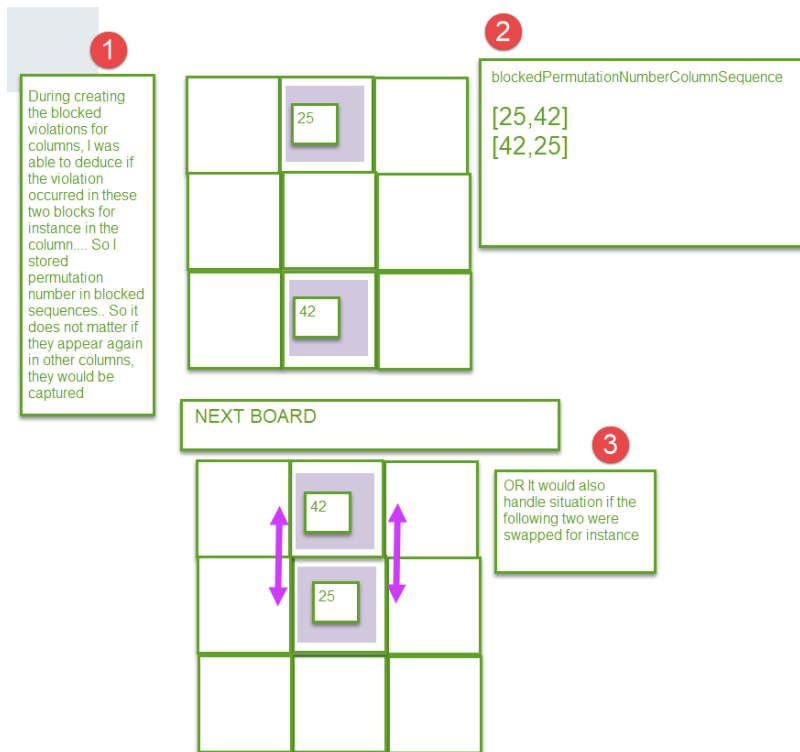
```
if (uniqueEntries%3==0)
{
    uniqueEntries=uniqueEntries-2;
}
else
{
    uniqueEntries = uniqueEntries - 1;
}
```

I am hoping also that it would be a similar case with occurrence =3 for column stacks...

I am now going to let my code execute and see the current experience when it hits a column block.

Perhaps I have over thought the situation, but it clearly suggests a single backtrack is required. I was under impression that it might require several movements. The confusion comes from the fact of when I generated the actual boards... I was actually unsure when I placed the third grid in column or third grid in row as to whether the occurrence happened between which three grids.. I did the proper analysis.

Perhaps the over analysis was storing blocked sequence for three in a row, since it would have already detected violation in first two row or column block respectively...



I am now letting my code run and hit as many column violations as possible and examining the code.

We know if column violates before row, we do not need to set any extra conditions since it will reach top of do while when uniqueEntries < 9 and then perform row operations again.

TEST CASE: Column violation scenario

*****IMPORTANT INFORMATION COLUMN*****: Number attempts: 4

Permutations selected: [6, 10, 8, 3, 9, 0, 0, 0, 0]

COL SEGMENT: [10, 9, 0, 0, 0, 0, 0, 0, 0]

Violating RULE[h]: [10, 9, 0, 0, 0, 0, 0, 0, 0]

h=0 numberMatchesCol=5 uniqueEntries=5

Grid number: 1

Location of zero (unique entry): 6

blockedPermutationNumberSequence[h][uniqueEntries]==0: 0

[6, 10, 8, 3, 9, 0, 0, 0, 0] is in full violation with [10, 9, 0, 0, 0, 0, 0, 0, 0] //This is as expected if we perform a column stack. It tells us that 10 and 9 are in question

This column permutation selection violates the board:

****BACKTRACKING TO: 4 //we can see it can be confusing what this relates to. We know it refers to 4(zero indexing), but I have given it a more meaningful output in my code now.

```
System.out.println("****BACKTRACKING TO UNIQUE ENTRY: " + (uniqueEntries+1));
```

REACHED END OF WHILE: 4 [6, 10, 8, 3, 9, 0, 0, 0, 0]

Starting to check violations at start second row: [3,7,0]

SEEING IF BACKTRACKING IS CORRECT (COLUMN VIOLATION):

It is filling unique entry: [6, 10, 8, 3, 7, 0, 0, 0, 0] //and we can see that is has successfully replaced the 9 with a 7

-----CURRENT violating permutation column sequences: [50003, 30196, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [30196, 50003, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [165051, 30196, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [30196, 165051, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [22015, 289630, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [289630, 22015, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [141230, 19255, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [19255, 141230, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [53346, 357555, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [357555, 53346, 0, 0, 0, 0, 0, 0, 0]

I let my code execute for eight hours and these were my observations.

```
*****Current completed sudoku board(s): 0 out of 108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 Attempts: 17468
NUMBER RECORDED PERMUTATION SEQUENCE ROW VIOLATIONS (includes duplicate entries): 356938
NUMBER ROW BLOCKED SEQUENCES IN EXECUTION: 0
NUMBER COL BLOCKED SEQUENCES IN EXECUTION: 0
SUCCESSFUL INPUTTED 3x3 GRIDS ONTO BOARD WITHOUT VIOLATION: 1
MAXIMUM INPUTTED SUCCESSFUL 3x3 GRIDS WITHOUT VIOLATION: 2
Better luck next time, failed on board attempt:0 Permutations selected: ([361026, 161318, 330730, 41188, 113681, 138521, 158035, 189331, 360233])minimum: 4
maximum:362876
Moving onto Board Number: 1
```

With over 17,000 boards, it is showing how tough it is for same violations to appear at row level

-----CURRENT violating permutation column sequences: [50003, 30196, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [30196, 50003, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [165051, 30196, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [30196, 165051, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [22015, 289630, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [289630, 22015, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [141230, 19255, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [19255, 141230, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [53346, 357555, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [357555, 53346, 0, 0, 0, 0, 0, 0, 0]

I was extremely surprised that there were so few readings at column level. only 5 occurrences since each entry has also been reversed.

```
int [][]blockedPermutationNumberColumnSequence = new int[9000000][9];
int [][]blockedPermutationNumberSequence = new int[9000000][9];
```

I went as high as possible without breaking the heap space. I have to question if I might have progressed through more than 17k boards in 8 hours if I based my store and search algorithm on Collections.

I knew straight away that it seemed unlikely column occurrences were that limited.

I also had a conversation with ChatGPT to understand this, and I realised in my own logic where I overcomplicated this area of code...

TEST CASE: Getting understanding from ChatGPT and attempting to resolve if blocks

I found ChatGPT gave insight into why it occurred, but I personally feel that I was able to resolve this once I fully annotated what I was setting out to achieve. For a fundamental concept, it can become extremely blurry:

```
.....//This is misleading part, when it returns here for the column check now on the same gridNumber-
.....//4 for instance against gridNumber1, since idx has increased, it will evaluate all conditions-
.....//and enter for loop-
.....
.....//I need to ensure that if idx>0, it performs a check of idx-1-
.....//otherwise we are not concerned with idx since we know its the first occurrence it is entering here-
.....//for gridNumber4 and gridNumber 1-
.....//we need to factor in the offset for this scenario...-
.....//we would simply need to get rid of the boolean storage-
.....//We know that colIndex is being incremented 0,3,6-
.....//After each match, this increment occurs, so that it prevents same two blocks until it-
.....//reaches the next ColIndex, 3 and then 6.-
.....//We effectively need that when it starts a new row, m is reset back to 0.-
.....//We would need to do this at rowIndex =6 ONLY or rowIndex = 0-
.....//since this if block is only relevant to rowIndex 3-
.....
.....//I have completely confused column concept with offset. It needed m=m+1-
.....//boolean hasIndexGreaterZero = idx>0;-
.....
..... if (occurrenceNumberCol==2 && (rowIndex>3 && rowIndex<6) && colIndex==m /*&& !hasViolationTwoStandingTopBlocks[idx]*/)-
..... //if (hasIndexGreaterZero?(occurrenceNumberCol==2 && rowIndex==3 && !hasViolationTwoStandingTopBlocks[idx-1]):-
..... //((occurrenceNumberCol==2 && rowIndex==3))-
..... {-
.....     m=m+1;
```

Still lots mental confusion how to genuinely proceed

```
if (occurrenceNumberCol==2 && rowIndex==3 && !hasViolationTwoStandingTopBlocks[idx])
{
    m=m+3;-
    idx++;-
```

Old logic

```
if (occurrenceNumberCol==2 && (rowIndex>3 && rowIndex<6) && colIndex==m /*&& !hasViolationTwoStandingTopBlocks[idx]*/)-
//if (hasIndexGreaterZero?(occurrenceNumberCol==2 && rowIndex==3 && !hasViolationTwoStandingTopBlocks[idx-1]):-
//((occurrenceNumberCol==2 && rowIndex==3))-
{-
    m=m+1;
```

New logic, still incorrect thinking that every column should be included. We actually do need to increment by 3 to ensure next adjacent block

```
if (occurrenceNumberCol==2 && (rowIndex>3 && rowIndex<6) && colIndex==m /*&& !hasViolationTwoStandingTopBlocks[idx]*/)-
//if (hasIndexGreaterZero?(occurrenceNumberCol==2 && rowIndex==3 && !hasViolationTwoStandingTopBlocks[idx-1]):-
//((occurrenceNumberCol==2 && rowIndex==3))-
{-
    m=m+3;
```

New logic again, but changing m like this is not useful. It should be m=m+1

Due to misunderstanding above, I was also performing reset of m=0 at completely wrong moments! I decided to only perform it once it confirms moving onto new board or even better inside the sudokuComplete() method.

At this point, I feel that since I do not really understand the changes fully.

And also that ChatGPT had mentioned loads of failures in this method, I need to re-visit this whole method again.

TEST CASE: Remediated the conditions on if block

I also found there were other errors, but it was related to how I reset variables in which I sometimes confused:

blockedPermutationNumberColumnSequence with

blockedPermutationNumberSequence

due to repetitive code in these areas.

These are type of examples, I consider them all critical given the context but had I read through my code properly in normal circumstances, I would have identified them. But I have to be grateful that chatGPT brought these to my attention:

1) Column “full match” uses the ROW blocked array

Inside the **column** matcher you currently have:

```
java                                                                    Copy code
beforeUniqueEntryEqualNine = (numberMatchesCol==uniqueEntries && hasMatchSecondPermutationNumberCo
    && blockedPermutationNumberSequence[h][uniqueEntries]==0);
```

✓ That must be the **column** blocked array:

```
java                                                                    Copy code
    && blockedPermutationNumberColumnSequence[h][uniqueEntries] == 0
```

Otherwise you're checking the *row* rules while evaluating a *column* rule.

2) Column “afterUniqueEntryEqualNine” uses the ROW match counter

You have:

```
java                                                                    Copy code
afterUniqueEntryEqualNine = numberMatches==uniqueEntries && hasMatchSecondPermutationNumberCol;
```

✓ Should be:

```
java                                                                    Copy code
afterUniqueEntryEqualNine = numberMatchesCol == uniqueEntries && hasMatchSecondPermutationNumberCo
```

Using `numberMatches` here makes the column logic depend on whatever the row matcher last did.


you are absolutely correct about:

2) You still have wrong-array / wrong-variable references in the column matcher

3) Column "u loop escape" uses the ROW blocked length

In the column full-violation block you set:


java

 Copy code

```
u = blockedPermutationNumberSequence[0].length;
```

✓ Should be:

java

 Copy code

```
u = blockedPermutationNumberColumnSequence[0].length;
```

Otherwise your loop-breaking values are inconsistent with the column loop you're inside.

You are right, in point 3 I have, so I overwrite the correct value!

```
h=positionColumn;
```

```
u=blockedPermutationNumberColumnSequence[0].length;
```

```
                //I need to max out these variables so it  
breaks out of entire for loop structure
```

```
                p=uniqueEntries;
```


```
                h=position;
```

```
u=blockedPermutationNumberSequence[0].length;
```

4) Column full-violation print uses the ROW blocked rule

You print:

java

 Copy code

```
System.out.println(Arrays.toString(storeRetrieved3x3Grid) +  
    "is in full violation with " +  
    Arrays.toString(blockedPermutationNumberColumnSequence[h]));
```

That part is fine — **but** a bit earlier in some of your older copies you had prints like:

java

 Copy code

```
Arrays.toString(blockedPermutationNumberSequence[h])
```

So do a quick search for `blockedPermutationNumberSequence` inside the *column* matcher region and make sure it's **never** referenced there.

You are right for point 4, I had
u=blockedPermutationNumberSequence[0].length;
it should have been referring to column

Yep — that one-liner will absolutely sabotage the column “escape”, because you’re setting `u` to the **row-array** length.

So inside the **column full-violation** block, this:

java

 Copy code

```
u = blockedPermutationNumberSequence[0].length;
```

should be:

java

 Copy code

```
u = blockedPermutationNumberColumnSequence[0].length;
```

TEST CASE: Revisiting CheckUniqueColumn() method again

For such a relatively simple concept, it has taken extreme amount of back and forth and contradictory code in order to get it correct. I was on right track first time round, then I decided against it and then I realised the first effort required tweaking,,,,

*****Current completed sudoku board(s): 0 out of

108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 Attempts: 1

NUMBER RECORDED PERMUTATION SEQUENCE ROW VIOLATIONS (includes duplicate entries): 16

NUMBER RECORDED PERMUTATION SEQUENCE COL VIOLATIONS (includes duplicate entries): 10

NUMBER ROW BLOCKED SEQUENCES IN EXECUTION: 0

NUMBER COL BLOCKED SEQUENCES IN EXECUTION: 0

SUCCESSFUL INPUTTED 3x3 GRIDS ONTO BOARD WITHOUT VIOLATION: 2

MAXIMUM INPUTTED SUCCESSFUL 3X3 GRIDS WITHOUT VIOLATION: 2

Better luck next time, failed on board attempt:0 Permutations selected: ([48509, 160672, 176035, 267708, 213314, 16390, 123494, 25158, 86658])minimum: 16390 maximum:267708

5 7 8 3 6 9 8 5 3 //This is result of the last board

9 4 3 1 2 8 2 1 6

6 2 1 7 4 5 7 9 4

8 4 5 6 5 4 5 3 7

2 7 1 7 9 2 2 6 1

9 6 3 3 1 8 8 9 4

7 4 2 9 1 8 9 2 3

5 3 9 6 4 5 6 1 4

8 6 1 7 2 3 7 8 5

Moving onto Board Number: 1

ColINDEX RIGHT NOW: 9

m right now: 0

THIS IS COLINDEX: 0

5 4 3 4 2 1 1 6 3

8 2 9 7 5 3 2 7 5

7 1 6 6 8 9 9 4 8

8 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

THIS IS COLINDEX: 4

5 4 3 4 2 1 1 6 3

8 2 9 7 5 3 2 7 5

7 1 6 6 8 9 9 4 8

8 7 5 2 5 0 0 0 0 //And it has correctly identified this

2 1 4 0 0 0 0 0 0

9 6 3 0 0 0 0 0 0 //We can see that it has now skipped 3 which is still same block

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

THIS IS COLINDEX: 6

5 4 3 4 2 1 1 6 3

8 2 9 7 5 3 2 7 5

7 1 6 6 8 9 9 4 8

8 7 5 2 5 6 4 3 6

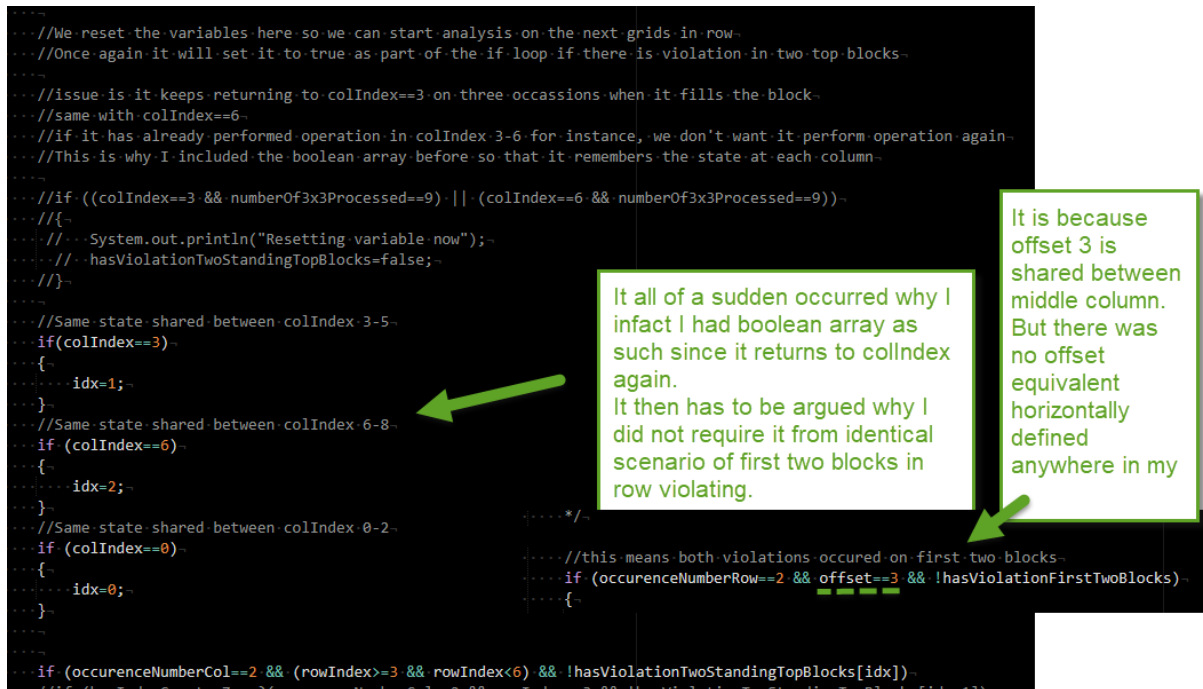
2 1 4 8 4 3 8 5 7 //we can see how it has skipped 3 which is in the same block

9 6 3 1 7 9 2 0 0 //we can see how it has skipped 9 which is in the same block

0 0 0 0 0 0 0 0 0

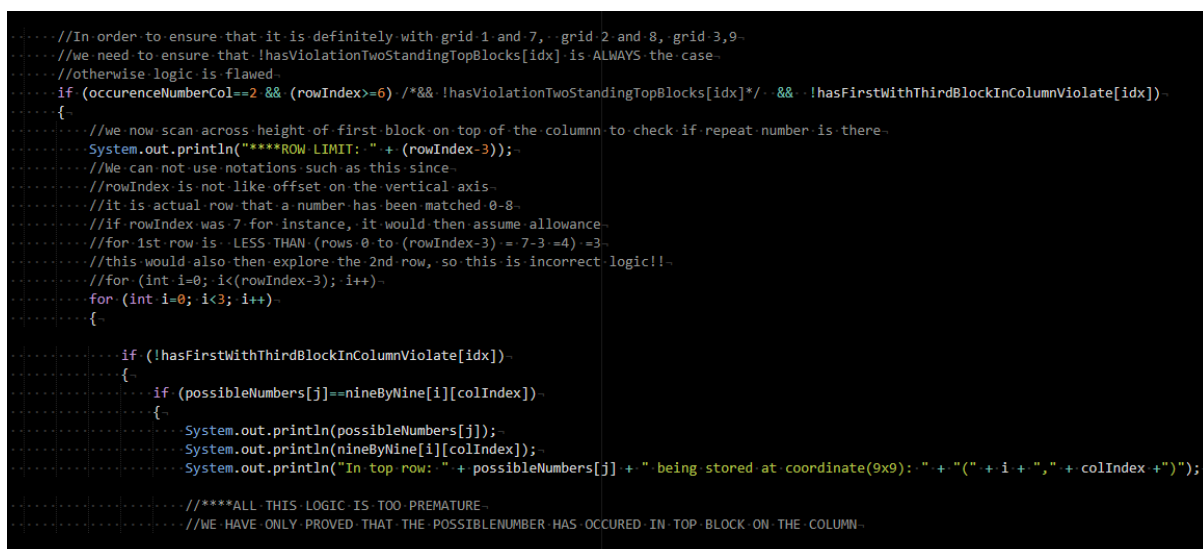
0 0 0 0 0 0 0 0 0

000000000



TEST CASE: Examining the next if statement in checkUniqueColumn block

This took lots of understanding of my code again and I could clearly understand why chatGPT was totally not satisfied. When I looked at my code again, it became much clearer on objectives..



8

In top row: 8 being stored at coordinate(9x9): (1,1) *//now its clear where we are looking*

8 being stored at coordinate(9x9): (7,1) *//we have used RowIndexMatch to ensure it can only be in bottom row of grids*

Coordinate match: 8 [1][1]

This is part of 1st row block

2Blocked permutation sequence first with third block in column reverse sequence: [205835, 210517, 0, 0, 0, 0, 0, 0] *//now we can make this comment with confidence*

3 1 6 6 4 7 4 7 8

5 8 7 9 2 8 1 5 3

9 4 2 5 1 3 2 9 6

7 9 4 5 1 6 2 1 8

8 2 5 3 2 9 4 5 7

1 6 3 7 8 4 6 3 9

6 3 1 0 0 0 0 0 0

7 8 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

I now need to move system.exit(0) statement in the else statement, this is where rowIndexMatch does not satisfy bottom row of grids..

Hopefully it will present information on first and second row grids violating .

I am then in a position where I have handled:

Top with middle

Top with bottom

Middle with bottom

OUTPUT**

Total numbers processed so far: 56 out of 81

***ROW LIMIT: 3

4

4

In top row: 4 being stored at coordinate(9x9): (1,1)

2Blocked permutation sequence second with third block in column reverse: [137315, 169143, 0, 0, 0, 0, 0, 0, 0]

256641382

143975496

897328571

948641865

657723274

321958391

590000000

000000000

000000000

It can be seen I was getting wrong result. So I kept on engineering this scenario. I eventually found I could house all my violations under one umbrella.

TEST CASE: Re-creating both methods

I have now created checkUniqueColumns method and adapted similar logic into checkUniqueRows

CheckUniqueRows()

Both scenarios run inside a single for loop and the natural area is that once it hits the criteria, it can just break out of the for loop. Since we know events are mutually exclusive.. If there are two occurrences in grid 1 and grid 2 in row (offset is at 3), it can not be (grid 1 and grid 3) or (grid 2 and grid 3).

If it finds a match in grid 3 (colIndexMatch==6) and variable i remains less than offset 3, then grid 1 and 3 violate.

If variable i is 3,4,5 and colIndexMatch==offset(6) then occurrence is in last two blocks of row

CheckUniqueColumns()

This turned out to be trickier since we had to rely on idx and also since there is no such thing as offset in horizontal direction. However the principles were just same as above method, since I transposed this method into the one above.

And now unfortunately, I need to check if it is performing correct storage and checks. I have maximised screen outputs..

TEST CASE: Examining outcomes

CheckUniqueColumns()

Note: Permutation.java uses unchecked or unsafe operations.

Note: Recompile with -Xlint:unchecked for details.

Welcome to Online IDE!! Happy Coding :)

PERMUTATIONS

$$P(n,r) = n! / (n-r)!$$

$$P(9,9) = 9! / (9-9)!$$

There are : 362880 permutations of arranging 3 x 3 grid

There are : 108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 permutations of arranging 3 x 3 grid into 9 x 9: P(362880,9)

There are : 6,670,903,752,021,072,936,960 permutations of completing sudoku (taken from internet)

This code will attempt to explore but its impossible to expect much

It is used for foundation of experimentation but also it has made serious attempt to complete random process to make a grid

I am removing excess code so it is ready future development.

Number: 1 has occurred: 0 times in column 0 grid number: 1 //AS EXPECTED, NOT REACHED GRID NUMBER 4

Number: 2 has occurred: 0 times in column 0 grid number: 1

Number: 9 has occurred: 1 times in column 8 grid number: 3 //AS EXPECTED, NOT REACHED GRID NUMBER 4

Number: 8 has occurred: 2 times in column 0 grid number: 4

CURRENT GRID NUMBER: 4

5Coordinate match: 8 [0][0]with [3][0] //This is fine

8 6 9 5 9 6 2 7 3

7 1 2 1 8 7 8 5 6

3 5 4 3 4 2 1 4 9

8 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 6 //This matches with below sequence

Storing this into blocked violation sequence: 3 //This matches with below sequence

//I would officially need to check at end of complete board if they do actually correspond to Grid number 4 and 1

1Blocked permutation sequence two standing top blocks at offset: 0 row: 3 [6, 3, 0, 0, 0, 0, 0, 0, 0]

1

3Blocked permutation sequence two standing top blocks in reverse: [3, 6, 0, 0, 0, 0, 0, 0, 0]

STATE CHECK: 0 false

STATE CHECK: 3 false

Number: 9 has occurred: 0 times in column 0 grid number: 4

Number: 8 has occurred: 2 times in column 0 grid number: 4 //We correctly see no more examinations on same grid. This is same entry as above

Number: 4 has occurred: 2 times in column 2 grid number: 4 //Again it has skipped analysis, we are no longer interested if grid 4 violates with grid 1. It will only become concern when we reach grid 7

Number: 8 has occurred: 2 times in column 0 grid number: 4

Number: 2 has occurred: 2 times in column 2 grid number: 4

Number: 5 has occurred: 2 times in column 3 grid number: 5 //we can see a new grid number 5

CURRENT GRID NUMBER: 5

5Coordinate match: 5 [0][3]with [3][3]

8 6 9 5 9 6 2 7 3

7 1 2 1 8 7 8 5 6

3 5 4 3 4 2 1 4 9

8 3 6 5 0 0 0 0 0

1 9 4 0 0 0 0 0 0

5 7 2 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 5 //Matches up with below

Storing this into blocked violation sequence: 9

1Blocked permutation sequence two standing top blocks at offset: 3 row: 3 [5, 9, 0, 0, 0, 0, 0, 0, 0]

//We can see it is the top two standing blocks. Again we would need to officially check with permutation numbers at end once it finishes the board

3

3Blocked permutation sequence two standing top blocks in reverse: [9, 5, 0, 0, 0, 0, 0, 0, 0]

STATE CHECK: 0 false

STATE CHECK: 3 false

Number: 6 has occurred: 0 times in column 3 grid number: 5

Number: 5 has occurred: 2 times in column 3 grid number: 5

Number: 5 has occurred: 2 times in column 3 grid number: 5

STATE CHECK: 0 false

STATE CHECK: 3 false

Number: 6 has occurred: 0 times in column 3 grid number: 5 //it has correctly taken no action

Number: 7 has occurred: 0 times in column 3 grid number: 5

Number: 8 has occurred: 0 times in column 3 grid number: 5

Number: 7 has occurred: 2 times in column 5 grid number: 5

Number: 5 has occurred: 2 times in column 7 grid number: 6

CURRENT GRID NUMBER: 6 //New grid number

5Coordinate match: 5 [1][7]with [3][7]

8 6 9 5 9 6 2 7 3

7 1 2 1 8 7 8 5 6

3 5 4 3 4 2 1 4 9

8 3 6 5 3 8 7 5 0

1 9 4 2 6 4 0 0 0

5 7 2 9 1 7 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 2

Storing this into blocked violation sequence: 1

1Blocked permutation sequence two standing top blocks at offset: 6 row: 3 [2, 1, 0, 0, 0, 0, 0, 0, 0] //we can see this is correct

5

3Blocked permutation sequence two standing top blocks in reverse: [1, 2, 0, 0, 0, 0, 0, 0, 0]

Number: 5 has occurred: 2 times in column 7 grid number: 6 //no actions as expected

Number: 8 has occurred: 2 times in column 6 grid number: 6

Number: 5 has occurred: 2 times in column 7 grid number: 6

Number: 5 has occurred: 2 times in column 0 grid number: 7

CURRENT GRID NUMBER: 7 //new grid

6Coordinate match: 5 [6][0] with block above //this is correct

8 6 9 5 9 6 2 7 3

7 1 2 1 8 7 8 5 6

3 5 4 3 4 2 1 4 9

8 3 6 5 3 8 7 5 1

1 9 4 2 6 4 6 9 2

5 7 2 9 1 7 8 3 4

5 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 3

Storing this into blocked violation sequence: 7

2Blocked permutation bottom sequence second with third block in column: [3, 7, 0, 0, 0, 0, 0, 0, 0]

7

2Blocked permutation sequence second with third block in column reverse: [0, 0, 0, 0, 0, 0, 0, 0, 0]

//this must be screen output upon incrementing positionColumn variable.. RESOLVED, printed wrong array

Number: 8 has occurred: 2 times in column 0 grid number: 7 //Action not required

CURRENT GRID NUMBER: 7 //we now see more action on grid number 7. We have to remember at this point, we are moving into scenario where it has grid number 1 and 4 and 7 in comparison

5Coordinate match: 8 [0][0]with [3][0]

8 6 9 5 9 6 2 7 3

7 1 2 1 8 7 8 5 6

3 5 4 3 4 2 1 4 9

8 3 6 5 3 8 7 5 1

1 9 4 2 6 4 6 9 2

5 7 2 9 1 7 8 3 4

5 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

000000000

//we have to question why it has performed this action.. Clearly it must not be linked to the Boolean in which it completed first two blocks in column violate... I will go back and look at this logic here...



Due to significant change in logic, I will need to execute code again.

TEST CASE: Re-running checkUniqueColumns

Note: Permutation.java uses unchecked or unsafe operations.

Note: Recompile with -Xlint:unchecked for details.

Welcome to Online IDE!! Happy Coding :)

PERMUTATIONS

$P(n,r) = n! / (n-r)!$

$P(9,9) = 9! / (9-9)!$

There are : 362880 permutations of arranging 3 x 3 grid

There are : 108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 permutations of arranging 3 x 3 grid into 9 x 9: $P(362880,9)$

There are : 6,670,903,752,021,072,936,960 permutations of completing sudoku (taken from internet)

This code will attempt to explore but its impossible to expect much

It is used for foundation of experimentation but also it has made serious attempt to complete random process to make a grid

I am removing excess code so it is ready future development.

Number: 1 has occurred: 0 times in column 0 grid number: 1

CURRENT GRID NUMBER: 4

5Coordinate match: 5 [0][2]with [4][2]

2 7 5 4 1 5 6 8 1

6 1 9 9 7 6 3 5 2

3 8 4 3 2 8 7 4 9

7 4 6 0 0 0 0 0 0

1 9 5 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 10

Storing this into blocked violation sequence: 8

1Blocked permutation sequence two standing top blocks at offset: 0 row: 4 [10, 8, 0, 0, 0, 0, 0, 0, 0]

1

3Blocked permutation sequence two standing top blocks in reverse: [8, 10, 0, 0, 0, 0, 0, 0, 0]

STATE CHECK: 0 false

STATE CHECK: 4 false

Number: 3 has occurred: 2 times in column 0 grid number: 4

Number: 8 has occurred: 2 times in column 1 grid number: 4

Number: 3 has occurred: 2 times in column 3 grid number: 5

CURRENT GRID NUMBER: 5

5Coordinate match: 3 [2][3]with [3][3]

2 7 5 4 1 5 6 8 1

6 1 9 9 7 6 3 5 2

3 8 4 3 2 8 7 4 9

7 4 6 3 0 0 0 0 0

1 9 5 0 0 0 0 0 0

3 8 2 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 7

Storing this into blocked violation sequence: 2

1Blocked permutation sequence two standing top blocks at offset: 3 row: 3 [7, 2, 0, 0, 0, 0, 0, 0, 0]

3

3Blocked permutation sequence two standing top blocks in reverse: [2, 7, 0, 0, 0, 0, 0, 0, 0]

Number: 3 has occurred: 2 times in column 3 grid number: 5

STATE CHECK: 2 false

STATE CHECK: 3 false

Number: 3 has occurred: 2 times in column 3 grid number: 5

STATE CHECK: 2 false

STATE CHECK: 3 false

Number: 7 has occurred: 2 times in column 4 grid number: 5

Number: 9 has occurred: 2 times in column 8 grid number: 6

CURRENT GRID NUMBER: 6

5Coordinate match: 9 [2][8]with [4][8]

2 7 5 4 1 5 6 8 1

6 1 9 9 7 6 3 5 2

3 8 4 3 2 8 7 4 9

7 4 6 3 6 2 8 3 6

1 9 5 5 9 1 1 2 9

3 8 2 8 7 4 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 6

Storing this into blocked violation sequence: 1

1Blocked permutation sequence two standing top blocks at offset: 6 row: 4 [6, 1, 0, 0, 0, 0, 0, 0, 0]

5

3Blocked permutation sequence two standing top blocks in reverse: [1, 6, 0, 0, 0, 0, 0, 0, 0]

Number: 7 has occurred: 2 times in column 6 grid number: 6

Number: 5 has occurred: 2 times in column 7 grid number: 6

Number: 9 has occurred: 2 times in column 8 grid number: 6

Number: 3 has occurred: 2 times in column 0 grid number: 7

Number: 7 has occurred: 2 times in column 0 grid number: 7

//UP TO HERE IS PERFECT, WE HAVE EXPLORED VIOLATION IN COLUMN ON ALL TOP TWO BLOCKS

CURRENT GRID NUMBER: 7

6Coordinate match: 7 [6][0] with block above

2 7 5 4 1 5 6 8 1

6 1 9 9 7 6 3 5 2

3 8 4 3 2 8 7 4 9

7 4 6 3 6 2 8 3 6

1 9 5 5 9 1 1 2 9

3 8 2 8 7 4 7 5 4

7 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 8

Storing this into blocked violation sequence: 9

2Blocked permutation bottom sequence second with third block in column: [8, 9, 0, 0, 0, 0, 0, 0, 0]

7

2Blocked permutation sequence second with third block in column reverse: [9, 8, 0, 0, 0, 0, 0, 0, 0]

Number: 8 has occurred: 2 times in column 1 grid number: 7

Number: 5 has occurred: 2 times in column 2 grid number: 7

Number: 9 has occurred: 2 times in column 2 grid number: 7

CURRENT GRID NUMBER: 7 //We are still on the same grid number

4Coordinate match: 9 [1][2]with [6][2]

2 7 5 4 1 5 6 8 1

6 1 9 9 7 6 3 5 2

3 8 4 3 2 8 7 4 9

7 4 6 3 6 2 8 3 6

1 9 5 5 9 1 1 2 9

3 8 2 8 7 4 7 5 4

7 6 9 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 10

Storing this into blocked violation sequence: 9

2Blocked permutation sequence first with third block in column: [10, 9, 0, 0, 0, 0, 0, 0, 0] //this is correct, we can see first with third...

9

2Blocked permutation sequence first with third block in column reverse sequence: [9, 10, 0, 0, 0, 0, 0, 0, 0]

STATE CHECK: 1 true

STATE CHECK: 6 true

Number: 3 has occurred: 2 times in column 0 grid number: 7

Number: 7 has occurred: 2 times in column 0 grid number: 7

Number: 4 has occurred: 2 times in column 1 grid number: 7

Number: 8 has occurred: 2 times in column 1 grid number: 7

Number: 2 has occurred: 2 times in column 2 grid number: 7

Number: 5 has occurred: 2 times in column 2 grid number: 7

Number: 9 has occurred: 2 times in column 2 grid number: 7

Number: 1 has occurred: 2 times in column 0 grid number: 7

Number: 3 has occurred: 2 times in column 0 grid number: 7

Number: 7 has occurred: 2 times in column 0 grid number: 7

Number: 4 has occurred: 2 times in column 1 grid number: 7

Number: 8 has occurred: 2 times in column 1 grid number: 7

Number: 2 has occurred: 2 times in column 2 grid number: 7

Number: 5 has occurred: 2 times in column 2 grid number: 7

Number: 9 has occurred: 2 times in column 2 grid number: 7

Number: 3 has occurred: 2 times in column 3 grid number: 8 //we can see a new grid and it has not taken into consideration this level of information anywhere. I will carry on examining forward for now. We can see this is historic information and not related to filling another 3 on the board in grid 8. It can be examined readily below

Number: 7 has occurred: 2 times in column 4 grid number: 8

CURRENT GRID NUMBER: 8

4Coordinate match: 5 [0][5]with [6][5]

2 7 5 4 1 5 6 8 1

6 1 9 9 7 6 3 5 2

3 8 4 3 2 8 7 4 9

7 4 6 3 6 2 8 3 6

1 9 5 5 9 1 1 2 9

3 8 2 8 7 4 7 5 4

7 6 9 7 8 5 0 0 0

8 4 2 0 0 0 0 0 0

1 5 3 0 0 0 0 0 0

Storing this into blocked violation sequence: 7

Storing this into blocked violation sequence: 4

2Blocked permutation sequence first with third block in column: [7, 4, 0, 0, 0, 0, 0, 0, 0] //this is correct

11

2Blocked permutation sequence first with third block in column reverse sequence: [4, 7, 0, 0, 0, 0, 0, 0, 0]

Number: 3 has occurred: 2 times in column 3 grid number: 8

Number: 4 has occurred: 2 times in column 3 grid number: 8

Number: 1 has occurred: 2 times in column 4 grid number: 8

Number: 7 has occurred: 2 times in column 4 grid number: 8

Number: 5 has occurred: 2 times in column 5 grid number: 8

Number: 3 has occurred: 3 times in column 3 grid number: 8 //this is first time we are seeing three occurrences... I have to examine my logic since it was structured along occurrenceNumber==2

This is the current board

```
2 7 5 4 1 5 6 8 1
6 1 9 9 7 6 3 5 2
3 8 4 3 2 8 7 4 9
7 4 6 3 6 2 8 3 6
1 9 5 5 9 1 1 2 9
3 8 2 8 7 4 7 5 4
7 6 9 7 8 5 9 5 1
8 4 2 4 1 9 0 0 0
1 5 3 3 2 6 0 0 0
```

We can see fortunately we have already covered second row with third row offset (0) above. But I need to add logic so that it can check if !secondwiththirdboolean[idx] where occurrence=3

I had to include this logic in the loop and also had to move the check for 2nd and third block in column into it.

```

if ((occurrenceNumberCol==3 || occurrenceNumberCol==2) && rowIndexMatch>=6 && !hasSecondWithThirdBlockInColumnViolate[idx])
{
    //If it reaches here, it means it has not broken out of the statement above-
    //it means that there is not a match in first row-
    //so it is the second and third rows with violation-
    System.out.println("STATE CHECK: " + i + " " + hasSecondWithThirdBlockInColumnViolate[idx]);
    if (i>=6 && !hasSecondWithThirdBlockInColumnViolate[idx] && !hasFirstWithThirdBlockInColumnViolate[idx])
    {
        //System.out.println("In middle row: " + possibleNumbers[j] + " being stored at coordinate(9x9): " + "(" + i + " , " + c
        //We can now be sure that both occurrences are in middle and lower row-
        //We can copy lots content into here-
    }
}

```

I will need to execute this again..

TEST CASE: Again with the changes incorporated (check unique rows)

Note: Permutation.java uses unchecked or unsafe operations.

Note: Recompile with -Xlint:unchecked for details.

Welcome to Online IDE!! Happy Coding :)

PERMUTATIONS

$P(n,r) = n! / (n - r)!$

$P(9,9) = 9! / (9-9)!$

There are : 362880 permutations of arranging 3 x 3 grid

There are : 108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 permutations of arranging 3 x 3 grid into 9 x 9: $P(362880,9)$

There are : 6,670,903,752,021,072,936,960 permutations of completing sudoku (taken from internet)

This code will attempt to explore but its impossible to expect much

It is used for foundation of experimentation but also it has made serious attempt to complete random process to make a grid

I am removing excess code so it is ready future development.

Number: 1 has occurred: 2 times in column 0 grid number: 4

CURRENT GRID NUMBER: 4

5Coordinate match: 1 [1][0]with [3][0]

6 5 9 9 4 3 9 2 7

1 4 3 5 2 7 5 8 3

8 7 2 1 6 8 6 4 1

1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 4

Storing this into blocked violation sequence: 3

1Blocked permutation sequence two standing top blocks at offset: 0 row: 3 [4, 3, 0, 0, 0, 0, 0, 0, 0]

1

3Blocked permutation sequence two standing top blocks in reverse: [3, 4, 0, 0, 0, 0, 0, 0, 0]

Number: 1 has occurred: 2 times in column 0 grid number: 4

Number: 1 has occurred: 2 times in column 0 grid number: 4

Number: 4 has occurred: 2 times in column 1 grid number: 4

Number: 5 has occurred: 2 times in column 3 grid number: 5

CURRENT GRID NUMBER: 5

5Coordinate match: 5 [1][3]with [4][3]

6 5 9 9 4 3 9 2 7

1 4 3 5 2 7 5 8 3

8 7 2 1 6 8 6 4 1

1 9 8 3 1 2 0 0 0

5 2 6 5 0 0 0 0 0

3 4 7 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 6

Storing this into blocked violation sequence: 1

1Blocked permutation sequence two standing top blocks at offset: 3 row: 4 [6, 1, 0, 0, 0, 0, 0, 0, 0]

3

3Blocked permutation sequence two standing top blocks in reverse: [1, 6, 0, 0, 0, 0, 0, 0, 0]

Number: 6 has occurred: 2 times in column 4 grid number: 5

Number: 7 has occurred: 2 times in column 5 grid number: 5

Number: 5 has occurred: 2 times in column 3 grid number: 5

Number: 4 has occurred: 2 times in column 4 grid number: 5

Number: 6 has occurred: 2 times in column 4 grid number: 5

Number: 7 has occurred: 2 times in column 5 grid number: 5

Number: 2 has occurred: 2 times in column 7 grid number: 6

CURRENT GRID NUMBER: 6

5Coordinate match: 2 [0][7]with [3][7]

6 5 9 9 4 3 9 2 7

1 4 3 5 2 7 5 8 3

8 7 2 1 6 8 6 4 1

1 9 8 3 1 2 8 2 0

5 2 6 5 6 7 0 0 0

3 4 7 8 4 9 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 8

Storing this into blocked violation sequence: 5

1Blocked permutation sequence two standing top blocks at offset: 6 row: 3 [8, 5, 0, 0, 0, 0, 0, 0, 0]

5

3Blocked permutation sequence two standing top blocks in reverse: [5, 8, 0, 0, 0, 0, 0, 0, 0]

Number: 2 has occurred: 2 times in column 7 grid number: 6

Number: 9 has occurred: 2 times in column 6 grid number: 6

Number: 2 has occurred: 2 times in column 7 grid number: 6

Number: 3 has occurred: 2 times in column 8 grid number: 6

Number: 1 has occurred: 2 times in column 0 grid number: 7

Number: 3 has occurred: 2 times in column 0 grid number: 7

CURRENT GRID NUMBER: 7

6Coordinate match: 3 [6][0] with block above

6 5 9 9 4 3 9 2 7

1 4 3 5 2 7 5 8 3

8 7 2 1 6 8 6 4 1

1 9 8 3 1 2 8 2 5

5 2 6 5 6 7 4 7 6

3 4 7 8 4 9 9 1 3

3 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 3

Storing this into blocked violation sequence: 2

2Blocked permutation bottom sequence second with third block in column: [3, 2, 0, 0, 0, 0, 0, 0, 0]

7

2Blocked permutation sequence second with third block in column reverse: [2, 3, 0, 0, 0, 0, 0, 0, 0]

Number: 4 has occurred: 2 times in column 1 grid number: 7

Number: 7 has occurred: 2 times in column 1 grid number: 7

CURRENT GRID NUMBER: 7

4Coordinate match: 7 [2][1]with [6][1]

6 5 9 9 4 3 9 2 7

1 4 3 5 2 7 5 8 3

8 7 2 1 6 8 6 4 1

1 9 8 3 1 2 8 2 5

5 2 6 5 6 7 4 7 6

3 4 7 8 4 9 9 1 3

3 7 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 4

Storing this into blocked violation sequence: 2

2Blocked permutation sequence first with third block in column: [4, 2, 0, 0, 0, 0, 0, 0, 0]

9

2Blocked permutation sequence first with third block in column reverse sequence: [2, 4, 0, 0, 0, 0, 0, 0, 0]

Number: 1 has occurred: 2 times in column 0 grid number: 7

Number: 3 has occurred: 2 times in column 0 grid number: 7

Number: 4 has occurred: 2 times in column 1 grid number: 7

Number: 5 has occurred: 2 times in column 1 grid number: 7

Number: 7 has occurred: 2 times in column 1 grid number: 7

Number: 1 has occurred: 2 times in column 0 grid number: 7

Number: 2 has occurred: 0 times in column 0 grid number: 7

Number: 3 has occurred: 2 times in column 0 grid number: 7

Number: 6 has occurred: 2 times in column 0 grid number: 7

Number: 7 has occurred: 0 times in column 0 grid number: 7

Number: 4 has occurred: 2 times in column 1 grid number: 7

Number: 5 has occurred: 2 times in column 1 grid number: 7

Number: 7 has occurred: 2 times in column 1 grid number: 7

Number: 2 has occurred: 2 times in column 2 grid number: 7

Number: 5 has occurred: 2 times in column 3 grid number: 8

Number: 4 has occurred: 2 times in column 4 grid number: 8

Number: 6 has occurred: 2 times in column 4 grid number: 8

Number: 3 has occurred: 2 times in column 5 grid number: 8

CURRENT GRID NUMBER: 8

4Coordinate match: 3 [0][5]with [6][5]

6 5 9 9 4 3 9 2 7

1 4 3 5 2 7 5 8 3

8 7 2 1 6 8 6 4 1

1 9 8 3 1 2 8 2 5

5 2 6 5 6 7 4 7 6

3 4 7 8 4 9 9 1 3

3 7 1 6 8 3 0 0 0

9 5 4 0 0 0 0 0 0

6 8 2 0 0 0 0 0 0

Storing this into blocked violation sequence: 6

Storing this into blocked violation sequence: 7

2Blocked permutation sequence first with third block in column: [6, 7, 0, 0, 0, 0, 0, 0, 0]

11

2Blocked permutation sequence first with third block in column reverse sequence: [7, 6, 0, 0, 0, 0, 0, 0, 0]

Number: 7 has occurred: 2 times in column 5 grid number: 8

Number: 5 has occurred: 2 times in column 3 grid number: 8

Number: 9 has occurred: 2 times in column 3 grid number: 8

Number: 4 has occurred: 2 times in column 4 grid number: 8

Number: 6 has occurred: 2 times in column 4 grid number: 8

Number: 3 has occurred: 2 times in column 5 grid number: 8

Number: 7 has occurred: 3 times in column 5 grid number: 8 // we can see again it has not entered into loop

This is the grid, red is showing the last write in which it performed 1st block with 3rd block in column
It would have set hasFirstWithThirdBlockInColumnViolate[idx]=true;

6 5 9 9 4 3 9 2 7

1 4 3 5 2 7 5 8 3

8 7 2 1 6 8 6 4 1

1 9 8 3 1 2 8 2 5

5 2 6 5 6 7 4 7 6

3 4 7 8 4 9 9 1 3

3 7 1 6 8 3 2 8 0

9 5 4 9 5 7 0 0 0

6 8 2 4 2 1 0 0 0

But in the blue check, we are looking at the following:

We know it is correct up to here

idx would be 2 here and it should enter since no operation done here

```
if ((occurrenceNumberCol==3 || occurrenceNumberCol==2) && rowIndexMatch>=6 && !hasSecondWithThirdBlockInColumnViolate[idx]) {  
    // If it reaches here, it means it has not broken out of the statement above
```

I am going to include `system.exit(0)` in my code when `occurene==3` and check other variables

TEST CASE: Using `System.exit(0)` as described above

Note: `Permutation.java` uses unchecked or unsafe operations.

Note: Recompile with `-Xlint:unchecked` for details.

Welcome to Online IDE!! Happy Coding :)

PERMUTATIONS

$P(n,r) = n! / (n - r)!$

$P(9,9) = 9! / (9-9)!$

There are : 362880 permutations of arranging 3 x 3 grid

There are : 108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 permutations of arranging 3 x 3 grid into 9 x 9: $P(362880,9)$

There are : 6,670,903,752,021,072,936,960 permutations of completing sudoku (taken from internet)

This code will attempt to explore but its impossible to expect much

It is used for foundation of experimentation but also it has made serious attempt to complete random process to make a grid

I am removing excess code so it is ready future development.

Number: 4 has occurred: 2 times in column 0 grid number: 4

CURRENT GRID NUMBER: 4

5Coordinate match: 4 [0][0]with [3][0]

465251254

287763398

319849671

400000000

000000000

000000000

000000000

000000000

000000000

Storing this into blocked violation sequence: 2

Storing this into blocked violation sequence: 3

1Blocked permutation sequence two standing top blocks at offset: 0 row: 3 [2, 3, 0, 0, 0, 0, 0, 0]

1

3Blocked permutation sequence two standing top blocks in reverse: [3, 2, 0, 0, 0, 0, 0, 0]

CURRENT GRID NUMBER: 5

5Coordinate match: 7 [1][3]with [3][3]

465251254

287763398

319849671

492700000

786000000

153000000

000000000

000000000

000000000

Storing this into blocked violation sequence: 5

Storing this into blocked violation sequence: 9

1Blocked permutation sequence two standing top blocks at offset: 3 row: 3 [5, 9, 0, 0, 0, 0, 0, 0]

3

3Blocked permutation sequence two standing top blocks in reverse: [9, 5, 0, 0, 0, 0, 0, 0]

CURRENT GRID NUMBER: 6

5Coordinate match: 7 [2][7]with [3][7]

4 6 5 2 5 1 2 5 4

2 8 7 7 6 3 3 9 8

3 1 9 8 4 9 6 7 1

4 9 2 7 5 4 4 7 0

7 8 6 2 9 8 0 0 0

1 5 3 1 3 6 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

CURRENT GRID NUMBER: 7

6Coordinate match: 1 [7][0] with block above

4 6 5 2 5 1 2 5 4

2 8 7 7 6 3 3 9 8

3 1 9 8 4 9 6 7 1

4 9 2 7 5 4 4 7 2

7 8 6 2 9 8 6 3 1

1 5 3 1 3 6 8 9 5

5 3 4 0 0 0 0 0 0

1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 3

Storing this into blocked violation sequence: 7

2Blocked permutation bottom sequence second with third block in column: [3, 7, 0, 0, 0, 0, 0, 0, 0]

7

2Blocked permutation sequence second with third block in column reverse: [7, 3, 0, 0, 0, 0, 0, 0, 0]

Number: 2 has occurred: 1 times in column 0 grid number: 7

Number: 3 has occurred: 1 times in column 0 grid number: 7

Number: 4 has occurred: 2 times in column 0 grid number: 7

Number: 5 has occurred: 1 times in column 0 grid number: 7

Number: 6 has occurred: 0 times in column 0 grid number: 7

Number: 7 has occurred: 1 times in column 0 grid number: 7
Number: 8 has occurred: 0 times in column 0 grid number: 7
Number: 9 has occurred: 0 times in column 0 grid number: 7
Number: 1 has occurred: 1 times in column 1 grid number: 7
Number: 2 has occurred: 0 times in column 1 grid number: 7
Number: 3 has occurred: 1 times in column 1 grid number: 7
Number: 4 has occurred: 0 times in column 1 grid number: 7
Number: 5 has occurred: 1 times in column 1 grid number: 7
Number: 6 has occurred: 1 times in column 1 grid number: 7
Number: 7 has occurred: 1 times in column 1 grid number: 7
Number: 8 has occurred: 2 times in column 1 grid number: 7
Number: 9 has occurred: 1 times in column 1 grid number: 7
Number: 1 has occurred: 0 times in column 2 grid number: 7
Number: 2 has occurred: 1 times in column 2 grid number: 7
Number: 3 has occurred: 1 times in column 2 grid number: 7
Number: 4 has occurred: 1 times in column 2 grid number: 7
Number: 5 has occurred: 1 times in column 2 grid number: 7
Number: 6 has occurred: 2 times in column 2 grid number: 7
Number: 7 has occurred: 1 times in column 2 grid number: 7
Number: 8 has occurred: 0 times in column 2 grid number: 7
Number: 9 has occurred: 1 times in column 2 grid number: 7
Number: 1 has occurred: 2 times in column 0 grid number: 7
Number: 2 has occurred: 1 times in column 0 grid number: 7
Number: 3 has occurred: 1 times in column 0 grid number: 7
Number: 4 has occurred: 2 times in column 0 grid number: 7
Number: 5 has occurred: 1 times in column 0 grid number: 7
Number: 6 has occurred: 0 times in column 0 grid number: 7
Number: 7 has occurred: 1 times in column 0 grid number: 7
Number: 8 has occurred: 1 times in column 0 grid number: 7
Number: 9 has occurred: 0 times in column 0 grid number: 7
Number: 1 has occurred: 1 times in column 1 grid number: 7

Number: 2 has occurred: 1 times in column 1 grid number: 7
Number: 3 has occurred: 1 times in column 1 grid number: 7
Number: 4 has occurred: 0 times in column 1 grid number: 7
Number: 5 has occurred: 1 times in column 1 grid number: 7
Number: 6 has occurred: 1 times in column 1 grid number: 7
Number: 7 has occurred: 1 times in column 1 grid number: 7
Number: 8 has occurred: 2 times in column 1 grid number: 7
Number: 9 has occurred: 1 times in column 1 grid number: 7
Number: 1 has occurred: 0 times in column 2 grid number: 7
Number: 2 has occurred: 1 times in column 2 grid number: 7
Number: 3 has occurred: 1 times in column 2 grid number: 7
Number: 4 has occurred: 1 times in column 2 grid number: 7
Number: 5 has occurred: 1 times in column 2 grid number: 7
Number: 6 has occurred: 2 times in column 2 grid number: 7
Number: 7 has occurred: 1 times in column 2 grid number: 7
Number: 8 has occurred: 0 times in column 2 grid number: 7
Number: 9 has occurred: 2 times in column 2 grid number: 7
CURRENT GRID NUMBER: 7

4Coordinate match: 9 [2][2]with [8][2]

4 6 5 2 5 1 2 5 4

2 8 7 7 6 3 3 9 8

3 1 9 8 4 9 6 7 1

4 9 2 7 5 4 4 7 2

7 8 6 2 9 8 6 3 1

1 5 3 1 3 6 8 9 5

5 3 4 0 0 0 0 0 0

1 7 6 0 0 0 0 0 0

8 2 9 0 0 0 0 0 0

CURRENT GRID NUMBER: 8

4Coordinate match: 1 [0][5]with [7][5]

4 6 5 2 5 1 2 5 4

287763398

319849671

492754472

786298631

153136895

534625000

176471000

829000000

CURRENT GRID NUMBER: 9

6Coordinate match: 8 [6][6] with block above

465251254

287763398

319849671

492754472

786298631

153136895

534625800

176471000

829938000

Number: 7 has occurred: 3 times in column 7 grid number: 9

6

2

True

It now reaches the correct area, so I need to execute the code normally.

Once it finds occurrence=3 and processes the block violation, I will force it to exit after then.

It would be great to see if it behaves ok, since we know it is fine for occurrence=2

```

CURRENT GRID NUMBER: 8
6Coordinate match: 9 [6][4] with block above
9 2 8 8 7 6 3 7 2
6 7 1 3 1 2 1 4 6
5 3 4 5 9 4 5 9 8
9 5 4 4 6 7 3 9 1
8 6 2 2 8 1 4 6 2
7 1 3 5 9 3 8 7 5
1 6 4 7 9 0 0 0 0
3 2 9 0 0 0 0 0 0
5 7 8 0 0 0 0 0 0
Storing this into blocked violation sequence: 7
Storing this into blocked violation sequence: 5
2Blocked permutation bottom sequence second with third block in column: [7, 5, 0, 0, 0, 0, 0, 0, 0]
11
2Blocked permutation sequence second with third block in column reverse: [5, 7, 0, 0, 0, 0, 0, 0, 0]
6
1
true

```

I am hoping 5 in the blue block is captured with the row above.... And fortunately it is, so all seem well

```

5Coordinate match: 5 [2][3]with [5][3]
9 2 8 8 7 6 3 7 2
6 7 1 3 1 2 1 4 6
5 3 4 5 9 4 5 9 8
9 5 4 4 6 7 0 0 0
8 6 2 2 8 1 0 0 0
7 1 3 5 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
Storing this into blocked violation sequence: 9
Storing this into blocked violation sequence: 7
1Blocked permutation sequence two standing top blocks at offset: 3 row: 5 [9, 7, 0, 0, 0, 0, 0, 0, 0]
3
3Blocked permutation sequence two standing top blocks in reverse: [7, 9, 0, 0, 0, 0, 0, 0, 0]

```

Everything is looking perfect...

Now, as a final guard, once it performs this if section, I know that if it reaches occurrence==3 and either of the booleans in index are false, there is a mistake somewhere.

I think it's a good guard at the column level since we expect 3 occurrences of a number to happen practically every time.

This would be outside of the main if (occurrenceNumberCol>=2) I would also increment this variable and place it in my summary. It will effectively mean that I will have three number streak in blockedPermutationNumberColumnSequence[positionColumn]

```

if (totalNumbersProcessed==81 && (j==possibleNumbers.length-1) && occurrenceNumberCol==3 && (!hasSecondWithThirdBlockInColumnViolate[idx] || !hasFirstWithThirdBlockInColumnViolate[idx]))
{
    System.out.println("ERROR");
    //We can potentially look to add the logic we did for occurrenceNumberCol=3 into here
    //All content will be added for the entire column and
    numStoreOccurenceThreeInColumn++;
}

```


TEST CASE: Running through execution again (check Unique Columns)

Note: Permutation.java uses unchecked or unsafe operations.

Note: Recompile with -Xlint:unchecked for details.

Welcome to Online IDE!! Happy Coding :)

PERMUTATIONS

$P(n,r) = n! / (n-r)!$

$P(9,9) = 9! / (9-9)!$

There are : 362880 permutations of arranging 3 x 3 grid

There are : 108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 permutations of arranging 3 x 3 grid into 9 x 9: $P(362880,9)$

There are : 6,670,903,752,021,072,936,960 permutations of completing sudoku (taken from internet)

This code will attempt to explore but its impossible to expect much

It is used for foundation of experimentation but also it has made serious attempt to complete random process to make a grid

I am removing excess code so it is ready future development.

CURRENT GRID NUMBER: 4

5Coordinate match: 2 [2][0]with [3][0]

8 3 7 2 8 9 8 7 1

1 6 4 7 1 3 6 2 3

2 9 5 6 4 5 4 9 5

2 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 8

Storing this into blocked violation sequence: 10

1Blocked permutation sequence two standing top blocks at offset: 0 row: 3 [8, 10, 0, 0, 0, 0, 0, 0, 0]

1

Permutations selected: ([8, 9, 3, 10, 4, 1, 7, 2, 5])

3Blocked permutation sequence two standing top blocks in reverse: [10, 8, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 9, 3, 10, 4, 1, 7, 2, 5])

Number: 3 has occurred: 0 times in column 0 grid number: 4

Number: 2 has occurred: 2 times in column 0 grid number: 4

Number: 1 has occurred: 2 times in column 0 grid number: 4

Number: 2 has occurred: 2 times in column 0 grid number: 4

Number: 7 has occurred: 2 times in column 3 grid number: 5

CURRENT GRID NUMBER: 5

5Coordinate match: 7 [1][3]with [4][3]

8 3 7 2 8 9 8 7 1

1 6 4 7 1 3 6 2 3

2 9 5 6 4 5 4 9 5

2 4 8 3 5 8 0 0 0

6 7 3 7 0 0 0 0 0

1 9 5 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 9

Storing this into blocked violation sequence: 4

1Blocked permutation sequence two standing top blocks at offset: 3 row: 4 [9, 4, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 9, 3, 10, 4, 1, 7, 2, 5])

3

3Blocked permutation sequence two standing top blocks in reverse: [4, 9, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 9, 3, 10, 4, 1, 7, 2, 5])

Number: 1 has occurred: 2 times in column 4 grid number: 5

Number: 7 has occurred: 2 times in column 3 grid number: 5

Number: 8 has occurred: 0 times in column 3 grid number: 5

Number: 9 has occurred: 0 times in column 3 grid number: 5

Number: 1 has occurred: 2 times in column 4 grid number: 5

Number: 2 has occurred: 0 times in column 4 grid number: 5

Number: 3 has occurred: 0 times in column 4 grid number: 5
Number: 4 has occurred: 1 times in column 4 grid number: 5
Number: 5 has occurred: 1 times in column 4 grid number: 5
Number: 6 has occurred: 0 times in column 4 grid number: 5
Number: 7 has occurred: 0 times in column 4 grid number: 5
Number: 8 has occurred: 1 times in column 4 grid number: 5
Number: 9 has occurred: 1 times in column 4 grid number: 5
Number: 1 has occurred: 0 times in column 5 grid number: 5
Number: 2 has occurred: 1 times in column 5 grid number: 5
Number: 3 has occurred: 1 times in column 5 grid number: 5
Number: 4 has occurred: 0 times in column 5 grid number: 5
Number: 5 has occurred: 1 times in column 5 grid number: 5
Number: 6 has occurred: 1 times in column 5 grid number: 5
Number: 7 has occurred: 0 times in column 5 grid number: 5
Number: 8 has occurred: 1 times in column 5 grid number: 5
Number: 9 has occurred: 1 times in column 5 grid number: 5
Number: 1 has occurred: 0 times in column 6 grid number: 6
Number: 2 has occurred: 0 times in column 6 grid number: 6
Number: 3 has occurred: 0 times in column 6 grid number: 6
Number: 4 has occurred: 1 times in column 6 grid number: 6
Number: 5 has occurred: 0 times in column 6 grid number: 6
Number: 6 has occurred: 2 times in column 6 grid number: 6

CURRENT GRID NUMBER: 6

5Coordinate match: 6 [1][6]with [3][6]

8 3 7 2 8 9 8 7 1

1 6 4 7 1 3 6 2 3

2 9 5 6 4 5 4 9 5

2 4 8 3 5 8 6 0 0

6 7 3 7 1 6 0 0 0

1 9 5 4 9 2 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 3

Storing this into blocked violation sequence: 1

1Blocked permutation sequence two standing top blocks at offset: 6 row: 3 [3, 1, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 9, 3, 10, 4, 1, 7, 2, 5])

5

3Blocked permutation sequence two standing top blocks in reverse: [1, 3, 0, 0, 0, 0, 0, 0, 0]

Number: 6 has occurred: 2 times in column 6 grid number: 6

Number: 2 has occurred: 2 times in column 0 grid number: 7

Number: 4 has occurred: 2 times in column 1 grid number: 7

CURRENT GRID NUMBER: 7

6Coordinate match: 4 [6][1] with block above

8 3 7 2 8 9 8 7 1

1 6 4 7 1 3 6 2 3

2 9 5 6 4 5 4 9 5

2 4 8 3 5 8 6 4 8

6 7 3 7 1 6 3 5 9

1 9 5 4 9 2 7 1 2

3 4 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 10

Storing this into blocked violation sequence: 7

2Blocked permutation bottom sequence second with third block in column: [10, 7, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 9, 3, 10, 4, 1, 7, 2, 5])

7

2Blocked permutation sequence second with third block in column reverse: [7, 10, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 9, 3, 10, 4, 1, 7, 2, 5])

Number: 9 has occurred: 2 times in column 1 grid number: 7

Number: 5 has occurred: 2 times in column 2 grid number: 7

Number: 1 has occurred: 2 times in column 0 grid number: 7

Number: 2 has occurred: 2 times in column 0 grid number: 7

Number: 4 has occurred: 2 times in column 1 grid number: 7

Number: 9 has occurred: 2 times in column 1 grid number: 7

Number: 5 has occurred: 2 times in column 2 grid number: 7

Number: 1 has occurred: 2 times in column 0 grid number: 7

Number: 2 has occurred: 2 times in column 0 grid number: 7

Number: 4 has occurred: 2 times in column 1 grid number: 7

Number: 7 has occurred: 2 times in column 1 grid number: 7

Number: 9 has occurred: 2 times in column 1 grid number: 7

Number: 7 has occurred: 2 times in column 3 grid number: 8

Number: 1 has occurred: 2 times in column 4 grid number: 8

Number: 4 has occurred: 2 times in column 4 grid number: 8

CURRENT GRID NUMBER: 8

4Coordinate match: 4 [2][4]with [6][4]

8 3 7 2 8 9 8 7 1

1 6 4 7 1 3 6 2 3

2 9 5 6 4 5 4 9 5

2 4 8 3 5 8 6 4 8

6 7 3 7 1 6 3 5 9

1 9 5 4 9 2 7 1 2

3 4 6 8 4 0 0 0 0

9 8 2 0 0 0 0 0 0

5 7 1 0 0 0 0 0 0

Storing this into blocked violation sequence: 9

Storing this into blocked violation sequence: 2

2Blocked permutation sequence first with third block in column: [9, 2, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 9, 3, 10, 4, 1, 7, 2, 5])

9

2Blocked permutation sequence first with third block in column reverse sequence: [2, 9, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 9, 3, 10, 4, 1, 7, 2, 5])

Number: 2 has occurred: 2 times in column 3 grid number: 8

Number: 7 has occurred: 2 times in column 3 grid number: 8

Number: 1 has occurred: 2 times in column 4 grid number: 8

Number: 4 has occurred: 2 times in column 4 grid number: 8

Number: 5 has occurred: 2 times in column 4 grid number: 8

Number: 6 has occurred: 2 times in column 5 grid number: 8

Number: 2 has occurred: 2 times in column 3 grid number: 8

Number: 7 has occurred: 2 times in column 3 grid number: 8

Number: 1 has occurred: 2 times in column 4 grid number: 8

Number: 4 has occurred: 2 times in column 4 grid number: 8

Number: 5 has occurred: 2 times in column 4 grid number: 8

Number: 3 has occurred: 2 times in column 5 grid number: 8

Number: 6 has occurred: 2 times in column 5 grid number: 8

Number: 6 has occurred: 2 times in column 6 grid number: 9

Number: 9 has occurred: 2 times in column 7 grid number: 9

STATE CHECK: 0 false

STATE CHECK: 1 false

CURRENT GRID NUMBER: 9

4Coordinate match: 9 [2][7]with [6][7]

8 3 7 2 8 9 8 7 1

1 6 4 7 1 3 6 2 3

2 9 5 6 4 5 4 9 5

2 4 8 3 5 8 6 4 8

6 7 3 7 1 6 3 5 9

1 9 5 4 9 2 7 1 2

3 4 6 8 4 1 5 9 0

9 8 2 2 5 6 0 0 0

5 7 1 9 7 3 0 0 0

Storing this into blocked violation sequence: 3

Storing this into blocked violation sequence: 5

2Blocked permutation sequence first with third block in column: [3, 5, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 9, 3, 10, 4, 1, 7, 2, 5])

11

2Blocked permutation sequence first with third block in column reverse sequence: [5, 3, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 9, 3, 10, 4, 1, 7, 2, 5])

Number: 6 has occurred: 2 times in column 6 grid number: 9

Number: 7 has occurred: 2 times in column 6 grid number: 9

Number: 4 has occurred: 2 times in column 7 grid number: 9

Number: 9 has occurred: 2 times in column 7 grid number: 9

Number: 3 has occurred: 2 times in column 8 grid number: 9

Number: 6 has occurred: 2 times in column 6 grid number: 9

Number: 7 has occurred: 2 times in column 6 grid number: 9

Number: 4 has occurred: 2 times in column 7 grid number: 9

Number: 9 has occurred: 2 times in column 7 grid number: 9

Number: 1 has occurred: 2 times in column 8 grid number: 9

Number: 3 has occurred: 2 times in column 8 grid number: 9

*****Current completed sudoku board(s): 0 out of

108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 Attempts: 1

NUMBER RECORDED PERMUTATION SEQUENCE ROW VIOLATIONS (includes duplicate entries): 0

NUMBER RECORDED PERMUTATION SEQUENCE COL VIOLATIONS (includes duplicate entries): 12

NUMBER ROW BLOCKED SEQUENCES IN EXECUTION: 0

NUMBER COL BLOCKED SEQUENCES IN EXECUTION: 0

SUCCESSFUL INPUTTED 3x3 GRIDS ONTO BOARD WITHOUT VIOLATION: 3

MAXIMUM INPUTTED SUCCESSFUL 3X3 GRIDS WITHOUT VIOLATION: 3

Number stored 3 sequence permutation in column (due error): 0

Number stored 3 sequence permutation in row (due error): 0

Better luck next time, failed on board attempt:0 Permutations selected: ([8, 9, 3, 10, 4, 1, 7, 2, 5])minimum: 1 maximum:10

8 3 7 2 8 9 8 7 1

1 6 4 7 1 3 6 2 3

2 9 5 6 4 5 4 9 5

2 4 8 3 5 8 6 4 8

6 7 3 7 1 6 3 5 9

1 9 5 4 9 2 7 1 2

3 4 6 8 4 1 5 9 6

9 8 2 2 5 6 7 4 3

5 7 1 9 7 3 2 8 1

Moving onto Board Number: 1

ColINDEX RIGHT NOW: 9

m right now: 0

We can see the data is clean and no repeats

-----CURRENT violating permutation column sequences: [8, 10, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [10, 8, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [9, 4, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [4, 9, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [3, 1, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [1, 3, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [10, 7, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [7, 10, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [9, 2, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [2, 9, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [3, 5, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [5, 3, 0, 0, 0, 0, 0, 0, 0]

** Process exited - Return Code: 0 **

I will now try to adapt same logic in the checkUniqueRows()

I will leave the logic intact but add the contingency similar to checkUniqueColumns()

TEST CASE: Checking performance of unique rows

```
if (totalNumbersProcessed%27==0 && (j==possibleNumbers.length-1) && occurrenceNumberRow==3 &&
    && (!hasViolationFirstTwoBlocks || !hasFirstWithThirdBlockViolate || !hasSecondWithThirdBlockViolate)) {
```

*** OUTPUT ***

Note: Permutation.java uses unchecked or unsafe operations.

Note: Recompile with -Xlint:unchecked for details.

Welcome to Online IDE!! Happy Coding :)

PERMUTATIONS

$P(n,r) = n! / (n-r)!$

$P(9,9) = 9! / (9-9)!$

There are : 362880 permutations of arranging 3 x 3 grid

There are : 108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 permutations of arranging 3 x 3 grid into 9 x 9: $P(362880,9)$

There are : 6,670,903,752,021,072,936,960 permutations of completing sudoku (taken from internet)

This code will attempt to explore but its impossible to expect much

It is used for foundation of experimentation but also it has made serious attempt to complete random process to make a grid

I am removing excess code so it is ready future development.

Number: 5 has occurred: 1 times in row 2 grid number: 1 //as expected on the first grid

Number: 6 has occurred: 0 times in row 2 grid number: 1

Number: 7 has occurred: 0 times in row 2 grid number: 1

Number: 8 has occurred: 1 times in row 2 grid number: 1

Number: 9 has occurred: 0 times in row 2 grid number: 1

Number: 1 has occurred: 0 times in row 0 grid number: 2

Number: 2 has occurred: 2 times in row 0 grid number: 2

1Coordinate match: 2 [0][1]with [0][3]

3 2 9 2 0 0 0 0 0

6 4 7 0 0 0 0 0 0

1 8 5 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

000000000

000000000

000000000

000000000

000000000

Storing this into blocked violation sequence: 9

Storing this into blocked violation sequence: 6

1Blocked permutation sequence first two blocks in row: [9, 6, 0, 0, 0, 0, 0, 0, 0] //This is correct

10

2

1Storing this into blocked violation sequence: 6

1Storing this into blocked violation sequence: 9

3Blocked permutation sequence first two blocks in row reverse: [6, 9, 0, 0, 0, 0, 0, 0, 0]

Number: 2 has occurred: 2 times in row 0 grid number: 2 //We can see we have not taken action anything on grid 2 again

Number: 2 has occurred: 2 times in row 0 grid number: 2

Number: 7 has occurred: 2 times in row 1 grid number: 2

Number: 7 has occurred: 2 times in row 1 grid number: 2

Number: 6 has occurred: 2 times in row 1 grid number: 2

Number: 7 has occurred: 2 times in row 1 grid number: 2

Number: 8 has occurred: 2 times in row 2 grid number: 2

Number: 8 has occurred: 2 times in row 2 grid number: 2

Number: 8 has occurred: 2 times in row 2 grid number: 2

Number: 5 has occurred: 2 times in row 0 grid number: 3 //it is ready to take action now

3Coordinate match: 5 [0][5]with [0][6]

329245500

647796000

185831000

000000000

000000000

000000000

000000000

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

4Storing this into blocked violation sequence: 6

4Storing this into blocked violation sequence: 8

2Blocked permutation sequence second with third block in row: [6, 8, 0, 0, 0, 0, 0, 0]

5Storing this into blocked violation sequence: 8

5Storing this into blocked violation sequence: 6

2Blocked permutation sequence second with third block in row reverse: [8, 6, 0, 0, 0, 0, 0, 0]

Number: 2 has occurred: 2 times in row 0 grid number: 3

Number: 5 has occurred: 2 times in row 0 grid number: 3

Number: 9 has occurred: 2 times in row 0 grid number: 3

2Coordinate match: 9 [0][2]with [0][7]

3 2 9 2 4 5 5 9 0

6 4 7 7 9 6 0 0 0

1 8 5 8 3 1 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

2Storing this into blocked violation sequence: 9

2Storing this into blocked violation sequence: 8

2Blocked permutation sequence first with third block in row : [9, 8, 0, 0, 0, 0, 0, 0]

3Storing this into blocked violation sequence: 8

3Storing this into blocked violation sequence: 9

2Blocked permutation first with third block in row reverse sequence: [8, 9, 0, 0, 0, 0, 0, 0]

Number: 2 has occurred: 2 times in row 0 grid number: 3

Number: 3 has occurred: 2 times in row 0 grid number: 3

Number: 5 has occurred: 2 times in row 0 grid number: 3

Number: 9 has occurred: 2 times in row 0 grid number: 3

Number: 6 has occurred: 2 times in row 1 grid number: 3

Number: 7 has occurred: 3 times in row 1 grid number: 3 //we can see it has taken no action here
//but I have not configured this logic the same as checkUniqueColumns() I will monitor see if this gets tackled

This is current violation

I need to question as to whether at 5,9,3 it has performed violation similar to 7,7,7

Since this would be only reason it has not tackled it

At **5** it has completed 2nd with 3rd block above

At **9** it has completed 1st with 3rd above

so at **7** no action is required

3 2 9 2 4 5 **5 9 3**

6 4 **7 7** 9 6 **7** 6 8

1 8 5 8 3 1 4 2 1

9 2 7 9 0 0 0 0 0

6 1 5 0 0 0 0 0 0

3 4 8 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Number: 6 has occurred: 3 times in row 1 grid number: 3 //no actions taken on grid 3 since all scenarios covered above

Number: 7 has occurred: 3 times in row 1 grid number: 3

Number: 6 has occurred: 3 times in row 1 grid number: 3

Number: 7 has occurred: 3 times in row 1 grid number: 3

Number: 1 has occurred: 2 times in row 2 grid number: 3

Number: 8 has occurred: 2 times in row 2 grid number: 3

Number: 1 has occurred: 2 times in row 2 grid number: 3

Number: 8 has occurred: 2 times in row 2 grid number: 3

Number: 1 has occurred: 3 times in row 2 grid number: 3

Number: 8 has occurred: 2 times in row 2 grid number: 3

Number: 8 has occurred: 1 times in row 5 grid number: 4 //and we expect no violation on first grid of next row

Number: 9 has occurred: 0 times in row 5 grid number: 4

Number: 1 has occurred: 0 times in row 3 grid number: 5

Number: 2 has occurred: 1 times in row 3 grid number: 5

Number: 3 has occurred: 0 times in row 3 grid number: 5

Number: 4 has occurred: 0 times in row 3 grid number: 5

Number: 5 has occurred: 0 times in row 3 grid number: 5

Number: 6 has occurred: 0 times in row 3 grid number: 5

Number: 7 has occurred: 1 times in row 3 grid number: 5

Number: 8 has occurred: 0 times in row 3 grid number: 5

Number: 9 has occurred: 2 times in row 3 grid number: 5

1Coordinate match: 9 [3][0]with [3][3]

3 2 9 2 4 5 5 9 3

6 4 7 7 9 6 7 6 8

1 8 5 8 3 1 4 2 1

9 2 7 9 0 0 0 0 0

6 1 5 0 0 0 0 0 0

3 4 8 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 5

Storing this into blocked violation sequence: 4

1Blocked permutation sequence first two blocks in row: [5, 4, 0, 0, 0, 0, 0, 0, 0] //This is fine

5

1Storing this into blocked violation sequence: 4

1Storing this into blocked violation sequence: 5

3Blocked permutation sequence first two blocks in row reverse: [4, 5, 0, 0, 0, 0, 0, 0]

Number: 9 has occurred: 2 times in row 3 grid number: 5 //no further actions on grid 5

Number: 9 has occurred: 2 times in row 3 grid number: 5

Number: 1 has occurred: 2 times in row 4 grid number: 5

Number: 1 has occurred: 2 times in row 4 grid number: 5

Number: 6 has occurred: 2 times in row 4 grid number: 5

Number: 8 has occurred: 2 times in row 5 grid number: 5

Number: 3 has occurred: 2 times in row 5 grid number: 5

Number: 8 has occurred: 2 times in row 5 grid number: 5

Number: 5 has occurred: 2 times in row 3 grid number: 6

3Coordinate match: 5 [3][5]with [3][6]

3 2 9 2 4 5 5 9 3

6 4 7 7 9 6 7 6 8

1 8 5 8 3 1 4 2 1

9 2 7 9 4 5 5 0 0

6 1 5 7 1 6 0 0 0

3 4 8 2 8 3 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

4Storing this into blocked violation sequence: 4

4Storing this into blocked violation sequence: 10

2Blocked permutation sequence second with third block in row: [4, 10, 0, 0, 0, 0, 0, 0] //This is fine

5Storing this into blocked violation sequence: 10

5Storing this into blocked violation sequence: 4

2Blocked permutation sequence second with third block in row reverse: [10, 4, 0, 0, 0, 0, 0, 0]

Number: 9 has occurred: 2 times in row 3 grid number: 6

Number: 5 has occurred: 2 times in row 3 grid number: 6

Number: 7 has occurred: 2 times in row 3 grid number: 6 //This has occurred since above we had 2nd with 3rd. Now we have 1st with 3rd

2Coordinate match: 7 [3][2]with [3][7]

3 2 9 2 4 5 5 9 3

6 4 7 7 9 6 7 6 8

1 8 5 8 3 1 4 2 1

9 2 7 9 4 5 5 7 0

6 1 5 7 1 6 0 0 0

3 4 8 2 8 3 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

2Storing this into blocked violation sequence: 5

2Storing this into blocked violation sequence: 10

2Blocked permutation sequence first with third block in row : [5, 10, 0, 0, 0, 0, 0, 0, 0]

3Storing this into blocked violation sequence: 10

3Storing this into blocked violation sequence: 5

2Blocked permutation first with third block in row reverse sequence: [10, 5, 0, 0, 0, 0, 0, 0, 0]

Number: 9 has occurred: 2 times in row 3 grid number: 6 //We expect no more activity with grid number 6 since both scenarios explored above with grid Number 6

Number: 5 has occurred: 2 times in row 3 grid number: 6

Number: 7 has occurred: 2 times in row 3 grid number: 6

Number: 9 has occurred: 2 times in row 3 grid number: 6

Number: 1 has occurred: 2 times in row 4 grid number: 6

Number: 6 has occurred: 2 times in row 4 grid number: 6

Number: 1 has occurred: 2 times in row 4 grid number: 6

Number: 6 has occurred: 2 times in row 4 grid number: 6

Number: 1 has occurred: 2 times in row 4 grid number: 6

Number: 6 has occurred: 2 times in row 4 grid number: 6

Number: 3 has occurred: 2 times in row 5 grid number: 6

Number: 8 has occurred: 2 times in row 5 grid number: 6

Number: 3 has occurred: 2 times in row 5 grid number: 6

Number: 4 has occurred: 2 times in row 5 grid number: 6

Number: 8 has occurred: 2 times in row 5 grid number: 6

Number: 3 has occurred: 3 times in row 5 grid number: 6

Number: 4 has occurred: 2 times in row 5 grid number: 6

Number: 8 has occurred: 2 times in row 5 grid number: 6

Number: 9 has occurred: 0 times in row 5 grid number: 6

Number: 8 has occurred: 1 times in row 8 grid number: 7 //No activity grid number 7 as expected

Number: 9 has occurred: 1 times in row 8 grid number: 7

Number: 1 has occurred: 0 times in row 6 grid number: 8

Number: 2 has occurred: 1 times in row 6 grid number: 8

Number: 3 has occurred: 1 times in row 6 grid number: 8

Number: 4 has occurred: 2 times in row 6 grid number: 8

1Coordinate match: 4 [6][2]with [6][3]

3 2 9 2 4 5 5 9 3

6 4 7 7 9 6 7 6 8

1 8 5 8 3 1 4 2 1

9 2 7 9 4 5 5 7 6

6 1 5 7 1 6 9 2 8

3 4 8 2 8 3 1 4 3

2 3 4 4 0 0 0 0 0

5 6 7 0 0 0 0 0 0

8 1 9 0 0 0 0 0 0

Storing this into blocked violation sequence: 1

Storing this into blocked violation sequence: 3

1Blocked permutation sequence first two blocks in row: [1, 3, 0, 0, 0, 0, 0, 0, 0]

64

8

1Storing this into blocked violation sequence: 3

1Storing this into blocked violation sequence: 1

3Blocked permutation sequence first two blocks in row reverse: [3, 1, 0, 0, 0, 0, 0, 0, 0]

Number: 4 has occurred: 2 times in row 6 grid number: 8 //No further actions Grid Number 8

Number: 4 has occurred: 2 times in row 6 grid number: 8

Number: 7 has occurred: 2 times in row 7 grid number: 8

Number: 7 has occurred: 2 times in row 7 grid number: 8

Number: 4 has occurred: 3 times in row 6 grid number: 9 //Need to understand why not picked up
//it is a new grid, so would expect a response. Most likely it is due to

```
if (occurrenceNumberCol==2) {
```

This was keeping
occurrenceNumberCol=3
exempt when it started the
third

```
for (int i=0; i<nineByNine.length; i++) {  
    //if (!hasFirstWithThirdBlockInColumnViolate[idx]) {  
    // {  
    if (occurrenceNumberCol>=2) {  
    }
```

Now changed to this

This means my reasoning was incorrect for the above test case
So I will quickly run checking again across the rows.
Once I am satisfied, I will tidy up the code.

TEST CASE: Running checkUniqueRows () again

Note: Permutation.java uses unchecked or unsafe operations.

Note: Recompile with -Xlint:unchecked for details.

Welcome to Online IDE!! Happy Coding :)

PERMUTATIONS

$P(n,r) = n! / (n-r)!$

$P(9,9) = 9! / (9-9)!$

There are : 362880 permutations of arranging 3 x 3 grid

There are : 108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 permutations of
arranging 3 x 3 grid into 9 x 9: $P(362880,9)$

There are : 6,670,903,752,021,072,936,960 permutations of completing sudoku (taken from internet)

This code will attempt to explore but its impossible to expect much

It is used for foundation of experimentation but also it has made serious attempt to complete random process to make a grid

I am removing excess code so it is ready future development.

Number: 9 has occurred: 2 times in row 0 grid number: 2

1Coordinate match: 9 [0][0]with [0][3]

9 1 3 9 0 0 0 0 0

2 6 8 0 0 0 0 0 0

7 4 5 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

Storing this into blocked violation sequence: 8

Storing this into blocked violation sequence: 6

1Blocked permutation sequence first two blocks in row: [8, 6, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 6, 7, 2, 10, 4, 3, 1, 5] – This is the final information and it does correspond correctly

10

2

1Storing this into blocked violation sequence: 6

1Storing this into blocked violation sequence: 8

3Blocked permutation sequence first two blocks in row reverse: [6, 8, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 6, 7, 2, 10, 4, 3, 1, 5] – This is the final information and it does correspond correctly

Number: 9 has occurred: 2 times in row 0 grid number: 2 //no more activity Grid 2

Number: 9 has occurred: 2 times in row 0 grid number: 2

Number: 2 has occurred: 2 times in row 1 grid number: 2

Number: 7 has occurred: 2 times in row 2 grid number: 2

Number: 4 has occurred: 2 times in row 2 grid number: 2

Number: 7 has occurred: 2 times in row 2 grid number: 2

Number: 4 has occurred: 2 times in row 2 grid number: 2

Number: 5 has occurred: 2 times in row 2 grid number: 2

Number: 7 has occurred: 2 times in row 2 grid number: 2

Number: 9 has occurred: 2 times in row 0 grid number: 3

Number: 1 has occurred: 2 times in row 0 grid number: 3

2Coordinate match: 1 [0][1]with [0][7]

9 1 3 9 8 6 4 1 0

2 6 8 3 1 2 0 0 0

7 4 5 7 4 5 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

2Storing this into blocked violation sequence: 8

2Storing this into blocked violation sequence: 7

2Blocked permutation sequence first with third block in row : [8, 7, 0, 0, 0, 0, 0, 0, 0] //this is ok

3Storing this into blocked violation sequence: 7

3Storing this into blocked violation sequence: 8

2Blocked permutation first with third block in row reverse sequence: [7, 8, 0, 0, 0, 0, 0, 0, 0]

Number: 9 has occurred: 2 times in row 0 grid number: 3

Number: 1 has occurred: 2 times in row 0 grid number: 3

Number: 9 has occurred: 2 times in row 0 grid number: 3

Number: 2 has occurred: 2 times in row 1 grid number: 3

Number: 2 has occurred: 2 times in row 1 grid number: 3

Number: 3 has occurred: 2 times in row 1 grid number: 3 //there is activity here since grid 2 and grid 3 are violating

3Coordinate match: 3 [1][3]with [1][7]

9 1 3 9 8 6 4 1 7

2 6 8 3 1 2 5 3 0

7 4 5 7 4 5 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

4 Storing this into blocked violation sequence: 6

4 Storing this into blocked violation sequence: 7

2 Blocked permutation sequence second with third block in row: [6, 7, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 6, 7, 2, 10, 4, 3, 1, 5] – This is the final information and it does correspond correctly

5 Storing this into blocked violation sequence: 7

5 Storing this into blocked violation sequence: 6

2 Blocked permutation sequence second with third block in row reverse: [7, 6, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 6, 7, 2, 10, 4, 3, 1, 5] – This is the final information and it does correspond correctly

Number: 2 has occurred: 3 times in row 1 grid number: 3 //it has processed grid 2 and grid 3 AND grid 1 and grid 3, so this is not required. Even though it is 3. So the code change above has had no adverse effect

Number: 3 has occurred: 2 times in row 1 grid number: 3

Number: 4 has occurred: 2 times in row 2 grid number: 3

Number: 5 has occurred: 2 times in row 2 grid number: 3

Number: 7 has occurred: 2 times in row 2 grid number: 3

Number: 4 has occurred: 2 times in row 2 grid number: 3

Number: 5 has occurred: 2 times in row 2 grid number: 3

Number: 7 has occurred: 2 times in row 2 grid number: 3

Number: 4 has occurred: 2 times in row 2 grid number: 3

Number: 5 has occurred: 2 times in row 2 grid number: 3

Number: 7 has occurred: 2 times in row 2 grid number: 3

Number: 8 has occurred: 0 times in row 5 grid number: 4 //Do not expect violation on grid 4

Number: 9 has occurred: 0 times in row 5 grid number: 4

Number: 1 has occurred: 0 times in row 3 grid number: 5 //it has found no violations grid 5 which is rare, I will just double check below to be sure. It is accurate

Number: 8 has occurred: 0 times in row 5 grid number: 5

Number: 9 has occurred: 1 times in row 5 grid number: 5

Number: 1 has occurred: 0 times in row 3 grid number: 6

Number: 2 has occurred: 1 times in row 3 grid number: 6

Number: 3 has occurred: 1 times in row 3 grid number: 6

Number: 4 has occurred: 1 times in row 3 grid number: 6

Number: 5 has occurred: 0 times in row 3 grid number: 6

Number: 6 has occurred: 1 times in row 3 grid number: 6

Number: 7 has occurred: 2 times in row 3 grid number: 6

2Coordinate match: 7 [3][0]with [3][6]

9 1 3 9 8 6 4 1 7

2 6 8 3 1 2 5 3 2

7 4 5 7 4 5 8 6 9

7 3 4 8 6 2 7 0 0

8 1 9 5 3 4 0 0 0

2 5 6 9 1 7 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

2Storing this into blocked violation sequence: 2

2Storing this into blocked violation sequence: 4

2Blocked permutation sequence first with third block in row : [2, 4, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 6, 7, 2, 10, 4, 3, 1, 5] – This is the final information and it does correspond correctly

3Storing this into blocked violation sequence: 4

3Storing this into blocked violation sequence: 2

2Blocked permutation first with third block in row reverse sequence: [4, 2, 0, 0, 0, 0, 0, 0, 0]

Number: 7 has occurred: 2 times in row 3 grid number: 6

Number: 2 has occurred: 2 times in row 3 grid number: 6 //it has taken action because 2nd and 3rd violate and not registered so far

3Coordinate match: 2 [3][5]with [3][8]

9 1 3 9 8 6 4 1 7

2 6 8 3 1 2 5 3 2

7 4 5 7 4 5 8 6 9

7 3 4 8 6 2 7 1 2

8 1 9 5 3 4 0 0 0

2 5 6 9 1 7 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

4Storing this into blocked violation sequence: 10

4Storing this into blocked violation sequence: 4

2Blocked permutation sequence second with third block in row: [10, 4, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 6, 7, 2, 10, 4, 3, 1, 5] – This is the final information and it does correspond correctly

5Storing this into blocked violation sequence: 4

5Storing this into blocked violation sequence: 10

2Blocked permutation sequence second with third block in row reverse: [4, 10, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 6, 7, 2, 10, 4, 3, 1, 5] – This is the final information and it does correspond correctly

Number: 7 has occurred: 2 times in row 3 grid number: 6

Number: 9 has occurred: 2 times in row 4 grid number: 6

Number: 9 has occurred: 2 times in row 4 grid number: 6

Number: 4 has occurred: 2 times in row 4 grid number: 6

Number: 9 has occurred: 2 times in row 4 grid number: 6

Number: 5 has occurred: 2 times in row 5 grid number: 6

NO ACTIVITY GRID 7 AS EXPECTED

Number: 3 has occurred: 2 times in row 8 grid number: 8

1Coordinate match: 3 [8][2]with [8][5]

9 1 3 9 8 6 4 1 7

2 6 8 3 1 2 5 3 2

7 4 5 7 4 5 8 6 9

7 3 4 8 6 2 7 1 2

8 1 9 5 3 4 9 6 4

2 5 6 9 1 7 3 8 5

2 5 9 7 6 4 0 0 0

6 7 8 2 1 5 0 0 0

4 1 3 9 8 3 0 0 0

Storing this into blocked violation sequence: 3

Storing this into blocked violation sequence: 1

1Blocked permutation sequence first two blocks in row: [3, 1, 0, 0, 0, 0, 0, 0, 0] //This is ok

Permutations selected: ([8, 6, 7, 2, 10, 4, 3, 1, 5] – This is the final information and it does correspond correctly

72

8

1Storing this into blocked violation sequence: 1

1Storing this into blocked violation sequence: 3

3Blocked permutation sequence first two blocks in row reverse: [1, 3, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 6, 7, 2, 10, 4, 3, 1, 5] – This is the final information and it does correspond correctly

Number: 4 has occurred: 2 times in row 6 grid number: 9

3Coordinate match: 4 [6][5]with [6][8]

9 1 3 9 8 6 4 1 7

2 6 8 3 1 2 5 3 2

7 4 5 7 4 5 8 6 9

7 3 4 8 6 2 7 1 2

8 1 9 5 3 4 9 6 4

2 5 6 9 1 7 3 8 5

2 5 9 7 6 4 1 3 4

6 7 8 2 1 5 0 0 0

4 1 3 9 8 3 0 0 0

4Storing this into blocked violation sequence: 1

4Storing this into blocked violation sequence: 5

2Blocked permutation sequence second with third block in row: [1, 5, 0, 0, 0, 0, 0, 0, 0] //this is ok
Permutations selected: ([8, 6, 7, 2, 10, 4, 3, 1,5] – This is the final information and it does correspond correctly

5Storing this into blocked violation sequence: 5

5Storing this into blocked violation sequence: 1

2Blocked permutation sequence second with third block in row reverse: [5, 1, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 6, 7, 2, 10, 4, 3, 1,5] – This is the final information and it does correspond correctly

Number: 5 has occurred: 2 times in row 7 grid number: 9 //no activity since already processed 2nd and 3rd block in row

Number: 5 has occurred: 2 times in row 7 grid number: 9 //no activity since already processed 2nd and 3rd block in row

Number: 8 has occurred: 2 times in row 7 grid number: 9 //it is exploring 1st and 3rd in row

2Coordinate match: 8 [7][2]with [7][7]

9 1 3 9 8 6 4 1 7

2 6 8 3 1 2 5 3 2

7 4 5 7 4 5 8 6 9

7 3 4 8 6 2 7 1 2

8 1 9 5 3 4 9 6 4

2 5 6 9 1 7 3 8 5

2 5 9 7 6 4 1 3 4

6 7 8 2 1 5 5 8 0

4 1 3 9 8 3 0 0 0

2Storing this into blocked violation sequence: 3

2Storing this into blocked violation sequence: 5

2Blocked permutation sequence first with third block in row : [3, 5, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 6, 7, 2, 10, 4, 3, 1,5] – This is the final information and it does correspond correctly

3Storing this into blocked violation sequence: 5

3Storing this into blocked violation sequence: 3

2Blocked permutation first with third block in row reverse sequence: [5, 3, 0, 0, 0, 0, 0, 0, 0]

Permutations selected: ([8, 6, 7, 2, 10, 4, 3, 1, 5] – This is the final information and it does correspond correctly

Number: 5 has occurred: 2 times in row 7 grid number: 9 //no activity here since it has already processed 2nd and 3rd block in row

Number: 8 has occurred: 2 times in row 7 grid number: 9

Number: 3 has occurred: 2 times in row 8 grid number: 9

Number: 3 has occurred: 2 times in row 8 grid number: 9

Number: 3 has occurred: 2 times in row 8 grid number: 9

*****Current completed sudoku board(s): 0 out of
108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 Attempts: 1

NUMBER RECORDED PERMUTATION SEQUENCE ROW VIOLATIONS (includes duplicate entries): 16

NUMBER RECORDED PERMUTATION SEQUENCE COL VIOLATIONS (includes duplicate entries): 0

NUMBER ROW BLOCKED SEQUENCES IN EXECUTION (BACK TRACKING): 0

NUMBER COL BLOCKED SEQUENCES IN EXECUTION (BACK TRACKING): 0

SUCCESSFUL INPUTTED 3x3 GRIDS ONTO BOARD WITHOUT VIOLATION: 1

//I can see that this is also incorrect, I experienced two blocks together where it did not violate

//But this was intermittent blocks

//It is a very delicate area to try and capture this in between.. I am more interested in this analysis from the start of the board, so technically this information is correct

MAXIMUM INPUTTED SUCCESSFUL 3X3 GRIDS WITHOUT VIOLATION: 1

Number stored 3 sequence permutation in column (due error): 0

Number stored 3 sequence permutation in row (due error): 0

//It is extremely important that what I have stored actually corresponds to these locations, so I will just check everything above for the rows

Better luck next time, failed on board attempt:0 Permutations selected: ([8, 6, 7, 2, 10, 4, 3, 1, 5])minimum: 1 maximum:10

9 1 3 9 8 6 4 1 7

2 6 8 3 1 2 5 3 2

7 4 5 7 4 5 8 6 9

7 3 4 8 6 2 7 1 2

8 1 9 5 3 4 9 6 4

2 5 6 9 1 7 3 8 5

2 5 9 7 6 4 1 3 4

6 7 8 2 1 5 5 8 9

4 1 3 9 8 3 7 2 6

Moving onto Board Number: 1

ColINDEX RIGHT NOW: 9 //not required

m right now: 0 //not required

We can see the data is clean and no repeats

-----CURRENT violating permutation row sequences: [8, 6, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [6, 8, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [8, 7, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [7, 8, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [6, 7, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [7, 6, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [2, 4, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [4, 2, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [10, 4, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [4, 10, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [3, 1, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [1, 3, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [1, 5, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [5, 1, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [3, 5, 0, 0, 0, 0, 0, 0, 0]
-----CURRENT violating permutation row sequences: [5, 3, 0, 0, 0, 0, 0, 0, 0]

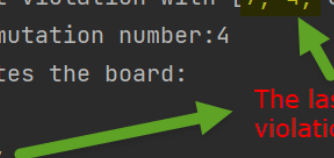
** Process exited - Return Code: 0 **

I am now at a point where I have tidied up my code and turned on all screen outputs that I think were genuinely useful during my development. I have run the code for two full board executions and I will try to remove any screen outputs that I think are not relevant....

But perhaps the most important part of this exercise. We need to officially test if backtracking is doing exactly its purpose.

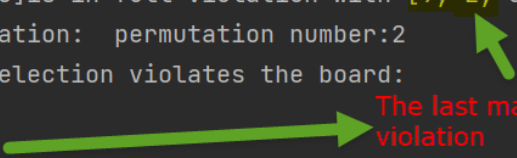
TEST CASE: Backtracking due to row violation

```
*****IMPORTANT INFORMATION ROW VIOLATION CHECK*****
Permutations selected: [7, 4, 0, 0, 0, 0, 0, 0, 0] Current selections
ROW SEGMENT: [7, 4, 0, 0, 0, 0, 0, 0, 0] Portion to be analysed
Violating RULE[h]: [7, 4, 0, 0, 0, 0, 0, 0, 0] Collated from previous
h=8 numberMatches=2 uniqueEntries=2 violation
Grid number: 1 BOTH IN AGREEMENT
Location of zero (unique entry): 3
[7, 4, 0, 0, 0, 0, 0, 0, 0] is in full violation with [7, 4, 0, 0, 0, 0, 0, 0, 0]
At index: 1 there is violation: permutation number:4
This row permutation selection violates the board:
Number unique entries: 2
****1BACKTRACKING TO UNIQUE ENTRY: 4,
CURRENT permutation numbers generated: [7, 0, 0, 0, 0, 0, 0, 0, 0] VALUE DROPPED OFF
*****GRID ALREADY SELECTED(7) GENERATING ANOTHER
CURRENT permutation numbers generated: [7, 0, 0, 0, 0, 0, 0, 0, 0]
*****NEW ENTRY: 2
Starting to check violations at second column: [2,0,0]
Starting to check violations at start first row: [7,2,0]
SEEING IF BACKTRACKING IS CORRECT (ROW VIOLATION):
This is current storeRetrieved3x3Grid: [2]
It is filling unique entry: [7, 2, 0, 0, 0, 0, 0, 0, 0] Issue resolved
```



TEST CASE: Backtracking due to column violation

```
*****IMPORTANT INFORMATION COLUMN VIOLATION CHECK*****
Permutations selected: [7, 4, 9, 5, 3, 2, 0, 0, 0] Current selections
COL SEGMENT: [9, 2, 0, 0, 0, 0, 0, 0, 0] Portion to be analysed
Violating RULE[h]:      [9, 2, 0, 0, 0, 0, 0, 0, 0]
h=0 numberMatchesCol=6 uniqueEntries=6 Collated from previous
Grid number: 1 BOTH IN AGREEMENT violation
Location of zero (unique entry): 7
[7, 4, 9, 5, 3, 2, 0, 0, 0] is in full violation with [9, 2, 0, 0, 0, 0, 0, 0, 0]
At index: 5 there is violation: permutation number:2
This column permutation selection violates the board:
Number unique entries: 6
****2BACKTRACKING TO: 2
CURRENT permutation numbers generated: [7, 4, 9, 5, 3, 0, 0, 0, 0]
```



7	4	9
5	3	2
0	0	0

The last match ascertains full violation

VALUE DROPPED OFF

```
CURRENT permutation numbers generated: [7, 4, 9, 5, 3, 2, 0, 0, 0]
*****GRID ALREADY SELECTED(2) GENERATING ANOTHER
CURRENT permutation numbers generated: [7, 4, 9, 5, 3, 2, 0, 0, 0]
*****GRID ALREADY SELECTED(4) GENERATING ANOTHER
CURRENT permutation numbers generated: [7, 4, 9, 5, 3, 2, 0, 0, 0]
*****GRID ALREADY SELECTED(5) GENERATING ANOTHER
CURRENT permutation numbers generated: [7, 4, 9, 5, 3, 2, 0, 0, 0]
*****GRID ALREADY SELECTED(5) GENERATING ANOTHER
CURRENT permutation numbers generated: [7, 4, 9, 5, 3, 2, 0, 0, 0]
*****GRID ALREADY SELECTED(7) GENERATING ANOTHER
CURRENT permutation numbers generated: [7, 4, 9, 5, 3, 2, 0, 0, 0]
*****GRID ALREADY SELECTED(5) GENERATING ANOTHER
CURRENT permutation numbers generated: [7, 4, 9, 5, 3, 2, 0, 0, 0]
*****NEW ENTRY: 8
Starting to check violations at third column: [9,8,0]
Starting to check violations at start second row: [5,3,8]
SEEING IF BACKTRACKING IS CORRECT (COLUMN VIOLATION):
This is current storeRetrieved3x3Grid: [8]
It is filling unique entry: [7, 4, 9, 5, 3, 8, 0, 0, 0]
```

Issue resolved

I am extremely glad I did not stop testing after first attempt on both, given a complex task

TEST CASE: Another execution for violation at column level

```
*****GRID ALREADY SELECTED(1) GENERATING ANOTHER
CURRENT permutation numbers generated: [6, 7, 1, 4, 8, 3, 5, 9, 0]
*****GRID ALREADY SELECTED(2) GENERATING ANOTHER
CURRENT permutation numbers generated: [6, 7, 1, 4, 8, 3, 5, 9, 0]
*****GRID ALREADY SELECTED(4) GENERATING ANOTHER
CURRENT permutation numbers generated: [6, 7, 1, 4, 8, 3, 5, 9, 0]
*****GRID ALREADY SELECTED(3) GENERATING ANOTHER
CURRENT permutation numbers generated: [6, 7, 1, 4, 8, 3, 5, 9, 0]
*****GRID ALREADY SELECTED(7) GENERATING ANOTHER
CURRENT permutation numbers generated: [6, 7, 1, 4, 8, 3, 5, 9, 0]
*****GRID ALREADY SELECTED(1) GENERATING ANOTHER
CURRENT permutation numbers generated: [6, 7, 1, 4, 8, 3, 5, 9, 0]
*****GRID ALREADY SELECTED(7) GENERATING ANOTHER
```

Code is in this state since it has executed continuously here.. I know that there was a 2 at the end and it is not available again for selection. This reminds me that I configured ALREADY PROCESSED to ensure it is not available for selection again.

I would need to retract this otherwise I am effectively losing grids for selection, which can cause this loop if there are insufficient available for selection.

This is prime example since I set

So there are 9 entries and we know numbers allowed are

1-9. All have been taken including 2 (since this was rejected

later on due to backtracking).... So it is rather fortunate I explored with this size. OTHERWISE I MIGHT HAVE NOT FOUND THIS UNDERLYING ISSUE!!!! And I would have effectively lost a selection each time when I backtracked

```
while (s.size() < 10);
```

So I had to go to extreme measures to ensure the most accuracy in getting the permutation grid back in operation...

```
--System.out.println("****1BACKTRACKING TO UNIQUE ENTRY: " + storeRetrieved3x3Grid[uniqueEntries-1]);
--
--
--int location = storeRetrieved3x3Grid[uniqueEntries-1]-1;
--System.out.println("LOCATION: " + location);
--
--
--System.out.println("Current contents as perm3x3Selection (using permutation number): " + perm3x3Selection[location]);
--System.out.println("Current contents as perm3x3Selection (using random number): " + perm3x3Selection[randomNumber-1]);
--//We know that index taken in perm3x3Selection which has already selected
--//is one index less than the actual value in storeRetrieved3x3Grid
--
--//We know random number is the permutation number, so we perform substitution (see location variable declared above)
--perm3x3Selection[randomNumber-1];
--
--//We also know that random number should be last permutation number selected
--perm3x3Selection[location]="";
--System.out.println("Contents perm3x3Selection should be empty and available for selection: " + perm3x3Selection[location]);
--System.out.println("Contents perm3x3Selection should be empty and available for selection: " + perm3x3Selection[randomNumber-1]);
--
--System.exit(0);
--
--storeRetrieved3x3Grid[uniqueEntries-1]=0;
--uniqueEntries = uniqueEntries - 1;
```

I also performed same operation on column operation.. Infact it was realised that it could be categorised into another method in interface... But for now, I will just duplicate the code, since I have had not changed the skeleton of the interface almost from the offset.

```

System.out.println("****2BACKTRACKING TO: " + storeRetrieved3x3Grid[uniqueEntries-1]);
locationRestoreBackTrack = storeRetrieved3x3Grid[uniqueEntries-1]-1;
System.out.println("LOCATION: " + locationRestoreBackTrack);

System.out.println("Current contents as perm3x3Selection (using permutation number): " + perm3x3Selection[locationRestoreBackTrack]);
System.out.println("Current contents as perm3x3Selection (using random number): " + perm3x3Selection[randomNumber-1]);
//We know that index taken in perm3x3Selection which has already selected
//is one index less than the actual value in storeRetrieved3x3Grid
//We know random number is the permutation number, so we perform substitution (see location variable declared above)
//perm3x3Selection[randomNumber-1]);
//We also know that random number should be last permutation number selected
perm3x3Selection[locationRestoreBackTrack]="";
System.out.println("Contents perm3x3Selection should be empty and available for selection: " + perm3x3Selection[locationRestoreBackTrack]);
System.out.println("Contents perm3x3Selection should be empty and available for selection: " + perm3x3Selection[(randomNumber-1)]);

storeRetrieved3x3Grid[uniqueEntries-1]=0;
uniqueEntries = uniqueEntries - 1;
//System.exit(0);

```

Once I stripped down my code and completed versions for maximum performance, and all relevant screen outputs... I noticed a significant error arising with it reaching the error section for storing the blocked violations. It happened for both rows and columns, it infact meant that I had to do vast more troubleshooting at various points, and I had to

```

public void displayViolationBlocks(int numAttempts)
{
    if (!hasExecuteOnce)
    {
        hasExecuteOnce=true;

        if (numAttempts%10000==0 && numAttempts!=0)
        //if(numAttempts==4)
        {
            for (int i=0; i<position;i++)
            {
                System.out.println("-----CURRENT violating permutation row sequences: "
                    + Arrays.toString(blockedPermutationNumberSequence[i]));
            }
        }
    }
}

```

IF the method is called as part of the routine code execution, it will pass in numAttempts and if 10,000 it will show all store.
For troubleshooting, I also passed in argument 10,000. Since there is shadowing numAttempts, it will not affect the current running numAttempts I could have performed given a different name to local variable.

TEST CASE: Investigating issue

I am examining one of the error situations in my code.

This refers to handling of a column. We can see that it is based at grid number 9.

So I have highlighted the entire column.

I will simply go through each block one number at a time (in order they are filled).

My focus is on the column violation, so I can simply skip to second block.

My thought process was the following, so I will try to understand if the data is already in the violations so far

```
if (totalNumbersProcessed%27==0 && (j==possibleNumbers.length-1)
    && occurrenceNumberCol==3
    && (!hasSecondWithThirdBlockInColumnViolate[idx]
        || !hasFirstWithThirdBlockInColumnViolate[idx]))
{
```

111517, 201571 - not recorded and also not reverse

111517, 209661 - recorded and also its reverse

Also looking at the data, there is lots repeats taking place.

It suggests that the booleans are not going exactly to plan.

I will need to revisit this testing. Its frustrating because it was fine during extensive testing.

ERROR - not handled violations in column correctly

8 6 1 3 5 1 6 1 4

2 4 9 2 9 8 5 2 9

7 5 3 4 6 7 7 3 8

7 2 6 5 9 3 6 7 1 //We would expect 111517, 201571 and reverse

4 9 8 7 1 2 3 8 9 //this would be skipped

1 3 5 4 6 8 4 5 2

6 7 4 9 3 5 7 3 5 //7 = first with third block in column 111517, 209661 and reverse

3 1 2 6 4 2 2 4 9 //3 = no action //9=no action

8 5 9 1 7 8 1 6 8

[42416, 79401, 111517, 234346, 350965, 201571, 139759, 324828, 209661]

CURRENT GRID NUMBER: 9

Storing this into blocked violation sequence: 111517

Storing this into blocked violation sequence: 201571

Storing this into blocked violation sequence: 209661

2Blocked permutation in column sequence: [111517, 201571, 209661, 0, 0, 0, 0, 0, 0]

Storing this into blocked violation sequence: 111517

Storing this into blocked violation sequence: 209661

Storing this into blocked violation sequence: 201571

2Blocked permutation in column sequence: [111517, 209661, 201571, 0, 0, 0, 0, 0, 0]

2Blocked permutation in column sequence: [111517, 209661, 201571, 0, 0, 0, 0, 0, 0]

Storing this into blocked violation sequence: 201571

Storing this into blocked violation sequence: 111517

Storing this into blocked violation sequence: 209661

2Blocked permutation in column sequence: [201571, 111517, 209661, 0, 0, 0, 0, 0, 0]

2Blocked permutation in column sequence: [201571, 111517, 209661, 0, 0, 0, 0, 0, 0]

Storing this into blocked violation sequence: 201571

Storing this into blocked violation sequence: 209661

Storing this into blocked violation sequence: 111517

2Blocked permutation in column sequence: [201571, 209661, 111517, 0, 0, 0, 0, 0, 0]

2Blocked permutation in column sequence: [201571, 209661, 111517, 0, 0, 0, 0, 0, 0]

Storing this into blocked violation sequence: 209661

Storing this into blocked violation sequence: 201571

Storing this into blocked violation sequence: 111517

2Blocked permutation in column sequence: [209661, 201571, 111517, 0, 0, 0, 0, 0, 0]

2Blocked permutation in column sequence: [209661, 201571, 111517, 0, 0, 0, 0, 0, 0]

Storing this into blocked violation sequence: 209661

Storing this into blocked violation sequence: 201571

Storing this into blocked violation sequence: 111517

2Blocked permutation in column sequence: [209661, 201571, 111517, 0, 0, 0, 0, 0, 0]

Storing this into blocked violation sequence: 209661

Storing this into blocked violation sequence: 111517

Storing this into blocked violation sequence: 201571

2Blocked permutation in column sequence: [209661, 111517, 201571, 0, 0, 0, 0, 0, 0]

2Blocked permutation in column sequence: [209661, 111517, 201571, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [94714, 130271, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [130271, 94714, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [239128, 210402, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [210402, 239128, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [94714, 257645, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [257645, 94714, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [130271, 257645, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [257645, 130271, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [94714, 257645, 0, 0, 0, 0, 0, 0, 0]

//repeat has started here to above

-----CURRENT violating permutation column sequences: [257645, 94714, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [94714, 257645, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [257645, 94714, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [94714, 257645, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [257645, 94714, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [94714, 257645, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [257645, 94714, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [94714, 257645, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [257645, 94714, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation column sequences: [94714, 257645, 0, 0, 0, 0, 0, 0, 0]

I realised two massive errors, during my code tidy up and also working on back tracking section, I had accidentally commented out crucial line.


```
//System.out.println("*****NEW ENTRY: " + randomNumber);  
perm3x3Selection[randomNumber-1]="ALREADY SELECTED";
```

But when I set the statement around

```
System.out.println("****CHECKS");
System.out.println("occurrenceNumberCol: " + occurrenceNumberCol);
System.out.println("hasFirstWithThirdBlockInColumnViolate[idx]:" + hasFirstWithThirdBlockInCo
System.out.println(rowIndexMatch);

if (occurrenceNumberCol>=2 && !hasFirstWithThirdBlockInColumnViolate[idx])
{
    for (int i=0; i<nineByNine.length; i++)
    {
        if (occurrenceNumberCol>=2)
        {
            if (possibleNumbers[j]==nineByNine[i][colIndex])
            {
                System.out.println("CHECK HERE");
                if (i<=2)
                {

```



1

This was remaining as true so it also prevented the column blocked sequences (for legitimate pairs) to fill the blocked violations

Hence it did not enter here.

2

I decided to perform the reset here instead

```
public boolean sudokuComplete(boolean duplicateNumbersRow, boolean duplicateNumbersCol)
{
    hasRegisteredInvalidPermutationSequenceRow=false;
    hasRegisteredInvalidPermutationSequenceColumn=false;
    gridNumber=1;
    uniqueEntries=0;
    successfulInputted3x3=0;

    numberOf3x3Processed=0;
    pos=0;
    marker=0;
    m=0;

    hasFirstWithThirdBlockInColumnViolate=new boolean[3];

```

I ran the testing up much higher number boards and there were no longer errors shown. I also checked the violated boards for columns, and there were no duplicates..

I am now at a point where I do not want to exhaust my code anymore with these comments and will just remove them back out of the code.

TEST CASE: Running code further and checking inconsistency row violations

It has not added a single entry into the blocked violations, so I will similarly set up similar troubleshooting to columns

ERROR - not handled violations in row correctly

[199311, 166924, 297415, 278463, 63932, 166752, 261547, 289287, 205811]

5 2 4 1 6 9 4 7 1 //expected this [199311, 297415]

3 9 6 4 2 5 2 9 6

8 7 1 7 3 8 8 5 3 [199311, 166924], [166924, 297415]

8 1 4 1 7 9 3 8 2 , [278463, 63932]

6 9 2 2 6 8 9 4 5 [278463 166752]

7 5 3 3 4 5 6 1 7

4 5 7 5 4 2 2 8 4 [261547 205811] [261547, 289287]

3 6 2 3 1 7 1 7 5 [289287, 205811]

1 8 9 9 8 6 3 9 6

CURRENT GRID NUMBER: 9

Storing this into blocked violation sequence: 261547

Storing this into blocked violation sequence: 289287

Storing this into blocked violation sequence: 205811

2Blocked permutation sequence entire row: (Grid number(7),Grid number(8),Grid number(9) [261547, 289287, 205811, 0, 0, 0, 0, 0, 0]

Storing this into blocked violation sequence: 261547

Storing this into blocked violation sequence: 205811

Storing this into blocked violation sequence: 289287

2Blocked permutation sequence entire row: (Grid number(7),Grid number(8),Grid number(9) [261547, 205811, 289287, 0, 0, 0, 0, 0, 0]

2Blocked permutation sequence: [289287, 261547, 205811, 0, 0, 0, 0, 0, 0]

Storing this into blocked violation sequence: 289287

Storing this into blocked violation sequence: 261547

Storing this into blocked violation sequence: 205811

2Blocked permutation sequence entire row: (Grid number(7),Grid number(8),Grid number(9) [289287, 261547, 205811, 0, 0, 0, 0, 0, 0]

Storing this into blocked violation sequence: 0

Storing this into blocked violation sequence: 0

Storing this into blocked violation sequence: 261547

2Blocked permutation sequence entire row: (Grid number(7),Grid number(8),Grid number(9) [289287, 205811, 261547, 0, 0, 0, 0, 0, 0]

Storing this into blocked violation sequence: 0

Storing this into blocked violation sequence: 0

Storing this into blocked violation sequence: 261547

2Blocked permutation sequence entire row: (Grid number(7),Grid number(8),Grid number(9) [205811, 289287, 261547, 0, 0, 0, 0, 0, 0]

2Blocked permutation sequence: [205811, 261547, 289287, 0, 0, 0, 0, 0, 0]

Storing this into blocked violation sequence: 0

Storing this into blocked violation sequence: 0

Storing this into blocked violation sequence: 289287

2Blocked permutation sequence entire row: [205811, 261547, 289287, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [185090, 61265, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [61265, 185090, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [61265, 336869, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [336869, 61265, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [185090, 336869, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [336869, 185090, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [149229, 69157, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [69157, 149229, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [69157, 62364, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [62364, 69157, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [149229, 62364, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [62364, 149229, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [183105, 247870, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [247870, 183105, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [205997, 247870, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [247870, 205997, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [323889, 268372, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [268372, 323889, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [12912, 264444, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [264444, 12912, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [264444, 103369, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [103369, 264444, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [12912, 103369, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [103369, 12912, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [175883, 344985, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [344985, 175883, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [344985, 87761, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [87761, 344985, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [175883, 87761, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [87761, 175883, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [280599, 336968, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [336968, 280599, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [336968, 53364, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [53364, 336968, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [280599, 53364, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [53364, 280599, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [136499, 94463, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [94463, 136499, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [260387, 94463, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [94463, 260387, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [111219, 21829, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [21829, 111219, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [113500, 237839, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [237839, 113500, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [295493, 15796, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [15796, 295493, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [295493, 225184, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [225184, 295493, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [15796, 225184, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [225184, 15796, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [195825, 323507, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [323507, 195825, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [195825, 39773, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [39773, 195825, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [323507, 39773, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [39773, 323507, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [349559, 215238, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [215238, 349559, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [215238, 100834, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [100834, 215238, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [349559, 100834, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [100834, 349559, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [214089, 196169, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [196169, 214089, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [196169, 313942, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [313942, 196169, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [214089, 313942, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [313942, 214089, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [62965, 323087, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [323087, 62965, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [323087, 318431, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [318431, 323087, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [62965, 318431, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [318431, 62965, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [87148, 169023, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [169023, 87148, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [169023, 208416, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [208416, 169023, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [235800, 274223, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [274223, 235800, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [235800, 157691, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [157691, 235800, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [274223, 157691, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [157691, 274223, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [29985, 343490, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [343490, 29985, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [343490, 285604, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [285604, 343490, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [29985, 285604, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [285604, 29985, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [274899, 348818, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [348818, 274899, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [348818, 133518, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [133518, 348818, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [274899, 133518, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [133518, 274899, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [122822, 234351, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [234351, 122822, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [225359, 234351, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [234351, 225359, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [32925, 236163, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [236163, 32925, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [236163, 97487, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [97487, 236163, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [98186, 251205, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [251205, 98186, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [98186, 315822, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [315822, 98186, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [251205, 315822, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [315822, 251205, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [31085, 332438, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [332438, 31085, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [31085, 295437, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [295437, 31085, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [332438, 295437, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [295437, 332438, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [288251, 351402, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [351402, 288251, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [351402, 311993, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [311993, 351402, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [288251, 311993, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [311993, 288251, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [73575, 309761, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [309761, 73575, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [309761, 156357, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [156357, 309761, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [73575, 156357, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [156357, 73575, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [69796, 144467, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [144467, 69796, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [144467, 126988, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [126988, 144467, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [69796, 126988, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [126988, 69796, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [79715, 289817, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [289817, 79715, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [208613, 289817, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [289817, 208613, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [142138, 34041, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [34041, 142138, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [202027, 63395, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [63395, 202027, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [63395, 335216, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [335216, 63395, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [202027, 335216, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [335216, 202027, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [355179, 346435, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [346435, 355179, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [346435, 229567, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [229567, 346435, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [355179, 229567, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [229567, 355179, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [179259, 340526, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [340526, 179259, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [340526, 351498, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [351498, 340526, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [179259, 351498, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [351498, 179259, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [85849, 2454, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [2454, 85849, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [85849, 291839, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [291839, 85849, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [2454, 291839, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [291839, 2454, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [278463, 63932, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [63932, 278463, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [278463, 166752, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [166752, 278463, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [261547, 289287, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [289287, 261547, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [289287, 205811, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [205811, 289287, 0, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [261547, 289287, 205811, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [261547, 205811, 289287, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [289287, 261547, 205811, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [289287, 205811, 261547, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [205811, 289287, 261547, 0, 0, 0, 0, 0, 0]

-----CURRENT violating permutation row sequences: [205811, 261547, 289287, 0, 0, 0, 0, 0, 0]

```

switch(gridNumber){
{
case 1:
rowIndex=0;
hasProcessedThirdBlockInColumn=new boolean[3];

hasFirstWithThirdBlockInColumnViolate=new boolean[3];
hasSecondWithThirdBlockInColumnViolate=new boolean[3];
hasViolationTwoStandingTopBlocks=new boolean[3];

case 4:
rowIndex=3;

hasProcessedThirdBlockInRow=false;
hasViolationFirstTwoBlocks=false;
hasFirstWithThirdBlockViolate=false;
hasSecondWithThirdBlockViolate=false;
break;

case 7:
rowIndex=6;

hasProcessedThirdBlockInRow=false;
hasViolationFirstTwoBlocks=false;
hasFirstWithThirdBlockViolate=false;
hasSecondWithThirdBlockViolate=false;
}
}

```

I previously had not even had a reset of the variables in case 1: where rowIndex=0 So this was first huge mistake. And also when I did move it there, it had complete wrong outcome

```

public boolean sudokuComplete(boolean duplicateNumbersRow, boolean duplicateNumbersCol)
{
    gridNumber=1;
    uniqueEntries=0;
    successfulInputted3x3=0;

    numberOf3x3Processed=0;
    pos=0;
    marker=0;
    m=0;

    hasFirstWithThirdBlockInColumnViolate=new boolean[3];

    hasProcessedThirdBlockInRow=false;
    hasViolationFirstTwoBlocks=false;
    hasFirstWithThirdBlockViolate=false;
    hasSecondWithThirdBlockViolate=false;
}

```

I moved the variables here

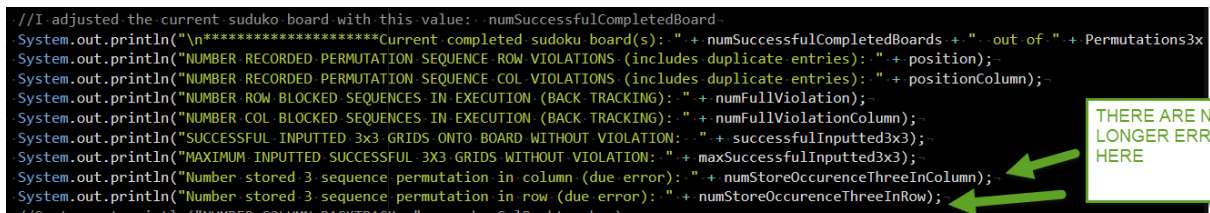
```

System.out.println("****CHECKS TO TROUBLESHOOT FATAL ISSUES WITH LOGGING ROW VIOLATION");
System.out.println("occurenceNumberRow: " + occurenceNumberRow);
System.out.println("offset: " + offset);
System.out.println("Grid: " + gridNumber);
System.out.println("columnIndexMatch: " + columnIndexMatch);
System.out.println("!hasViolationFirstTwoBlocks (true condition to enter loop): " + !hasViolationFirstTwoBlocks);
System.out.println("should be true !hasFirstWithThirdBlockViolate: " + !hasFirstWithThirdBlockViolate);
System.out.println("should be true to enter !hasSecondWithThirdBlockViolate: " + !hasSecondWithThirdBlockViolate);

```

At long last the code is running smoothly,

```
//I adjusted the current suduko board with this value: numSuccessfulCompletedBoard-
System.out.println("\n*****Current completed sudoku board(s): " + numSuccessfulCompletedBoards + ". out of " + Permutations3x3);
System.out.println("NUMBER RECORDED PERMUTATION SEQUENCE ROW VIOLATIONS (includes duplicate entries): " + position);
System.out.println("NUMBER ROW BLOCKED SEQUENCES IN EXECUTION (BACK TRACKING): " + numFullViolation);
System.out.println("NUMBER COL BLOCKED SEQUENCES IN EXECUTION (BACK TRACKING): " + numFullViolationColumn);
System.out.println("SUCCESSFUL INPUTTED 3x3 GRIDS ONTO BOARD WITHOUT VIOLATION: " + successfulInputted3x3);
System.out.println("MAXIMUM INPUTTED SUCCESSFUL 3X3 GRIDS WITHOUT VIOLATION: " + maxSuccessfulInputted3x3);
System.out.println("Number stored 3 sequence permutation in column (due error): " + numStoreOccurenceThreeInColumn);
System.out.println("Number stored 3 sequence permutation in row (due error): " + numStoreOccurenceThreeInRow);
```



So one final thought has come to mind.

I have done a bit above on tracking back.

But what is the status of failed row or failed column at that time and the impact on successful inputted 3x3

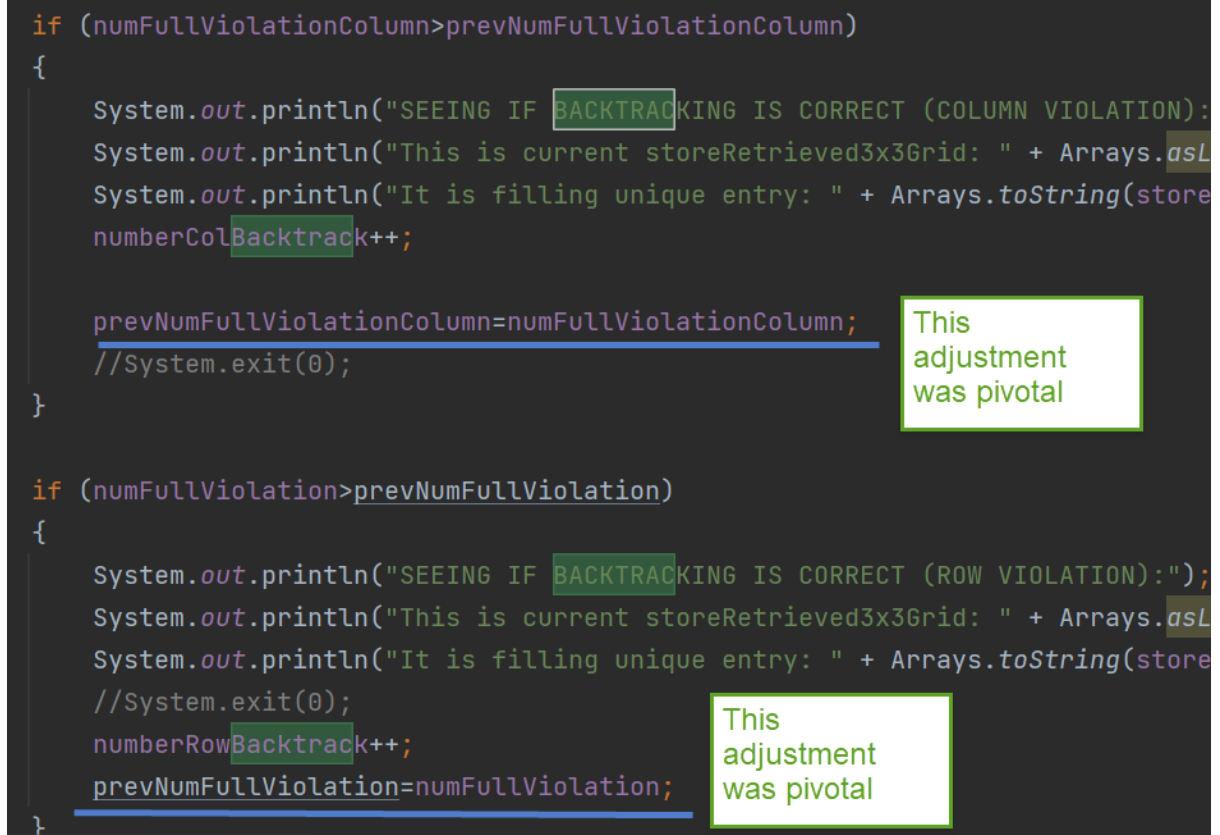
TEST CASE: Exploring backtracking and state of variables

To get this information on the screen, I have to set the perm3x3 to be 10 or something quite low. I also realised a blunder in which it was still showing the backtracking over and over again once it had completed. It required an adjustment to the variable as below.

```
if (numFullViolationColumn>prevNumFullViolationColumn)
{
    System.out.println("SEEING IF BACKTRACKING IS CORRECT (COLUMN VIOLATION):");
    System.out.println("This is current storeRetrieved3x3Grid: " + Arrays.asList(storeRetrieved3x3Grid));
    System.out.println("It is filling unique entry: " + Arrays.toString(storeRetrieved3x3Grid));
    numberColBacktrack++;

    prevNumFullViolationColumn=numFullViolationColumn;
    //System.exit(0);
}

if (numFullViolation>prevNumFullViolation)
{
    System.out.println("SEEING IF BACKTRACKING IS CORRECT (ROW VIOLATION):");
    System.out.println("This is current storeRetrieved3x3Grid: " + Arrays.asList(storeRetrieved3x3Grid));
    System.out.println("It is filling unique entry: " + Arrays.toString(storeRetrieved3x3Grid));
    //System.exit(0);
    numberRowBacktrack++;
    prevNumFullViolation=numFullViolation;
}
```



I executed my code through few thousand executions and there was no backtracking. This is still perfectly understandable. But in my thought process, I felt it might be worthwhile to have a troubleshooting area where it can confirm if two consecutive grids on row/column conform to sudoku. I was still hitting this scenario quite a bit. So I am keeping an allowedPermutationSequenceRow and allowedPermutationSequenceCol.

This will slow down the execution and it also meant I had to split the array size between four arrays to manage the heap. but it would be good to see which is occurring the most across the board, failed pairs or successful pairs. If I wanted, I could also break it down to see if first with second OR second with third is most problematic OR first with third is issue. We know if none violate, it is a successful row/column, which in itself would be approx 1/50,000. I have taken approach to add this into ALLOW array since it will be in minority / non-existent. Taking the option to add three or more violating sequences (from start of board) will weigh down array quickly and hinder execution time. I can consider thinking about longer blocked permutation sequences once it actually performs a back track.

TEST CASE: Running enhanced code and my observations:

FIRST INSTANCE OF 3 SUCCESSFUL INPUTTED GRIDS

```

*****Current completed sudoku board(s): 0 out of 108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 Attempts: 14585
RECORDED ENTRIES (ROW VIOLATIONS, includes bi-direction of pairs): 256682
RECORDED ENTRIES (COL VIOLATIONS, includes bi-direction of pairs): 84660
NUMBER ROW BLOCKED SEQUENCES IN EXECUTION (REQUIRED BACK TRACKING): 0
NUMBER COL BLOCKED SEQUENCES IN EXECUTION (REQUIRED BACK TRACKING): 0
SUCCESSFUL INPUTTED 3x3 GRIDS ONTO BOARD WITHOUT VIOLATION: 3
MAXIMUM INPUTTED SUCCESSFUL 3X3 GRIDS (FROM GRID 1) WITHOUT VIOLATION: 3
Number stored (3 sequence permutation into VIOLATIONS) in col (due error): 0
Number stored (3 sequence permutation into VIOLATIONS) in row (due error): 0
Frequency (2 sequence permutation into ALLOWED) in row (ANYWHERE on board): 1019
Frequency (2 sequence permutation into ALLOWED) in col (ANYWHERE on board): 0
Number stored (3 sequence permutation into ALLOWED) in row (ANYWHERE on board): 1
Number stored (3 sequence permutation into ALLOWED) in col (ANYWHERE on board): 0
Better luck next time, failed on board attempt:0
Permutations selected: ([89099, 12109, 165330, 123471, 174841, 131724, 202306, 77290, 115694]) minimum: 4 maximum:362879
Moving onto Board Number: 1
  
```

2Allowed permutation sequence (first and third) in row: (Grid number(1),Grid number(3), [89099, 165330, 0, 0, 0, 0, 0, 0, 0])

2Allowed permutation sequence (third and first) in row: (Grid number(3),Grid number(1), [165330, 89099, 0, 0, 0, 0, 0, 0, 0])

****CHECKS TO TROUBLESHOOT FATAL ISSUES WITH THIRD BLOCK IN ROW AND PREVIOUS TWO CONFORM TO SUDOKU

[89099, 12109, 165330, 123471, 174841, 131724, 202306, 77290, 115694]

2Allowed permutation sequence (first and second) in row: (Grid number(1),Grid number(2), [89099, 12109, 0, 0, 0, 0, 0, 0, 0])

2Allowed permutation sequence (second and first) in row: (Grid number(2),Grid number(1), [12109, 89099, 0, 0, 0, 0, 0, 0, 0])

2Allowed permutation sequence (second and third) in row : (Grid number(2),Grid number(3), [12109, 165330, 0, 0, 0, 0, 0, 0, 0])

2Allowed permutation sequence (third and second) in row : (Grid number(3),Grid number(2), [165330, 12109, 0, 0, 0, 0, 0, 0, 0])

Full permitted permutation sequence (first second and third) in row : (Grid number(1),Grid number(2),Grid number(3), [89099, 12109, 165330, 0, 0, 0, 0, 0, 0])

There is still no backtracking all the way up over

So it has reached this on basis of approx 1/50,000 chance of hitting three in row

No errors, which is excellent news

I knew there was 3.3% chance of getting two consecutive blocks conforming to the rules. So I found this to be a bit too low for execution. I found this to be incorrect. But there were no

There is excellent coverage almost the entire range

```

9 5 8 1 3 2 4 6 7
4 2 3 7 6 5 9 8 1
6 7 1 9 4 8 3 5 2
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
          
```

And we can see, this is legitimate and correctly added into ALLOWED. We could potentially store

We could potentially add this into the blocked array when situation arises, but the chance of hitting exact same first three again (any order) are very improbable $3! \times P(392,880 \times 3)$

```

Permutations selected: ([89099, 12109, 165330, 123471, 174841, 131724, 202306, 77290, 115694]) minimum: 4 maximum:362879
  
```

FIRST INSTANCE OF 4 SUCCESSFUL INPUTTED GRIDS

```
*****Current completed sudoku board(s): 0 out of 108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 Attempts: 19101
RECORDED ENTRIES (ROW VIOLATIONS, includes bi-direction of pairs): 335974
RECORDED ENTRIES (COL VIOLATIONS, includes bi-direction of pairs): 110752
NUMBER ROW BLOCKED SEQUENCES IN EXECUTION (REQUIRED BACK TRACKING): 0
NUMBER COL BLOCKED SEQUENCES IN EXECUTION (REQUIRED BACK TRACKING): 0
SUCCESSFUL INPUTTED 3x3 GRIDS ONTO BOARD WITHOUT VIOLATION: 4
MAXIMUM INPUTTED SUCCESSFUL 3X3 GRIDS (FROM GRID 1) WITHOUT VIOLATION: 4
Number stored (3 sequence permutation into VIOLATIONS) in col (due error): 0
Number stored (3 sequence permutation into VIOLATIONS) in row (due error): 0
Frequency (2 sequence permutation into ALLOWED) in row (ANYWHERE on board): 1358
Frequency (2 sequence permutation into ALLOWED) in col (ANYWHERE on board): 0
Number stored (3 sequence permutation into ALLOWED) in row (ANYWHERE on board): 2
Number Stored (3 sequence permutation into ALLOWED) in col (ANYWHERE on board): 0
Better luck next time, failed on board attempt:0
Permutations selected: ([37799, 46804, 268413, 64278, 339814, 296316, 103118, 284264, 165182]) minimum: 4 maximum:362879
Moving onto Board Number: 1
```

Again no backtracking, I find this extraordinary given that algorithm configured to execute across all 9 positions of uniqueEntry. I had no issue on smaller perm3x3

This was an even bigger milestone getting 4 grids compliant in a row

```
2Allowed permutation sequence (first and third) in row: (Grid number(1),Grid number(3), [37799, 268413, 0, 0, 0, 0, 0, 0, 0]
2Allowed permutation sequence (third and first) in row: (Grid number(3),Grid number(1), [268413, 37799, 0, 0, 0, 0, 0, 0, 0]
****CHECKS TO TROUBLESHOOT FATAL ISSUES WITH THIRD BLOCK IN ROW AND PREVIOUS TWO CONFORM TO SUDOKU
[37799, 46804, 268413, 64278, 339814, 296316, 103118, 284264, 165182]
2Allowed permutation sequence (first and second) in row: (Grid number(1),Grid number(2), [37799, 46804, 0, 0, 0, 0, 0, 0, 0]
2Allowed permutation sequence (second and first) in row: (Grid number(2),Grid number(1), [46804, 37799, 0, 0, 0, 0, 0, 0, 0]
2Allowed permutation sequence (second and third) in row : (Grid number(2),Grid number(3), [46804, 268413, 0, 0, 0, 0, 0, 0, 0]
2Allowed permutation sequence (third and second) in row : (Grid number(3),Grid number(2), [268413, 46804, 0, 0, 0, 0, 0, 0, 0]
Full permitted permutation sequence (first second and third) in row : (Grid number(1),Grid number(2),Grid number(3), [37799, 46804, 268413, 0, 0, 0, 0, 0, 0]
0]
```

```
5 7 1 3 8 4 6 2 9
6 4 3 9 1 2 8 7 5
9 2 8 6 5 7 4 3 1
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

This would be state of board when it performs first row analysis. So I need to perform display9x9() when successful inputs exceeds 3. To ensure I can 100% verify all is accurate

```
*****Current completed sudoku board(s): 0 out of 108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 Attempts: 19101
RECORDED ENTRIES (ROW VIOLATIONS, includes bi-direction of pairs): 335974
RECORDED ENTRIES (COL VIOLATIONS, includes bi-direction of pairs): 110752
NUMBER ROW BLOCKED SEQUENCES IN EXECUTION (REQUIRED BACK TRACKING): 0
NUMBER COL BLOCKED SEQUENCES IN EXECUTION (REQUIRED BACK TRACKING): 0
SUCCESSFUL INPUTTED 3x3 GRIDS ONTO BOARD WITHOUT VIOLATION: 4
MAXIMUM INPUTTED SUCCESSFUL 3X3 GRIDS (FROM GRID 1) WITHOUT VIOLATION: 4
Number stored (3 sequence permutation into VIOLATIONS) in col (due error): 0
Number stored (3 sequence permutation into VIOLATIONS) in row (due error): 0
Frequency (2 sequence permutation into ALLOWED) in row (ANYWHERE on board): 1358
Frequency (2 sequence permutation into ALLOWED) in col (ANYWHERE on board): 0
Number stored (3 sequence permutation into ALLOWED) in row (ANYWHERE on board): 2
Number Stored (3 sequence permutation into ALLOWED) in col (ANYWHERE on board): 0
Better luck next time, failed on board attempt:0
Permutations selected: ([37799, 46804, 268413, 64278, 339814, 296316, 103118, 284264, 165182]) minimum: 4 maximum:362879
```

TEST CASE: Adding logic to support addition of a four streak chain and populating the Allow array

```
//Given the code logic, since the fourth grid will overflow into two rows or two columns-
//it is not possible to know the starting grid. So difficult to place in the Allowed array.
//But for now, displaying information on screen suffices.
if (successfulInputted3x3>3)
{
    System.out.println("***Successful: " + successfulInputted3x3 + " consecutive permutation numbers");
    display9x9();
    System.out.println(Arrays.toString(storeRetrieved3x3Grid));
}
```

TEST CASE: Investigating with smaller perm3x3 and ascertaining if backtracking actually functions. I will try 10,000 as opposed to 362,880 and see difference.

We know there are no issues with perm3x3 consisting of 10.

```
*****Current completed sudoku board(s): 0 out of 108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 Attempts: 522
RECORDED ENTRIES (ROW VIOLATIONS, includes bi-direction of pairs): 91906
RECORDED ENTRIES (COL VIOLATIONS, includes bi-direction of pairs): 30324
NUMBER ROW BLOCKED SEQUENCES IN EXECUTION (REQUIRED BACK TRACKING): 0
NUMBER COL BLOCKED SEQUENCES IN EXECUTION (REQUIRED BACK TRACKING): 0
SUCCESSFUL INPUTTED 3x3 GRIDS ONTO BOARD WITHOUT VIOLATION: 1
MAXIMUM INPUTTED SUCCESSFUL 3X3 GRIDS (FROM GRID 1) WITHOUT VIOLATION: 2
Number stored (3 sequence permutation into VIOLATIONS) in col (due error): 1
Number stored (3 sequence permutation into VIOLATIONS) in row (due error): 0
Frequency (2 sequence permutation into ALLOWED) in row (ANYWHERE on board): 410
Frequency (2 sequence permutation into ALLOWED) in col (ANYWHERE on board): 0
Number stored (3 sequence permutation into ALLOWED) in row (ANYWHERE on board): 0
Number stored (3 sequence permutation into ALLOWED) in col (ANYWHERE on board): 0
Better luck next time, failed on board attempt:0
Permutations selected: ([150, 6218, 217, 5780, 600, 4373, 7392, 1654, 3074]) minimum: 1 maximum:9999
Moving onto Board Number: 1
```

It has found an error, so it is extremely high priority to figure out scenario. I should have all screen outputs for this scenario by default

```
Frequency (2 sequence permutation into ALLOWED) in row (ANYWHERE on board): 0
Frequency (2 sequence permutation into ALLOWED) in col (ANYWHERE on board): 0
Number stored (3 sequence permutation into ALLOWED) in row (ANYWHERE on board): 0
Number stored (3 sequence permutation into ALLOWED) in col (ANYWHERE on board): 0
Better luck next time, failed on board attempt:0
Permutations selected: ([8924, 8374, 2678, 337, 9451, 5962, 2753, 7146, 8646]) minimum: 337 maximum:9451
Moving onto Board Number: 1
ERROR - not handled violations in column correctly
CURRENT GRID NUMBER: 9
storeRetrieved3x3Grid:[1765, 2205, 8440, 9597, 2408, 761, 5200, 2959, 9943]
1 4 2 8 7 1 4 2 1
8 9 5 5 9 6 7 8 6
7 3 6 2 3 4 3 5 9
5 9 1 2 9 7 6 5 3
4 6 2 3 6 8 1 2 8
3 7 8 1 4 5 7 4 9
1 9 8 1 8 2 8 6 9
4 2 5 3 5 4 7 4 2
6 3 7 7 6 9 1 5 3
Storing this into blocked violation sequence: 8440
Storing this into blocked violation sequence: 761
Storing this into blocked violation sequence: 9943
2Blocked permutation in column sequence: [8440, 761, 9943, 0, 0, 0, 0, 0, 0]
Storing this into blocked violation sequence: 8440
Storing this into blocked violation sequence: 9943
Storing this into blocked violation sequence: 761
2Blocked permutation in column sequence: [8440, 9943, 761, 0, 0, 0, 0, 0, 0]
Storing this into blocked violation sequence: 761
Storing this into blocked violation sequence: 8440
Storing this into blocked violation sequence: 9943
2Blocked permutation in column sequence: [761, 8440, 9943, 0, 0, 0, 0, 0, 0]
Storing this into blocked violation sequence: 761
```

It has failed to perform this, I am totally unsure of reason. So in this section of code, I have printed out status of all booleans and will analyse information

We expect all of these to be false since three

```
--if ((totalNumbersProcessed==63 || totalNumbersProcessed==72 || totalNumbersProcessed==81) && (j==possibleNumbers.length-1) --
--&& occurrenceNumberCol==3 --
--&& (!hasSecondWithThirdBlockInColumnViolate[idx] || !hasFirstWithThirdBlockInColumnViolate[idx] || !hasViolationTwoStandingTopBlocks[
--{
--    System.out.println("ERROR --not handled violations in column correctly");
--    System.out.println("**All of these should be false");
--    System.out.println("!hasSecondWithThirdBlockInColumnViolate[idx]: " + !hasSecondWithThirdBlockInColumnViolate[idx]);
--    System.out.println("!hasFirstWithThirdBlockInColumnViolate[idx]: " + !hasFirstWithThirdBlockInColumnViolate[idx]);
--    System.out.println("!hasViolationTwoStandingTopBlocks[idx]: " + !hasViolationTwoStandingTopBlocks[idx]);
--}
```

I have now included this within the section

TEST CASE: Running exact same execution again until same error flares up

There are : 362880 permutations of arranging 3 x 3 grid
There are : 108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 permutations of arranging 3 x 3 grid into 9 x 9:P(362880,9)
There are : 6,670,903,752,021,072,936,960 permutations of completing sudoku (taken from internet)
This code will attempt to explore but its impossible to expect much
It is used for foundation of experimentation but also it has made serious attempt to complete random process to make a grid
I am removing excess code so it is ready future development.
ERROR - not handled violations in column correctly
**All of these should be false
!hasSecondWithThirdBlockInColumnViolate[idx]: true
!hasFirstWithThirdBlockInColumnViolate[idx]: false
!hasViolationTwoStandingTopBlocks[idx]: false
CURRENT GRID NUMBER: 7
storeRetrieved3x3Grid:[765, 5109, 6270, 98, 651, 5175, 3925, 3474, 639]
6 2 9 5 3 9 5 2 3
5 7 8 6 7 4 7 4 6
3 4 1 2 1 8 9 8 1
3 7 8 6 7 1 2 1 4
6 1 2 4 9 3 3 9 5
4 5 9 2 8 5 8 6 7
1 2 7 0 0 0 0 0 0
8 3 4 0 0 0 0 0 0
6 5 9 0 0 0 0 0 0

We can see that it has missed out three chances to perform this operation, so I will go back to the code area closely.

All these conditions would be exactly true when it enters into the if loop

```
if ((occurenceNumberCol==3||occurenceNumberCol==2) && rowIndexMatch>=6 && !hasSecondWithThirdBlockInColumnViolate[idx])~  
{~  
    //System.out.println("STATE CHECK: "+i+" "+hasSecondWithThirdBlockInColumnViolate[idx]);~  
    if (i>=6 && !hasSecondWithThirdBlockInColumnViolate[idx] && !hasFirstWithThirdBlockInColumnViolate[idx])~  
    {~  
        //System.out.println("CURRENT GRID NUMBER: "+gridNumber);~  
        //System.out.println("6Coordinate match: "+possibleNumbers[j]+" "+[" "+rowIndexMatch+""] [" "+colIndex+""]~  
        //display9x9());~  
        hasSecondWithThirdBlockInColumnViolate[idx] = true;~  
        blockedPermutationNumberColumnSequence[positionColumn][0]=storeRetrieved3x3Grid[gridNumber-4];~  
        blockedPermutationNumberColumnSequence[positionColumn][1]=storeRetrieved3x3Grid[gridNumber-1];~  
    }~  
}
```

I would expect this to be correct since we are in nineBynine[6], nineBynine[7], nineBynine[8],

1

This is exactly same as outer if loop and boolean state has not been changed. But no detrimental effect

2

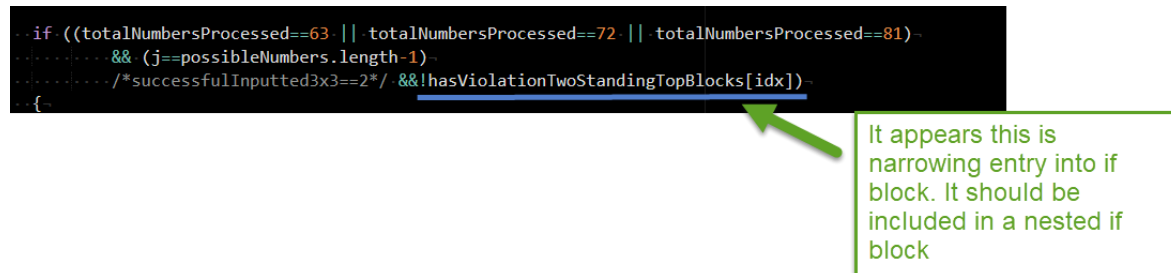
This is main issue, performing second with third block analysis has NO relation to first and third... This bit of logic has to be removed altogether. Infact both can be erased

TEST CASE: Running code again with critical change. I am hoping it might be reason of large gap between these

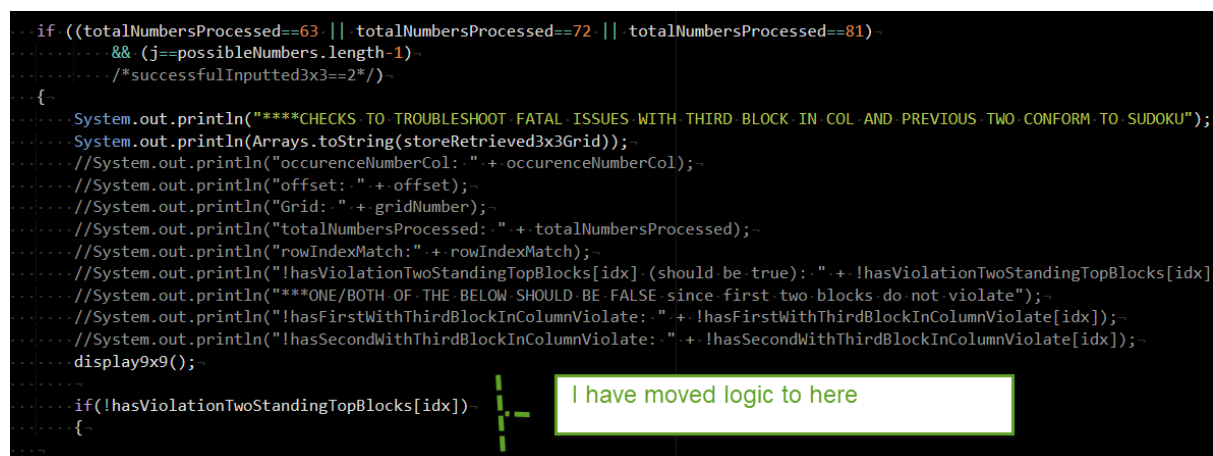
```
System.out.println("RECORDED ENTRIES (ROW VIOLATIONS, includes bi-direction of pairs): "+position);~  
System.out.println("RECORDED ENTRIES (COL VIOLATIONS, includes bi-direction of pairs): "+positionColumn);~
```

I am still getting odd issues of not getting any allowed columns (spanning two blocks).

Having examined my code in the area, I found this:



And fixed as below:



TEST CASE: Executing code again:

It all appears to be fine and here is my output

```
*****Current completed sudoku board(s): 0 out of 108,883,584,818,776,183,656,945,007,213,012,309,135,068,193,536,000 Attempts: 1
RECORDED ENTRIES (ROW VIOLATIONS, includes bi-direction of pairs): 18
RECORDED ENTRIES (COL VIOLATIONS, includes bi-direction of pairs): 18
NUMBER ROW BLOCKED SEQUENCES IN EXECUTION (REQUIRED BACK TRACKING): 0
NUMBER COL BLOCKED SEQUENCES IN EXECUTION (REQUIRED BACK TRACKING): 0
SUCCESSFUL INPUTTED 3x3 GRIDS ONTO BOARD WITHOUT VIOLATION: 1
MAXIMUM INPUTTED SUCCESSFUL 3X3 GRIDS (FROM GRID 1) WITHOUT VIOLATION: 1
Number stored (3 sequence permutation into VIOLATIONS) in col (due error): 0
Number stored (3 sequence permutation into VIOLATIONS) in row (due error): 0
Frequency (2 sequence permutation into ALLOWED) in row (ANYWHERE on board): 0 0.0%
Frequency (2 sequence permutation into ALLOWED) in col (ANYWHERE on board): 0 0.0%
Number stored (3 sequence permutation into ALLOWED) in row (ANYWHERE on board): 0
Number Stored (3 sequence permutation into ALLOWED) in col (ANYWHERE on board): 0
Better luck next time, failed on board attempt:0
Permutations selected: ([266052, 222825, 177092, 52917, 184804, 313787, 173676, 46295, 182387]) minimum: 46295 maximum:313787
Moving onto Board Number: 1
****CHECKS TO TROUBLESHOOT FATAL ISSUES WITH THIRD BLOCK IN COL AND PREVIOUS TWO CONFORM TO SUDOKU
[198075, 235576, 101221, 48919, 255623, 206852, 338111, 113048, 279488]
****CHECKS TO TROUBLESHOOT FATAL ISSUES WITH THIRD BLOCK IN COL AND PREVIOUS TWO CONFORM TO SUDOKU
[198075, 235576, 101221, 48919, 255623, 206852, 338111, 113048, 279488]
****CHECKS TO TROUBLESHOOT FATAL ISSUES WITH THIRD BLOCK IN COL AND PREVIOUS TWO CONFORM TO SUDOKU
[198075, 235576, 101221, 48919, 255623, 206852, 338111, 113048, 279488]
```


But after 20,000 executions

```
ERROR - not handled violations in row correctly
All of these should be false
!hasViolationFirstTwoBlocks: false
!hasFirstWithThirdBlockViolate: false
!hasSecondWithThirdBlockViolate: true
storeRetrieved3x3Grid: [206417, 253427, 291536, 134028, 23018, 1236, 103830, 293104, 125349]
9 4 7 4 7 5 4 6 2
8 5 2 1 9 8 9 1 3
6 1 3 6 3 2 5 7 8

5 3 1 1 5 3 4 6 7
7 2 6 7 6 2 3 1 8
4 8 9 4 9 8 5 2 9

0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
CURRENT GRID NUMBER: 6
Storing this into blocked violation sequence: 134028
Storing this into blocked violation sequence: 23018
```

```

*****Current completed sudoku board(s): 0 Attempts: 22241
RECORDED ENTRIES (ROW VIOLATIONS, includes bi-direction of pairs): 391094
RECORDED ENTRIES (COL VIOLATIONS, includes bi-direction of pairs): 129104
NUMBER ROW BLOCKED SEQUENCES IN EXECUTION (REQUIRED BACK TRACKING): 0
NUMBER COL BLOCKED SEQUENCES IN EXECUTION (REQUIRED BACK TRACKING): 0
SUCCESSFUL INPUTTED 3x3 GRIDS ONTO BOARD WITHOUT VIOLATION: 1
MAXIMUM INPUTTED SUCCESSFUL 3x3 GRIDS (FROM GRID 1) WITHOUT VIOLATION: 2
Number stored (3 sequence permutation into VIOLATIONS) in col (due error): 0
Number stored (3 sequence permutation into VIOLATIONS) in row (due error): 2
Frequency (2 sequence permutation into ALLOWED) in row (ANYWHERE on board): 1560
Frequency (2 sequence permutation into ALLOWED) in col (ANYWHERE on board): 4354
Number stored (3 sequence permutation into ALLOWED) in row (ANYWHERE on board): 0
Number Stored (3 sequence permutation into ALLOWED) in col (ANYWHERE on board): 0
Better luck next time, failed on board attempt: 0

```

chatGPT also worked out for me that in all of the arrays, there were no duplicates. Except for 13 in column allows. So this proves the backtracking

```

ERROR - not handled violations in row correctly
All of these should be false
!hasViolationFirstTwoBlocks: false
!hasFirstWithThirdBlockViolate: false
!hasSecondWithThirdBlockViolate: true
storeRetrieved3x3Grid: [206417, 253427, 291536, 134028, 23018, 1236,
9 4 7 4 7 5 4 6 2
8 5 2 1 9 8 9 1 3
6 1 3 6 3 2 5 7 8

5 3 1 1 5 3 4 6 7
7 2 6 7 6 2 3 1 8
4 8 9 4 9 8 5 2 9

0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0

```

1
I went through these outputs and I had to physically extract from output file during execution.

I put data of all arrays into separate worksheet and asked and then manually performed searched

CURRENT violating permutation row sequences: [23018, 1236]

2
No entry was found so it was understandable
hasSecondWithThirdBlockViolate=false

Sheet5 — duplicates found (13 unique duplicate entries)

- CURRENT violating permutation column sequences: [351264, 219] (count: 2, rows: 13382, 13383)
- CURRENT violating permutation column sequences: [347306, 213] (count: 2, rows: 14116, 14117)
- CURRENT violating permutation column sequences: [204778, 197] (count: 2, rows: 14156, 14157)
- CURRENT violating permutation column sequences: [204778, 197] (count: 2, rows: 14626, 14627)
- CURRENT violating permutation column sequences: [347306, 213] (count: 2, rows: 14666, 14667)
- CURRENT violating permutation column sequences: [204778, 197] (count: 2, rows: 14706, 14707)
- CURRENT violating permutation column sequences: [347306, 213] (count: 2, rows: 15076, 15077)
- CURRENT violating permutation column sequences: [204778, 197] (count: 2, rows: 15116, 15117)
- CURRENT violating permutation column sequences: [351264, 219] (count: 2, rows: 15156, 15157)
- CURRENT violating permutation column sequences: [341833, 209] (count: 2, rows: 15605, 15606)
- CURRENT violating permutation column sequences: [204778, 197] (count: 2, rows: 15645, 15646)
- CURRENT violating permutation column sequences: [341833, 209] (count: 2, rows: 15685, 15686)
- CURRENT violating permutation column sequences: [351264, 219] (count: 2, rows: 15725, 15726)

This was another situation where it went into error handling.

```
.....
Moving onto Board Number: 1
***CHECKS TO TROUBLESHOOT FATAL ISSUES WITH THIRD BLOCK IN COL AND PREVIOUS TWO CONFORM TO SUDOKU
[217520, 60792, 347461, 1413, 168840, 218085, 318949, 256659, 298799]
***CHECKS TO TROUBLESHOOT FATAL ISSUES WITH THIRD BLOCK IN COL AND PREVIOUS TWO CONFORM TO SUDOKU
[217520, 60792, 347461, 1413, 168840, 218085, 318949, 256659, 298799]
ERROR - not handled violations in row correctly
All of these should be false
!hasViolationFirstTwoBlocks: false
!hasFirstWithThirdBlockViolate: false
!hasSecondWithThirdBlockViolate: true
storeRetrieved3x3Grid: [217520, 60792, 347461, 1413, 168840, 218085, 318949, 256659, 298799]
6 1 4 8 5 3 7 3 6
8 5 3 7 4 2 4 8 2
9 2 7 6 1 9 5 9 1
9 2 8 7 9 1 9 7 2
6 7 3 8 3 6 3 8 4
5 4 1 4 5 2 5 6 1
7 4 5 7 2 6 8 1 3
8 1 3 3 5 4 2 7 6
6 9 2 1 9 8 4 5 9
CURRENT GRID NUMBER: 9
Storing this into blocked violation sequence: 318949
Storing this into blocked violation sequence: 256659
```

Same issue again, chatGPT has informed me that issue was related to using break in the method... And also advised to give me understanding if my check unique columns was susceptible

I went to ChatGPT for guidance and it pointed me in right direction. Up to this point, I had not taken on board any code offerings.

What *actually* causes it

Your bug happens whenever all of these are true:

1. You are placing the **3rd block in a 3-block row** (`offset == 6`) — i.e., you're working on "grid 6" (or grid 3 / 6 / 9 in general for a band).
2. For some number `x`, that row ends up containing `x` in:
 - **block 1** (cols 0–2),
 - **block 2** (cols 3–5),
 - **block 3** (cols 6–8),so `occurrenceNumberRow == 3`.
3. Your inner loop scans left-to-right and hits the **block-1 occurrence first** (`i <= 2`), enters:

```
java                                                                    Copy code

if (!hasFirstWithThirdBlockViolate && i <= 2) {
    hasFirstWithThirdBlockViolate = true;
    ... store (gridNumber-3, gridNumber-1)
    break;
}
```

That `break` exits the `for (i...)` loop **before** you ever set:

```
java                                                                    Copy code

hasSecondWithThirdBlockViolate = true; // (block2 vs block3)
```

✓ The safe fix is:

- **First-third** should only trigger when the current `i` is in block 1 **and** the other occurrence is in block 3.
- **Second-third** should only trigger when the current `i` is in block 2 **and** the other occurrence is in block 3.

Example structure (no `break` needed, and no wrong pair):

```
java                                                                    Copy code

if (offset == 6 && columnIndexMatch >= 6) {
    if (!hasFirstWithThirdBlockViolate && i >= 0 && i <= 2) {
        // store (gridNumber-3, gridNumber-1)
        hasFirstWithThirdBlockViolate = true;
    } else if (!hasSecondWithThirdBlockViolate && i >= 3 && i <= 5) {
        // store (gridNumber-2, gridNumber-1)
        hasSecondWithThirdBlockViolate = true;
    }
}
```

I already had `i <= 2`, so no adjustments required

I introduced these two conditions

I implemented this change and perhaps since the number processed was always looping, I assumed the break was actually a good choice and would not block the end outcome.

But this point, I was also quite adamant to determine if the underlying issue was stemming into my `checkUniqueColumns`.

I recall during design my intention was replicate exact row check for column check but I deviated.

ISSUES DESCRIBED BY CHATGPT

Yep — **your column logic has the same *shape* of problem as the row one**, but it's a bit sneakier:

1) Your “second-with-third” block is triggered on the wrong i range

This is the important part you pasted:

```
if ((occurenceNumberCol==3||occurenceNumberCol==2)
    && rowIndexMatch>=6
    && !hasSecondWithThirdBlockInColumnViolate[idx])
{
    if (i>=6)
    {
        hasSecondWithThirdBlockInColumnViolate[idx] = true;

        blockedPermutationNumberColumnSequence[positionColumn][0] =
storeRetrieved3x3Grid[gridNumber-4];

        blockedPermutationNumberColumnSequence[positionColumn][1] =
storeRetrieved3x3Grid[gridNumber-1];
        ...
        break;
    }
}
```

But conceptually:

- **“second-with-third” = middle block vs bottom block**
- Middle block row range = i in 3..5
- Bottom block row range = i in 6..8

Right now you only execute that code when $i \geq 6$ (bottom rows). That means:

- you are **not tying the trigger to actually seeing the number in the middle block**
- and for `occurenceNumberCol==2` with matches in **top+bottom only**, your code can still store a **middle+bottom** violation even if the middle block never

contained the number (because you only check `rowIndexMatch >= 6`, which just means “last match is in bottom”).

That can directly explain “missing expected pair, but weird other pair stored” situations.

✅ **Fix (mirror what you did in rows):**

Only store second-with-third when you are iterating the *middle* rows:

```
if ((occurenceNumberCol >= 2)
    && rowIndexMatch >= 6
    && !hasSecondWithThirdBlockInColumnViolate[idx]
    && (i >= 3 && i <= 5)
    && possibleNumbers[j] == nineByNine[i][colIndex])
{
    hasSecondWithThirdBlockInColumnViolate[idx] = true;

    blockedPermutationNumberColumnSequence[positionColumn][0] =
storeRetrieved3x3Grid[gridNumber-4];

    blockedPermutationNumberColumnSequence[positionColumn][1] =
storeRetrieved3x3Grid[gridNumber-1];

    positionColumn++;

    blockedPermutationNumberColumnSequence[positionColumn][0] =
storeRetrieved3x3Grid[gridNumber-1];

    blockedPermutationNumberColumnSequence[positionColumn][1] =
storeRetrieved3x3Grid[gridNumber-4];

    positionColumn++;

    // no break needed
}
```

That single change makes your “second-with-third” logging **only happen when the duplicate actually exists in the middle block.**

2) Your break here can cut off other detection for the same possibleNumbers[j]

Even if you keep the logic correct, the `break;` exits the `for (i=0; i<9; i++)` loop immediately after storing one case.

If the same number occurs 3 times, you may want to log **more than one relationship** (depending on your design). A `break` makes it easy to log only the first one you happen to hit.

Given your row fix, I'd recommend: **remove the break**, and rely on the booleans to prevent double-storing.

3) Your “two standing top blocks” check is using `rowIndex` (board fill position), not the match positions

This line:

```
if ((rowIndex>=3) && !hasViolationTwoStandingTopBlocks[idx])
```

`rowIndex` is where you're currently writing into the 9×9 board, not where the duplicates in this column actually are. You really want to decide based on **rowIndexMatch** (and the current `i`) instead.

A safer condition for “top+middle” is:

```
if (rowIndexMatch >= 3 && rowIndexMatch <= 5 &&  
!hasViolationTwoStandingTopBlocks[idx])
```

(meaning: “the last occurrence is in the middle block”.)

The big takeaway

Even if “only the last block has break”, **the bigger correctness issue is the `i>=6` trigger** for “second-with-third”. It can cause:

- false logging (middle+bottom when only top+bottom existed)
- missing the pair you expected
- error sections firing because the boolean never flipped (since the real middle match never triggered the store)

I was content now to accept its verdict.

ISSUES DESCRIBED BY CHATGPT

ok I am getting massive difference across 790 boards~ RECORDED ENTRIES (ROW VIOLATIONS, includes bi-direction of pairs): 13834 RECORDED ENTRIES (COL VIOLATIONS, includes bi-direction of pairs): 2307

Yeah — that size of gap usually means you're **not logging column violations at the same "rate"/conditions** as row violations (not that Sudoku magically violates rows 6× more).

There are **two big structural reasons in your code that can easily create exactly this kind of imbalance**:

1) checkUniqueColumns() is effectively "half-disabled" for most columns because idx is stale

Inside checkUniqueColumns you only set idx when:

```
if (colIndex==0) idx=0;
```

```
if (colIndex==3) idx=1;
```

```
if (colIndex==6) idx=2;
```

But your realTime9x9Fill() calls checkUniqueColumns(formattedBoard, colIndex) for **every** column 0..8.

So for colIndex = 1,2,4,5,7,8 you **never update idx** — it keeps whatever value it had last (0/1/2). That means your boolean gates like:

```
if (occurenceNumberCol>=2 && !hasFirstWithThirdBlockInColumnViolate[idx])
```

can be checking the wrong "stack bucket" and accidentally **suppressing logging** (or logging under the wrong bucket), which will dramatically reduce positionColumn.

✅ Fix (simple, reliable):

At the top of checkUniqueColumns, do:

```
int idx = colIndex / 3; // 0 for cols 0..2, 1 for 3..5, 2 for 6..8
```

...and **remove** the if (colIndex==0/3/6) ladder entirely.

Or, if your whole column-violation logic is only intended to run on the 3×3 boundaries, then guard it:

if (colIndex % 3 != 0) return failedColumns;

But right now you're in the worst middle-ground: called for all columns, but indexed like only 0/3/6 exist.

2) You're not calling the row/col checks using the *actual cell that was just written*

In `realTime9x9Fill`, you correctly compute:

`rowtoInt, coltoInt`

...but then you call:

`checkUniqueRows(formattedBoard, rowIndex);`

`checkUniqueColumns(formattedBoard, colIndex);`

Those are **class fields** that represent your fill-loop state, not necessarily the (`rowtoInt`, `coltoInt`) of the last placed number (especially when you're hopping within 3×3 placement logic).

This can cause “extra” row checks on the wrong row (or repeated checks on the same row) and inflate row logging.

✅ Fix:

Call the checks using the parsed coordinates:

`checkUniqueRows(formattedBoard, rowtoInt);`

`checkUniqueColumns(formattedBoard, coltoInt);`

This alone can change your row-vs-column counts a lot.

Why the gap looks like “Row >> Column”

- **Rows are being checked/logged more often and more correctly**
- **Columns are being gated by incorrect/stale idx**, so many column violations never get recorded into `blockedPermutationNumberColumnSequence`

So no — it's *not* “normal”, and it's *not* a Sudoku property. It's almost certainly an indexing/call-site mismatch.

If you make just these two changes, your `positionColumn` should jump up a lot and become much closer (not identical, but closer) to row counts.

If you paste your updated `realTime9x9Fill()` and the very top of `checkUniqueColumns()` after the edits, I can sanity-check that you've wired them in the safest way.

what change is needed to `realtime9x9fill`

In `realTime9x9Fill`, you're **calling the validators with the wrong coordinates**.

Right now you parse the *actual* cell you just wrote into `rowtoInt` / `coltoInt` (or `row` / `col` in the first-case), but then you call:

```
checkUniqueRows(formattedBoard, rowIndex);
```

```
checkUniqueColumns(formattedBoard, colIndex);
```

`rowIndex` and `colIndex` are your **global fill-loop state**, not guaranteed to match the cell you just placed (especially with your 3×3-to-9×9 mapping logic). That can cause false/missed checks and skew your logging.

The change

Call the checks using the **actual parsed coordinates** of the last inserted value.

Replace these two lines:

```
checkUniqueRows(formattedBoard, rowIndex);
```

```
checkUniqueColumns(formattedBoard, colIndex);
```

With this:

```
checkUniqueRows(formattedBoard, rowtoInt);
```

```
checkUniqueColumns(formattedBoard, coltoInt);
```

Full corrected `realTime9x9Fill` (only the relevant edits)

```
public void realTime9x9Fill(String history)
```

```
{
```

```
    int startLastNumber;
```

```
    int rowtoInt=0;
```

```
    int coltoInt=0;
```

```
    int boardValuetToInt=0;
```

```
    if (history.lastIndexOf(" ")== -1)
```

```

{
    int row = Character.getNumericValue(history.charAt(2));
    int col = Character.getNumericValue(history.charAt(4));
    int boardValue = Character.getNumericValue(history.charAt(0));

    formattedBoard[row][col] = boardValue;

    // ✅ validate the row/col you actually just wrote
    checkUniqueRows(formattedBoard, row);
    checkUniqueColumns(formattedBoard, col);
}
else
{
    startLastNumber = history.lastIndexOf(" ");

    char row = history.charAt(startLastNumber + 3);
    char col = history.charAt(startLastNumber + 5);
    char boardValue = history.charAt(startLastNumber + 1);

    rowtoInt = Character.getNumericValue(row);
    coltoInt = Character.getNumericValue(col);
    boardValuetoint = Character.getNumericValue(boardValue);

    formattedBoard[rowtoInt][coltoInt] = boardValuetoint;

    // ✅ validate the row/col you actually just wrote
    checkUniqueRows(formattedBoard, rowtoInt);
    checkUniqueColumns(formattedBoard, coltoInt);
}

```

```
}  
}
```

That's it — **no other logic needs to change** for this fix.

(Separately, in `checkUniqueColumns`, you should compute `idx = colIndex / 3` so columns 1,2,4,5,7,8 don't use a stale `idx` — but that's inside `checkUniqueColumns`, not `realTime9x9Fill`.)

This is my new `realTime9x9fill` method You seem to show I have `checkUniquerows` and `columns` in if and else

```
public void realTime9x9Fill(String history) { int row=0; int col=0;
int boardValue=0; int startLastNumber; int rowtoInt=0; int coltoInt=0; int
boardValuetToInt=0; if (history.lastIndexOf(" ")== -1) { row =
Character.getNumericValue(history.charAt(2)); col =
Character.getNumericValue(history.charAt(4)); boardValue=
Character.getNumericValue(history.charAt(0)); formattedBoard [row][col]=boardValue; }
else { startLastNumber=history.lastIndexOf(" "); row =
history.charAt((startLastNumber+3)); col = history.charAt((startLastNumber+5));
boardValue = history.charAt((startLastNumber+1)); rowtoInt =
Character.getNumericValue(row); coltoInt = Character.getNumericValue(col);
boardValuetToInt= Character.getNumericValue(boardValue); formattedBoard
[rowtoInt][coltoInt]=boardValuetToInt; } checkUniqueRows(formattedBoard, row);
checkUniqueColumns(formattedBoard, col); }
```

Yep — **your new version is still wrong**, because in the else branch you **never update row and col to be the *numeric* indices**.

In the else branch you do:

```
row = history.charAt(startLastNumber+3); // <-- this is a CHAR like '6'
```

```
col = history.charAt(startLastNumber+5); // <-- this is a CHAR like '8'
```

So `row` and `col` become the **ASCII code** of that character when stored in an `int`.

Example: if `row` is '6', then `row == 54`, not 6.

Then you call:

```
checkUniqueRows(formattedBoard, row); // row = 54 (bad)
```

```
checkUniqueColumns(formattedBoard, col); // col = 56 (bad)
```

So your checks are running with garbage indices after the first space exists in history.

Minimal correct fix (keep your structure)

Just change the final two calls to use the parsed ints for the else-case.

Option A (cleanest): **track the “last row/col” you wrote** and always call checks with those.

```
public void realTime9x9Fill(String history)
{
    int rowIdx;
    int colIdx;

    if (history.lastIndexOf(" ") == -1)
    {
        rowIdx = Character.getNumericValue(history.charAt(2));
        colIdx = Character.getNumericValue(history.charAt(4));
        int boardValue = Character.getNumericValue(history.charAt(0));

        formattedBoard[rowIdx][colIdx] = boardValue;
    }
    else
    {
        int startLastNumber = history.lastIndexOf(" ");

        rowIdx = Character.getNumericValue(history.charAt(startLastNumber + 3));
        colIdx = Character.getNumericValue(history.charAt(startLastNumber + 5));
        int boardValue = Character.getNumericValue(history.charAt(startLastNumber + 1));

        formattedBoard[rowIdx][colIdx] = boardValue;
    }

    checkUniqueRows(formattedBoard, rowIdx);
}
```

```
    checkUniqueColumns(formattedBoard, colIdx);  
}
```

This ensures you **always** validate the correct row/column for the last written cell.

One more small landmine (important)

This approach assumes your value and coordinates are **single-digit** (0–9), which is true for Sudoku indices and values. If you ever log anything like 10 (not in Sudoku), this parsing breaks.

But for Sudoku, this fix is correct and should stop the weird row/col mismatch that can inflate violations.

all done

Nice — then you're set.

Quick sanity check you can do to confirm it's behaving: add a temporary print right before the checks:

```
System.out.println("last write rowIdx=" + rowIdx + " colIdx=" + colIdx);
```

You should only ever see values 0–8. If you ever see 48–57 (ASCII for '0'..'9'), that would mean the old bug crept back in.

ISSUES DESCRIBED BY CHATGPT

One thing I *would* change (it affects counts + correctness across boards)

Right now, **your column flags are not reset at the start of each new board** in one place. You reset some of them in the `switch(gridNumber)` when `gridNumber==1`, **but that only happens after you finish the first 3×3 grid** (after 9 numbers).

So for the first 9 placements of each new board, `checkUniqueColumns()` runs with flags that could still reflect the previous board attempt.

It won't usually *inflate* your counts — it will more likely **suppress** column logging — but it's still a correctness smell.

- **Anything that's meant to be “once per board attempt” should be reset in `sudokuComplete()`** (because that is your “start new board” boundary).
- **Anything that's meant to be “once per 3×3 row-band inside a board” should stay in your `switch(gridNumber)`** (your row flags).

Move these OUT of `switch(gridNumber)` and INTO `sudokuComplete()`

These are your *column* gating arrays (they should reset at the start of every new board, not after the first 3×3 finishes):

Remove from `switch(gridNumber)` case 1: (delete these lines there)

```
hasProcessedThirdBlockInColumn = new boolean[3];
```

```
hasSecondWithThirdBlockInColumnViolate = new boolean[3];
```

```
hasViolationTwoStandingTopBlocks = new boolean[3];
```

Add into `sudokuComplete()` (right near the top, after you reset `gridNumber`, `uniqueEntries`, etc.)

```
// Reset column logging state per new board attempt
```

```
hasViolationTwoStandingTopBlocks = new boolean[3];
```

```
hasFirstWithThirdBlockInColumnViolate = new boolean[3];
```

```
hasSecondWithThirdBlockInColumnViolate = new boolean[3];
```

```
hasProcessedThirdBlockInColumn = new boolean[3]; // only if you still use it elsewhere
```

Important: you already have this line in `sudokuComplete()`:

```
hasFirstWithThirdBlockInColumnViolate = new boolean[3];
```

Keep it — but make it part of the full “reset all 3 column arrays” block above.

Why move them?

Because right now, your `switch(gridNumber)` only runs when `numberOf3x3Processed == 9`, meaning those column flags don't reset until **after the first full 3×3 grid is placed**, so the first 9 placements of the next board can inherit stale column-state from the previous board.

That won't necessarily inflate counts — but it *will* make logging/matching behavior inconsistent across boards.

Keep these IN the `switch(gridNumber)` (don't move)

These are *row* gating flags that should reset when you start a new **3×3 row band** inside the same board:

Keep in:

- case 4: (start middle band)
- case 7: (start bottom band)

```
hasProcessedThirdBlockInRow=false;
```

```
hasViolationFirstTwoBlocks=false;
```

```
hasFirstWithThirdBlockViolate=false;
```

```
hasSecondWithThirdBlockViolate=false;
```

Those are correctly “per row-band”, not “per board”.