How to generate sudoku game boards

by Kristian Colville

Firstly, let me address that I am not a mathematician. I simply have played the game so much that I was bound to have noticed patterns. To help myself create pseudorandom boards I needed to just play sudoku.

Yes, I said pseudorandom. What that means is not truly random. I googled randomness and discovered that I couldn't create a random board with my local machine because eventually, the random numbers would repeat themselves because of the hardware limitations and the seed values given to a computer.

However, I could still create something cool. I quickly began playing sudoku after settling on it as a project idea. Here is what I noticed:

1. When you select a number on the board usually whichever game you play on an app the board highlights all the numbers within one of the nine larger grids containing numbers one to nine.
2. It also highlights all the numbers horizontally and vertically for that number you select.

Given this information, I imagined if I could create a board based on this logic. What I found is yes, it is possible.

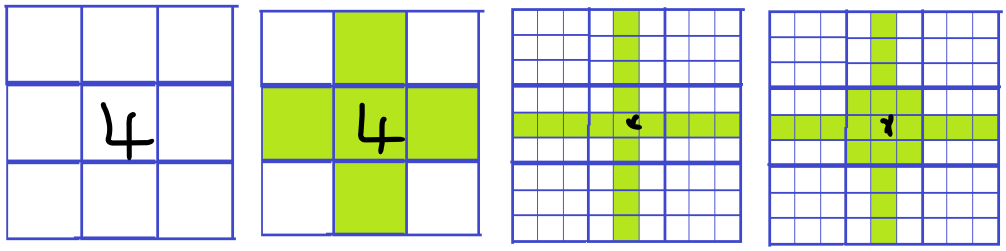
1. Create a first value variable and assign it a random number between 1 and 9.
2. Create a first position variable and assign it a random number between 1 and 81.
3. Place the **first value at the position of the first position.**

This is where the fun begins, all that is needed is one random number on the board. Once that is determined you can use the first two things that I noticed to generate an entire board.

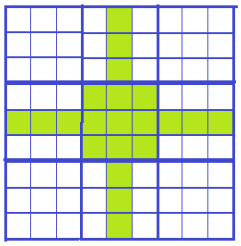
Please don't be fooled, my partner could attest to the number of hours I sat with a painful expression on my face trying to think of how to create this game all by myself. I only began to truly put two and two together when I kept playing the game. Reinventing the wheel is a silly notion and reinventing sudoku is that same concept multiplied by a factor of 81.

My design for the game is relatively straightforward from a high-level perspective, create a number and then create a random position to place that number. I promise you that at a low level it is more complicated. The low level would be the code used.

Once we have placed a number between 1 & 9 on the board, we need the ability for the JavaScript to be able to see all positions within the larger cube containing nine cubes. We also need to give it the ability to know the positions across vertically and horizontally.



The easy fix



Regardless of the first position, it's the same logic around no matter which position is chosen. This helps make the logic for the game only slightly simpler.

To build the game, we have to solve the sudoku board. To me, that is very ironic. I have built a few games already and I can tell you I have not seen that same logic yet.

If you google computer science enough you might eventually stumble upon the term critical thinking; A scientific approach is needed here to solve this puzzle.

What do we know so far?

1. We have 1 number on the board and 80 left to figure out.
2. To build the game we have to solve the board.
3. We don't need complicated math to build it surprisingly.
4. We just need sound logic and careful planning.

What options do we have?

1. Data structures.

I put one option there because that's all we have. It's all we need.

You get what you ask for

If only that was true in life. However, if I give my computer instructions it will carry out those instructions exactly.

“Computer jump” = does not compute.

You can’t ask a computer with your voice to do specifically what you tell it to do (at least not yet).

You can give a computer or more specifically the program used to write your program a set of instructions to carry out. That program is called an ‘integrated development environment’ or IDE for short.

Using the IDE we can create a script in the JavaScript programming language.

The point of what I’m trying to say is that we don’t speak to computers in English so we have to use programming languages and I will keep referring to it as the computer to keep it simpler.

What do I want?

Okay so here’s the logic we want in no specific order from the computer:

1. The computer picks a random number between 1 & 9.
2. It assigns that random value to a position in one of the 9 larger cells.
3. It recognises the larger area surrounding that number for a total of 9 positions.
4. It then recognises all numbers vertically and horizontally from that position bringing the total number of visible positions to 19.
5. The other 62 positions don’t matter because they’re not in the **conflict zone.**
6. The computer then assigns another random value within the scope of the **local zone.**
7. If the next random position is occupied try again until our list of possible random choices is empty for the **local zone.**

There are two options now, fill the conflict zone whilst filling the local zone or just fill the local zone and check the conflict zone.

I prefer the second option. I will fill the local zones while checking the conflict zone.

What do I want for the user?

I want the user to complete the board once I hide certain amounts of pieces depending on the difficulty level.

Data Structures

Okay, so now I must complete the more difficult side of things. I have to figure out how I want to structure my data to interact with it and traverse through it.

At the time of writing this, I have successfully implemented arrays to traverse through the grid. Okay, I'll admit, I failed many times in traversing through the 81 cells. It took many cups of coffee to figure out the best way to traverse the 81 grid cells.

I started with one array with 9 objects. I randomly did this. I then decided before testing it I would use a 2D array instead. I'll be honest it hurt my eyes to look at. I was able to traverse through it alright but only in one direction.

I implemented a function to simulate when the player was in at a certain index and used that for testing my data structure.

This is where I almost gave up.

However, I refused to quit what I had barely started.

Okay, so here's what I discovered:

1. To traverse horizontally the index's must increment by 1.
2. To traverse vertically the index's must increment by 9.
3. To traverse through the larger cells the values horizontally needed to be indexed.

Getting these values is straightforward.

```
// horizontal indexs
let hGrid = [
    [0, 1, 2, 3, 4, 5, 6, 7, 8],
    [9, 10, 11, 12, 13, 14, 15, 16, 17],
    [18, 19, 20, 21, 22, 23, 24, 25, 26],
    [27, 28, 29, 30, 31, 32, 33, 34, 35],
    [36, 37, 38, 39, 40, 41, 42, 43, 44],
    [45, 46, 47, 48, 49, 50, 51, 52, 53],
    [54, 55, 56, 57, 58, 59, 60, 61, 62],
    [63, 64, 65, 66, 67, 68, 69, 70, 71],
    [72, 73, 74, 75, 76, 77, 78, 79, 80]
];

// vertical indexs
let vGrid = [
    [0, 9, 18, 27, 36, 45, 54, 63, 72],
    [1, 10, 19, 28, 37, 46, 55, 64, 73],
    [2, 11, 20, 29, 38, 47, 56, 65, 74],
    [3, 12, 21, 30, 39, 48, 57, 66, 75],
    [4, 13, 22, 31, 40, 49, 58, 67, 76],
    [5, 14, 23, 32, 41, 50, 59, 68, 77],
    [6, 15, 24, 33, 42, 51, 60, 69, 78],
    [7, 16, 25, 34, 43, 52, 61, 70, 79],
    [8, 17, 26, 35, 44, 53, 62, 71, 80]
];
```

The third 2D array is a bit more complicated, but using the first horizontal array you can take the indexes from each array 3 at a time. After about 4 of these, I saw another pattern and just sped up by incrementing the next array values by three from the previous. The grids are going from the left to the right. Starting at the top, then after 3 starting in the middle and so on for the 9 larger grids.

```
// Larger grid indexs 1 - 9
let lGrid = [
    [0, 1, 2, 9, 10, 11, 18, 19, 20],
    [3, 4, 5, 12, 13, 14, 21, 22, 23],
    [6, 7, 8, 15, 16, 17, 24, 25, 26],
    [27, 28, 29, 36, 37, 38, 45, 46, 47],
    [30, 31, 32, 39, 40, 41, 48, 49, 50],
    [33, 34, 35, 42, 43, 44, 51, 52, 53],
    [54, 55, 56, 63, 64, 65, 72, 73, 74],
    [57, 58, 59, 66, 67, 68, 75, 76, 77],
    [60, 61, 62, 69, 70, 71, 78, 79, 80]
];
```

My naive assumptions

Okay, I hit a major roadblock in the project. I'm now writing this almost over a week later from my happy go lucky attitude. I discovered that my logic to solve the board no matter how ingenious in my mind was fundamentally flawed.

Firstly, I presumed that I could create sudoku boards without using math of any kind, except counting to 81. I was wrong, wrong, wrong and oh so wrong. I learned through 3 painful attempts that my logic was not going to work.

Secondly, I wasted 5 days of project building time testing different solutions to solve the board. That was the most heartbreaking experience. I did not reach out to my support systems either and that was foolish.

Finally, I had made a very big mistake. I did not understand that I did not understand the problem. Ironically. My entire notion of solving a board without using math of any kind was ridiculous. I should be ashamed of the nativity. However, I chose to remain humble and admit my mistakes with humility to myself and you as a reader of this.

You need humility

I discovered that sudoku is equally challenging from a computer science perspective as it is from the respective of the player/user of the game.

Once I admitted my bad decisions I chose to ramp up a gear and do this game justice.

My design failed, my tests failed, my logic failed.

But hey that's the journey. You can either quit making a game you are not fully able to comprehend within the time limit of the project and start a new easy game or you can stick to plan A all the way. As an addictive gambler might say, I'm going all in.

I need a plan B for my plan A

Indeed. The title sounds ironic. I'm not giving up on my game, I created a copy yes of someone else's game but I did do it from scratch by myself I didn't use anyone else's backtracking algorithm. I'm not comfortable implementing something into my project I don't comprehend fully. I want to be able to sit in front of an employer and discuss in great detail my capabilities, not someone else's; especially when the entire game depends on the backtracking algorithm.

Employer: Oh you understand what backtracking is?

Me: Yes I understand what it does. I implemented one in sudoku.

Employer: Excellent, excellent. Here's your technical interview.

(If node 'b' cannot touch node 'd' what time does little Timmy finish school?)

Plan B for my plan A

Yeah, I kind of wished I dared to implement this from the beginning, it took me roughly around 3 hours to make between writing down perfect grade 'A' logic and writing the code.

If you made it this far and I have not burnt your eyes reading this here's what I did:

1. I asked myself what would be the quickest way to generate valid sudoku boards and get back onto my schedule. I had wasted 5 precious days of project building on an empty promise to myself.
2. I need to catch up drastically on my workload. There's polishing to be done after all.
3. It turns out the answer is in the question, I needed to get a valid sudoku board.
4. Write a sudoku grid out and solve it, or look up a valid board.
5. To make multiple random boards just swap numbers around with each other.

That's what I did. I created multiple valid sudoku boards by using a valid sudoku board. I can generate **362880** possible different sudoku boards using just one valid sudoku board by rearranging the nine values across the board. That's amazing. Again I wish I dared to implement it this way. I came to that number using math. Ironically. I have 9 values and want to know all possible combinations available using that amount of numbers. The calculation is (9!) = **362880**.

As cool as that sounds I am not happy with this. My mind again aches to do better. It will either be my downfall or my greatest asset.

Plan B part one:

1. Generate some sudoku boards and work on making them engaging and responsive to the user. Another challenging task and effort are required.
2. Polish the website and improve the look and feel of the game.
3. Write up the README file and begin getting users to try to break the game. Any holes need to be filled and catch the momentum and meet the deadline.

Plan B part two:

1. Separate the sudoku board and game logic into different files.
2. Create another branch and use that other branch to test this feature fully.
3. Any leftover time daily after meeting my goals for the rest of the website should be spent learning as much as possible about backtracking algorithms and making tests to see if I can create one successfully.