I STILL FOUND IT EXTREMELY DIFFICULT TO UNDERSTAND CHATGPT ADVISE EVEN THOUGH IT ANSWERED MY SAME QUERY AROUND 10 TIMES BUT IN A DIFFERENT MANNER

i know i ask same thing lots times, but Its not easy to understand for me

That's completely ok — this stuff is genuinely hard to "hold in your head", especially when the code is big and stateful.

Here's the simplest way to think about it (and it maps directly to what you're debugging):

The one key idea

You have **two different "lengths"**:

1) uniqueEntries

How many 3×3 grids you've chosen so far for the whole board.

- ranges from **1 to 9**

2) "segment length"

How many grids are relevant to the rule you're checking **right now**.

For **rows**: the segment is "the current band row of 3 blocks"

- top band row uses grids 1–3
- middle band row uses grids 4–6
- bottom band row uses grids 7–9
  So the segment length is **1..3**.

For **columns**: the segment is "the vertical stack"

- stack is (1,4,7) or (2,5,8) or (3,6,9)
  So the segment length is **1..3**.

✅ Your blocked rules like [1, 9, 0, 0, …] are **segment rules**.
They are NOT "whole-board rules".

So when you do this:

numberMatches == uniqueEntries

you are accidentally saying:

"I only consider it a match if the segment is as long as the entire board progress."

That's why columns don't trigger when uniqueEntries is 4/5/6 but the column stack is only length 2.

Concrete example (this is the "aha")

Say you're placing grid **4** (first grid in middle band).

- uniqueEntries = 4 (you've picked 4 grids overall)
- Column stack relevant is **[grid1, grid4]** → length = **2**

You might have:

- copyStoreRetrieved3x3GridColumn = [6, 10, 0]
- blocked rule stored: [6, 10, 0, 0, …]

Now:

- numberMatches can reach **2**
- but it can never reach **4** (because the column segment only has 2 real values)

So the condition numberMatches == uniqueEntries can't be true.

That's the whole issue, in one sentence.

Why your debug prints help

When you print:

- FULL SELECTION (9-wide)
- COLUMN SEGMENT (3-wide, but often only 2 filled)
- BLOCKED COL RULE (9-wide but meaningful prefix is 2 or 3)

You'll *see* that the column segment has only 2 meaningful numbers when uniqueEntries is already 4 or 5.

The fix in plain English

When checking a blocked rule, compare:

- "Did I match the **segment length**?"
  not
- "Did I match the **uniqueEntries**?"

So you need a colLen and a rowLen.

THIS SEEMED LIKE EXCELLENT ADVISE TO CONSIDER

1) Print the *column* segment + the *column* blocked rule too

This level of advise was perfect for screen outputs

Right before the **column** `if(...)` add:

java Copy code

```
System.out.println("FULL SELECTION:  " + Arrays.toString(storeRetrieved3x3Grid));
System.out.println("COLUMN SEGMENT:  " + Arrays.toString(copyStoreRetrieved3x3GridColumn));
System.out.println("BLOCKED COL RULE:" + Arrays.toString(blockedPermutationNumberColumnSequence[h]
System.out.println("uniqueEntries=" + uniqueEntries + " numberMatches=" + numberMatches
        + " hasMatch=" + hasMatchSecondPermutationNumber);
if (uniqueEntries < 9) {
    System.out.println("blocked[next] (index uniqueEntries) = "
            + blockedPermutationNumberColumnSequence[h][uniqueEntries]);
}
```

(Use the correct arrays: **ColumnSequence**, not `blockedPermutationNumberSequence`.)

IT WANTED ME TO USE LOCAL VARIABLE TO GET THE LENGTH OF CopyStore3x3 which held the permutation numbers generated.. But then how would it be possible to navigate index p further down up to uniqueEntries to perform examination if it wanted me to change the delimiter (see its observarions highlighted in blue). I felt at this point, I understood my code better than it understood my intentions

I get why that output makes you doubt it — but it **doesn't contradict** the "use a local length (rowLen/colLen) instead of uniqueEntries" idea.

Why you can see this:

[5, 10, 6, 4, 2, 3, 1, 9, 7] is in full violation with [1, 9, 0, 0, 0, 0, 0, 0, 0]

That message prints **the entire 9-grid selection** (storeRetrieved3x3Grid), but the blocked rule [1,9,0,...] is a **2-grid rule**. It's basically saying:

"Somewhere in the part I'm currently checking, the prefix [1,9] occurred, and I have a rule banning that prefix."

In other words, the violation can be about the **current band row you copied into** copyStoreRetrieved3x3Grid, while you're printing the full 9 anyway. So seeing 9 numbers on the left and only 2 meaningful numbers on the right isn't automatically wrong.

If you want to prove that to yourself, print the candidate you're *actually comparing* at the moment you trigger:

System.out.println("CANDIDATE ROW COPY: " +
Arrays.toString(copyStoreRetrieved3x3Grid));

System.out.println("BLOCKED ROW: " +
Arrays.toString(blockedPermutationNumberSequence[h]));

System.out.println("uniqueEntries=" + uniqueEntries + " numberMatches=" +
numberMatches);

You'll almost certainly see that your row-copy is something like [1, 9, 7, 0, 0, 0, 0, 0, 0] (or [1,9,0,...] depending on when you build it), and that's why it matches the 2-prefix rule.

The real issue (and why columns break)

This line:

beforeUniqueEntryEqualNine =

(numberMatches==uniqueEntries && hasMatchSecondPermutationNumber

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

only works **by accident** for rows *some of the time*, because in your row logic you often arrange things so that:

- when you're checking a 2-block row prefix, uniqueEntries is effectively behaving like 2
- when you're checking a 3-block row prefix, uniqueEntries is effectively behaving like 3

...but for **columns**, that relationship is basically never true.

Example:

- When you place grid **4**, uniqueEntries == 4
- The column "stack" you should compare is only [grid1, grid4] → **length 2**
- Your condition demands numberMatches == 4, which can't happen if your meaningful candidate length is 2

So columns don't fire.

That's why your intuition "in principle there is nothing else getting in the way" feels right — except **the length you're using to declare a full match is wrong for columns**.

One more thing: your row output is also hinting at a fragility

In your snippet you loop:

for (int p = numberMatches; p < uniqueEntries; p++)

...but copyStoreRetrieved3x3Grid often only has **2–3 meaningful values** and the rest are **zeros**.

So once p goes beyond your meaningful length, you start comparing zeros, and your "blocked rule has zeros after the prefix" structure can make that logic behave in surprising ways.

That's another reason splitting the loops (as you said you will) is the clean fix.

The fix in one sentence

For rows and columns, "full violation" should be:

"Did we match the first len values, and is the next blocked value 0?"

Where len is:

- rowLen = 2 or 3 (depending on how many blocks are in that band row)
- colLen = 2 or 3 (depending on how many blocks exist in that vertical stack)

Not uniqueEntries.

If you want a quick sanity check: next time it prints a “full violation”, also print copyStoreRetrieved3x3Grid / copyStoreRetrieved3x3GridColumn and the computed rowLen/colLen. That will make it *visually obvious* why [1,9,0,...] can reject a 9-length global selection without contradiction.

I STILL FOUND IT EXTREMELY DIFFICULT TO UNDERSTAND CHATGPT ADVISE. IT PUT ME IN SITUATION WHERE I HAD TO OBSERVE ALL THE OUTPUTS (AS PER THE

SCREENSHOT ABOVE) AND UNDERSTAND IF MY CODE IS GENUINELY MISSING A SCENARIO.

I thought the row situation made sense all the time: The below has caught the 1,9 in the last row (starting from gridNumber=7). this was my intention [5, 10, 6, 4, 2, 3, 1, 9, 7]is in full violation with [1, 9, 0, 0, 0, 0, 0, 0, 0]

Yes — that specific example **does** make sense and it matches your intention.

What's happening is:

- Your full 9-grid choice is: [5, 10, 6, 4, 2, 3, 1, 9, 7]
- The **last row of 3 blocks** (grids 7–9) is: [1, 9, 7]
- You previously recorded a banned **prefix**: [1, 9] (stored as [1, 9, 0, 0, 0, 0, 0, 0, 0])

So when you start processing the third band row (grids 7–9), your row-copy becomes something like:

- copyStoreRetrieved3x3Grid = [1, 9, 7, 0, 0, 0, 0, 0, 0]

and that prefix [1, 9] matches the banned rule.

So the message:

[5,10,6,4,2,3,1,9,7] is in full violation with [1,9,0,...]

is just a *reporting choice* (printing the whole board selection), not the thing being compared. The actual comparison is on the **band-row copy** (the last-row [1,9,7]).

Where your logic is still inconsistent (and why columns don't work the same)

For rows, in this exact scenario, you got lucky because:

- the "row segment length" you care about is **2**
- your code's "full match" condition uses uniqueEntries
- and at the moment you checked the 3rd row band, uniqueEntries aligned closely enough with your internal numberMatches progression to trip the condition.

But for columns, uniqueEntries *almost never* equals the meaningful column length (2 or 3), so that same "prefix ban" approach won't trigger.

That's why I keep hammering the distinction:

- your **intent** is "ban prefix in the local segment"

- your **code** currently says “prefix length must equal uniqueEntries (global)”
- rows sometimes work anyway because the way you build copyStoreRetrieved3x3Grid and reset numberMatches makes the numbers line up
- columns generally won’t, because the global counter and the local stack length don’t line up

CHATGPT WAS CONVINCING ME ERROR WAS HERE, BUT I KNEW THIS LEVEL OF TESTING SEEMED OK WITH ROWS, IT DIDN’T APPEAR IT NEEDED CHANGING WITH

EXCEPTION OF VARIABLE NAMES WHEN CREATING CODE FOR CHECKING COLUMN SEQUENCES

The problem is here for column... You are right..... if (uniqueEntries<9) { beforeUniqueEntryEqualNine =(numberMatches==uniqueEntries && hasMatchSecondPermutationNumber && blockedPermutationNumberColumnSequence[h][uniqueEntries]==0);

Yep — that's the exact culprit.

For **columns**, uniqueEntries is the count of *all* grids selected so far (1..9), but a **column stack** is only ever length:

- **2** when you're placing in the middle band (needs 1 above)
- **3** when you're placing in the bottom band (needs 2 above)

So numberMatches == uniqueEntries will almost never be true once uniqueEntries > 3.

Fix: use colLen (2 or 3), not uniqueEntries

Right before that block, compute which band you're in and the correct stack length:

```
int g = uniqueEntries;        // 1..9 after you've incremented uniqueEntries
int band = (g - 1) / 3;       // 0=top, 1=middle, 2=bottom
int colLen = (band == 1) ? 2 : (band == 2 ? 3 : 0);
```

Then replace your condition with:

```
if (colLen > 0) {
  beforeUniqueEntryEqualNine =
    (numberMatches == colLen
     && hasMatchSecondPermutationNumber
     && blockedPermutationNumberColumnSequence[h][colLen] == 0);
}
```

And also make sure your loop that uses p doesn't run beyond colLen:

```
for (int p = 0; p < colLen; p++) { ... }
```

That's the full reason it wasn't entering the "full violation for column" block — the equality check was aiming at the wrong "length concept".

If you implement just those two changes (colLen + loop bound), your System.exit(0) in the column-full-violation block should start firing exactly like the row version.

I declared this now as : copyStoreRetrieved3x3GridColumn=new int[3];

Good move — making it new int[3] forces the code to respect the fact a column stack is